Серт Серкан, группа 8

Лабораторная работа №2

Вариант№1

Метод Куттера-Джордана-Боссена

относится к группе вероятностных методов стеганографического скрытия, реализующих встраивание битов сообщения в выбранные элементы пространственной области контейнеров изображений, представленных в цветовой модели RGB. Пространство сокрытия в данном методе формируется из значений синих цветовых компонент выбранного множества пикселей контейнера. Для встраивания данных выбирается синий цветовой канал, поскольку изменения в данном канале являются перцептивно наименее заметными.

Цель работы:

Реализовать метод Куттера-Джордана-Боссена. В качестве метрик для оценки искажений заполненных контейнеров . Построить зависимости вероятности ошибок при извлечении скрытых данных от энергии встраиваемого сигнала

Код программы:

```
from PIL import Image
import numpy as np
from dotenv import load_dotenv
import os
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.style as style
import seaborn as sns
import warnings
warnings.filterwarnings(action='once')
encoding: str = 'utf-8'
large = 22
med = 16
small = 12
params = {'axes.titlesize': large,
          'legend.fontsize': med,
          'figure.figsize': (16, 10),
          'axes.labelsize': med,
          'axes.titlesize': med,
          'xtick.labelsize': med,
          'ytick.labelsize': med,
          'figure.titlesize': large}
plt.rcParams.update(params)
plt.style.use('seaborn-v0_8-white')

class Generator:
    def __init__(self, base: int, key: int):
        self.base = base
        self._first_index = 40
        self._second_index = 0
        self._buffer = [i for i in range(abs(key), abs(key) + 60)]
```

```
    # Находим следующий {Xi}
    def next(self) -> int:
        value = (self._buffer[self._second_index] +
self._buffer[self._first_index]) % self.base
        del self._buffer[self._second_index]
        self._buffer.append(value)
        return value

    @property
    def base(self) -> int:
        return self._base

    @base.setter
    def base(self, value: int) -> None:
        if isinstance(value, int):
            if value <= 0:
                raise ValueError('base > 0!')
            self._base = value

class KutterMethod:
    def __init__(self, old_image_path: str, new_image_path: str):
        self.__empty_image_path: str = old_image_path
        self.__full_image_path: str = new_image_path
        self.__lam: float = 1
        self.__sigma: int = 1
        self.__occupancy: int = 0

    @staticmethod
    def str_to_bits(message: str) -> list:
        result = []
        for num in list(message.encode(encoding=encoding)):
            result.extend([(num >> x) & 1 for x in range(7, -1, -1)])
        return result

    @staticmethod
    def bits_to_str(bits: list) -> str:
        chars = []
        for b in range(len(bits) // 8):
            byte = bits[b * 8:(b + 1) * 8]
            chars.append(chr(int(''.join([str(bit) for bit in byte]), 2)))
        return ''.join(chars)

    def embed(self, message: str, key_generator: int):
        img = Image.open(self.__empty_image_path).convert('RGB')
        image = np.asarray(img, dtype='uint8')
        img.close()
        height, width = image.shape[0], image.shape[1]
        message_bits = KutterMethod.str_to_bits(message)
        if len(message_bits) > height * width:
            raise ValueError('Размер сообщения превышает размер
контейнера!')
        # использованные пиксели
        keys = []
        generator = Generator(base=height * width, key=key_generator)
        for bit in message_bits:
            coordinate = generator.next()
            while coordinate in keys:
                coordinate = generator.next()
            keys.append(coordinate)
            i, j = divmod(coordinate, width)
            pixel = image[i, j]
```

```
            lam = self.lam
            L = 0.299 * pixel[0] + 0.587 * pixel[1] + 0.114 * pixel[2]
            if bit == 1:
                pixel_copy = pixel.copy()
                pixel_copy[2] = np.uint8(min(255, pixel[2] + lam * L))
            elif bit == 0:
                pixel_copy = pixel.copy()
                pixel_copy[2] = np.uint8(max(0, pixel[2] - lam * L))
        self.__occupancy = len(message_bits)
        Image.fromarray(image).save(self.__full_image_path, 'PNG')

    def recover(self, key_generator: int) -> str:
        img = Image.open(self.__full_image_path).convert('RGB')
        image = np.asarray(img, dtype='uint8')
        img.close()
        height, width = image.shape[0], image.shape[1]
        keys = []
        generator = Generator(base=height * width, key=key_generator)
        while len(keys) < self.occupancy:
            coordinate = generator.next()
            while coordinate in keys:
                coordinate = generator.next()
            keys.append(coordinate)
        message_bits = []
        for coordinate in keys:
            i, j = divmod(coordinate, width)
            sigma = self.sigma
            summary = 0
            for n in range(1, sigma + 1):
                if 0 <= i - n < height and 0 <= j < width:
                    summary += image[i - n, j, 2]
                if 0 <= i + n < height and 0 <= j < width:
                    summary += image[i + n, j, 2]
                if 0 <= i < height and 0 <= j - n < width:
                    summary += image[i, j - n, 2]
                if 0 <= i < height and 0 <= j + n < width:
                    summary += image[i, j + n, 2]
            if image[i, j, 2] > (summary / (4 * sigma)):
                message_bits.append(1)
            else:
                message_bits.append(0)
        recovered_message = KutterMethod.bits_to_str(message_bits)
        return recovered_message

    @property
    def sigma(self) -> int:
        return self.__sigma

    @sigma.setter
    def sigma(self, value: int) -> None:
        if isinstance(value, int):
            if value <= 0:
                raise ValueError('sigma > 0!')
            self.__sigma = value

    @property
    def lam(self) -> float:
        return self.__lam

    @lam.setter
    def lam(self, value: float) -> None:
```

```
        if isinstance(value, float):
            if abs(value) < 1E-14:
                raise ValueError('lambda > 0!')
            self.__lam = value

    @property
    def occupancy(self) -> int:
        return self.__occupancy

def error_probability_analysis(message: str, key ,old_image: str,
new_image: str):
    lam_values = []  # Değişen lambda (lam) değerleri
    sigma_values = []  # Değişen sigma değerleri
    error_probabilities = []  # Hata olasılıkları
    message_bits = np.asarray(KutterMethod.str_to_bits(message))
    for lam in (0.5, 1, 1.5, 2, 2.5, 3):  # İstenilen lambda değerlerini
belirleyin
        kutter = KutterMethod(old_image, new_image)
        kutter.lam = lam
        kutter.embed(message, key)
        for sigma in (1, 2, 3, 4, 5, 6, 7):
            kutter.sigma = sigma  # Sigma değerini ayarla
            kutter.embed(message, key)  # 'key_range' değişkenini kullan
            recovered_message = kutter.recover(key)
            error_count = sum(1 for m1, m2 in zip(message,
recovered_message) if m1 != m2)
            error_probability = error_count / len(message)
            lam_values.append(lam)
            sigma_values.append(sigma)
            error_probabilities.append(error_probability)
    return lam_values, error_probabilities, sigma_values

def dependence(key: int, old_image: str, new_image: str, message: str):
    d = dict()
    message_bits = np.asarray(KutterMethod.str_to_bits(message))
    for lam in (0.5, 1, 1.5, 2, 2.5, 3):
        kutter = KutterMethod(old_image, new_image)
        kutter.lam = lam
        kutter.embed(message, key)
        for sigma in (1, 2, 3, 4, 5, 6, 7):
            kutter.sigma = sigma
            recovered_message = kutter.recover(key)
            recovered_message_bits =
np.asarray(KutterMethod.str_to_bits(recovered_message))
            d.setdefault('lambda', []).append(lam)
            d.setdefault('sigma', []).append(sigma)
            d.setdefault('e_probability', []).append(
                np.mean(np.abs(message_bits -
recovered_message_bits[:message_bits.shape[0]])) * 100)
    df = np.round(pd.DataFrame(d), decimals=2)
    df.to_csv('log.csv', sep='\t', encoding=encoding)
    print('Tablo:')
    print(df)
    print('Korelasyon:')
    print(np.round(df.corr(), decimals=2))

    df.groupby('lambda')['e_probability'].mean().plot(kind='bar',
grid=True, ylim=0)
    plt.show()
    df.groupby('sigma')['e_probability'].mean().plot(kind='bar', grid=True,
ylim=0)
```

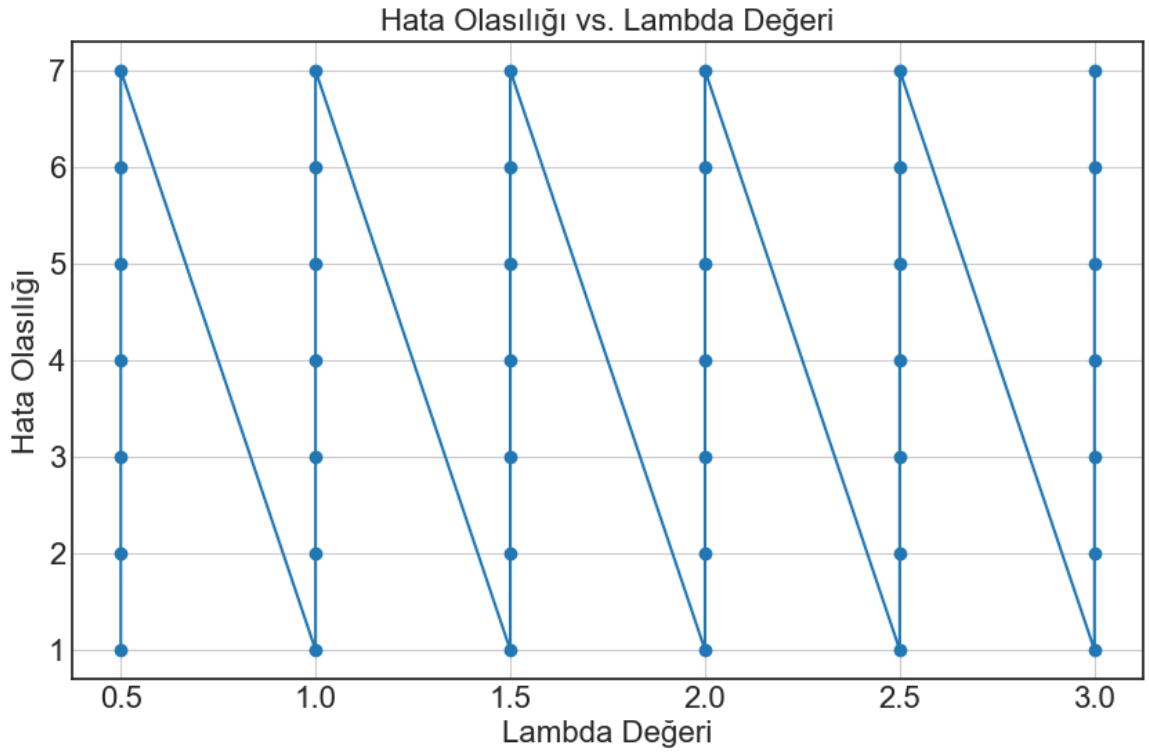
```
    plt.show()



def metrics(empty_image: str, full_image: str) -> None:
    img = Image.open(empty_image).convert('RGB')
    empty = np.asarray(img, dtype='uint8')
    img.close()
    img = Image.open(full_image).convert('RGB')
    full = np.asarray(img, dtype='uint8')
    img.close()
    max_d = np.max(np.abs(empty.astype(int) - full.astype(int)))
    epsilon = 1e-10  # A small constant to avoid division by zero
    SNR = np.sum(empty * empty) / (np.sum((empty - full) ** 2) + epsilon)
    H, W = empty.shape[0], empty.shape[1]
    MSE = np.sum((empty - full) ** 2) / (W * H)
    sigma = np.sum((empty - np.mean(empty)) * (full - np.mean(full))) / (H
* W)
    UQI = (4 * sigma * np.mean(empty) * np.mean(full)) / \
          ((np.var(empty) ** 2 + np.var(full) ** 2) * (np.mean(empty) ** 2
+ np.mean(full) ** 2))
    print('Максимальное абсолютное отклонение:{}'.format(max_d))
    print('Отношение сигнал-шум:{}'.format(SNR))
    print('Среднее квадратичное отклонение:{}'.format(MSE))
    print(f'Универсальный индекс качества (УИК):{UQI}\n')

def main():
    load_dotenv('.env')
    key= 1500
    old_image = 'input/s.png'
    new_image = 'output/S_new.png'
    with open('message.txt', mode='r', encoding=encoding) as file:
        message = file.read()
    lam_values, sigma_values, error_probabilities =
error_probability_analysis(message, key, old_image, new_image)
    # Verileri bir veri çerçevesine ekleyin
    data = {'Lambda': lam_values, 'Sigma': sigma_values, 'Error
Probability': error_probabilities}
    df = pd.DataFrame(data)
    # Verileri CSV dosyasına kaydedin
    df.to_csv('error_analysis.csv', index=False)
    plt.figure(figsize=(10, 6))
    plt.plot(lam_values, error_probabilities, marker='o', linestyle='-')
    plt.title('Hata Olasılığı vs. Lambda Değeri')
    plt.xlabel('Lambda Değeri')
    plt.ylabel('Hata Olasılığı')
    plt.grid(True)
    plt.show()
    kutter = KutterMethod(old_image, new_image)
    kutter.embed(message, key)
    recovered_message = kutter.recover(key)
    print('Ваше сообщение:{}'.format(recovered_message))
    print(message)
    metrics(old_image, new_image)
    dependence(key, old_image, 'output/imageS.png', message)
if __name__ == '__main__':
    main()
```

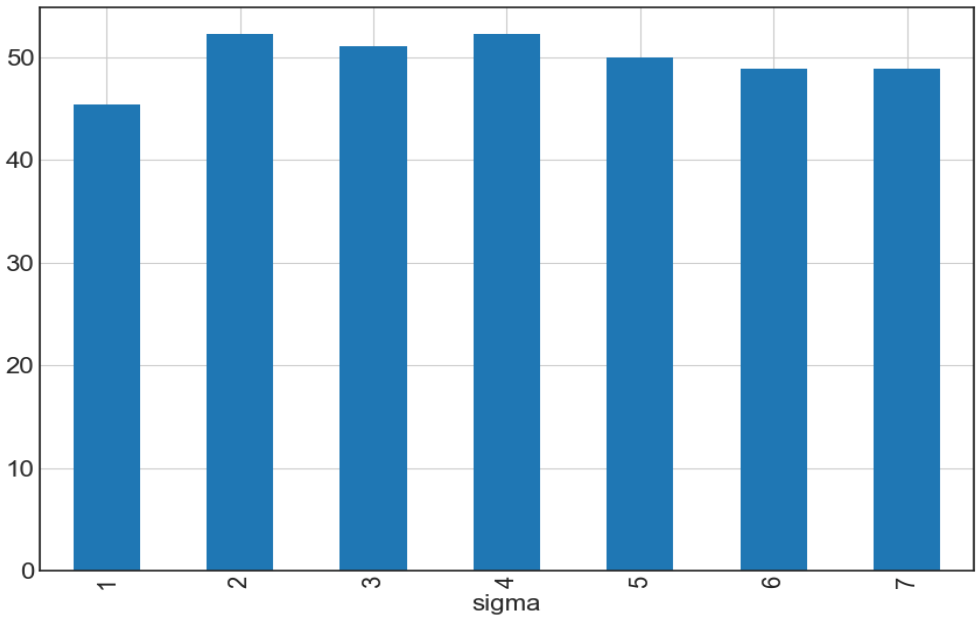
Результат работы программы:

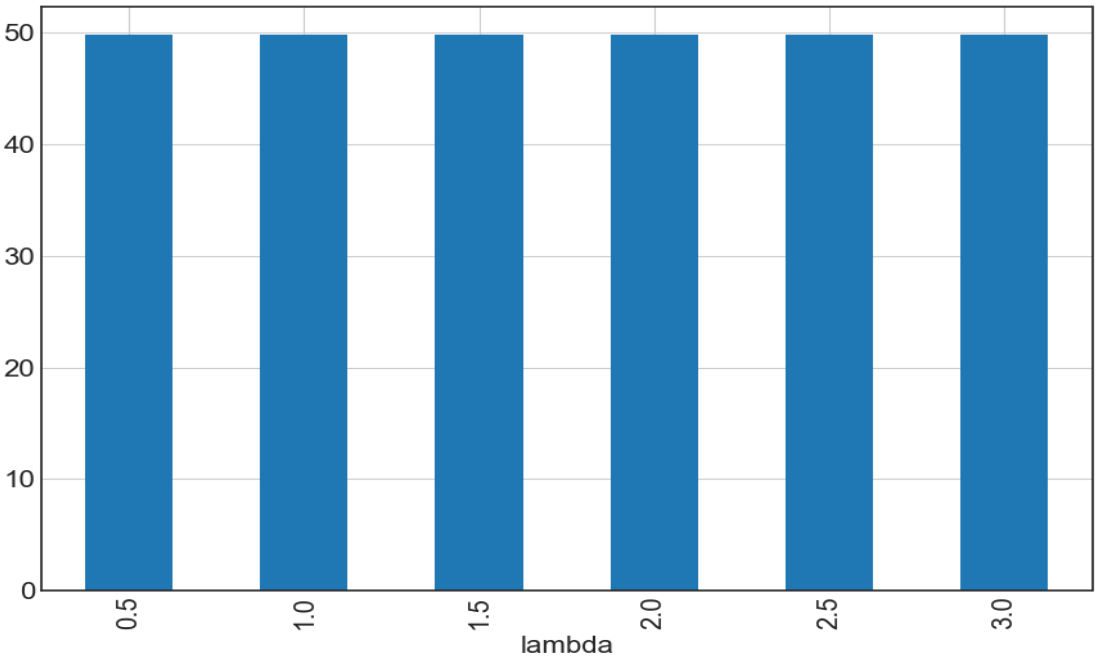
Исходное изображение (пустой контейнер)

Изображение со встроенным сообщением (заполненный контейнер)

Результаты работы программы (введено сообщение «My secret messages»):








```
Ваше сообщение:▯▯i8▯o▯▯▯éL
Serkan SERT
Максимальное абсолютное отклонение:0
Отношение сигнал-шум:3.1246418e+17
Среднее квадратичное отклонение:0.0
Универсальный индекс качества (УИК):0.0007731913502116144
```

Korelasyon:

	lambda	sigma	e_probability
lambda	1.0	-0.00	-0.00
sigma	-0.0	1.00	0.07
e_probability	-0.0	0.07	1.00