Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

Кафедра ПМиК

Лабораторная работа 3
по дисциплине «Прикладная стеганография»

Выполнил: ст. гр.
ЗМП-41 Лёвкин И. А.

Проверила: Мерзлякова Е.Ю.

Новосибирск 2025

Обзор реверсивных стеганографических методов, основанных на интерполяции

Реверсивные стеганографические методы, основанные на интерполяции изображений, позволяют скрывать информацию с возможностью полного восстановления исходного изображения после извлечения данных. Эти методы сочетают высокую вместимость (емкость внедрения) и хорошее визуальное качество стегоизображения. Рассмотрим ключевые методы, представленные в предложенных работах:

1. **Jung и Yoo (NMI, Neighbor Mean Interpolation)**
    - **Описание**: метод использует интерполяцию для увеличения изображения, после чего секретные данные внедряются в разницу между интерполированными и исходными пикселями.
    - **Преимущества**: высокая скорость вычислений, хорошее качество изображения (PSNR > 35 дБ).
    - **Недостатки**: ограниченная вместимость по сравнению с более современными методами.
2. **INP (Interpolation by Neighboring Pixel)**
    - **Описание**: улучшенная версия метода Jung, где для интерполяции используются соседние пиксели, что позволяет увеличить объем внедряемых данных.
    - **Преимущества**: большая вместимость, сохранение высокого PSNR.
    - **Недостатки**: сложность вычислений выше, чем у NMI.
3. **NIE (New Interpolation Expansion)**
    - **Описание**: основан на расширении изображения с использованием новых алгоритмов интерполяции, что повышает качество стегоизображения.
    - **Преимущества**: лучшее визуальное качество и высокая безопасность.
    - **Недостатки**: требует больше вычислительных ресурсов.
4. **Метод Нагиевой и Вердиева**
    - **Описание**: использует интерполяцию для создания контейнера, а затем внедряет данные, преобразуя разницу пикселей в двоичную систему.
    - **Преимущества**: очень высокая вместимость и PSNR > 38 дБ.
    - **Недостатки**: Сложность реализации из-за необходимости преобразования данных.
5. **Sabeen Govind (ENMI, Enhanced Neighbor Mean Interpolation)**
    - **Описание**: улучшенная версия NMI с двухэтапной схемой внедрения.
    - **Преимущества**: большая вместимость.
    - **Недостатки**: снижение визуального качества по сравнению с другими методами.

Для реализации был выбран метод INP по следующим причинам**:**

- INP обеспечивает высокую вместимость при сохранении высокого PSNR > 36.
- Метод просто в реализации в отличие от NIE или метода Нагиевой-Вердиева.
- Может быть адаптирован для работы с различными типами изображений.

Алгоритм шифрования:

1. Загрузка изображения-контейнера.
2. Уменьшение изображения.
3. Интерполяция для восстановления размера с использованием соседних пикселей.
4. Преобразование сообщения в бинарный формат.
5. Внедрение битов в разницу между интерполированными и исходными пикселями.

Алгоритм дешифрования:

1. Загрузка изображения-контейнера.
2. Чтение пикселей для восстановления сообщения.
3. Преобразование бинарного формата в сообщение.

Оценка алгоритма

Проведём встраивание данных в изображение и оценим ёмкость и величину искажения PSNR на 8-битном изображении с палитрой из оттенков серого. Встраиваемый текст на английском языке и составляет размер 16 КБ. Размер контейнера в свою очередь составляет 257 КБ.

Вывод программы:

```
Embed bits: 130576
Capacity (bit/pixel): 0.4981
PSNR: 38.56 dB
```

Здесь можно видеть, что ёмкость встраивания составляет 0.4981 бит на пиксель.

Мы встроили текст размером примерно в 50% от максимальной вместимости контейнера. В таком случае PSNR составляет 38.56 ДБ, при таком значении искажения могут быть заметны при внимательном расммотрении изображения.

Рис. 1. Контейнер без сообщения

Рис. 2. Контейнер после встраивания сообщения

Запустим программу на наборе из 10 контейнеров

20.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.56 dB	25.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.56 dB
21.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.56 dB	26.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.56 dB
22.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 39.51 dB	27.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.57 dB
23.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.88 dB	28.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.76 dB
24.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.61 dB	29.bmp Embed bits: 16322.0 Capacity (bit/pixel): 0.4981 PSNR: 38.66 dB

Листинг

```
import argparse
import math
from pathlib import Path
import sys
import numpy as np
from PIL import Image


import rdh


ROOT_DIR = Path(__file__).resolve().parent.parent


if str(ROOT_DIR) not in sys.path:
    sys.path.append(str(ROOT_DIR))


import utils.stego as stego


def main():
    parser = argparse.ArgumentParser(description="RDH Stenography for 8-
bit BMP images")


    parser.add_argument("-m", "--message", required=True, help="Message
file")
    parser.add_argument("-i", "--input", required=True, help="Input BMP
image")
    parser.add_argument("-o", "--output", required=True, help="Output
stego image")


    args = parser.parse_args()
```

```
    message = open(args.message, "r", encoding="utf-8").read()
    secret_binary = rdh.text_to_bits(message)

    input_img = Image.open(args.input).convert("L")

    full_img = np.array(input_img)
    small_img = rdh.downscale_image(full_img)
    cover_img = rdh.upscale_inp(small_img)

    stego_img, embedded_bits = rdh.embed_secret(cover_img, secret_binary)
    rdh.save_image(stego_img, args.output)

    recovered_bits = rdh.extract_secret(
        stego_img, cover_img.astype(np.uint8), embedded_bits
    )
    output_text = rdh.bits_to_text(recovered_bits)

    psnr_val = stego.psnr(cover_img, stego_img, input_img.mode)
    capacity = embedded_bits / (full_img.shape[0] * full_img.shape[1])

    print("Встраивание завершено.")
    print(f"Встроено бит: {embedded_bits}")
    print(f"Емкость (бит/пиксель): {capacity:.4f}")
    print(f"PSNR: {psnr_val:.2f} dB")
    print(f"Извлечённый текст: {output_text}")
    print(f"Совпадает: {output_text == message}")

if __name__ == "__main__":
```

```
    main()

import numpy as np
from PIL import Image

def text_to_bits(text, encoding="utf-8") -> str:
    return "".join(format(byte, "08b") for byte in text.encode(encoding))

def bits_to_text(bits, encoding="utf-8") -> str:
    chars = [bits[i : i + 8] for i in range(0, len(bits), 8)]
    byte_array = bytearray(int(b, 2) for b in chars if len(b) == 8)
    return byte_array.decode(encoding, errors="ignore")

def load_image(path):
    img = Image.open(path).convert("L")
    return np.array(img)

def save_image(image_array, path):
    img = Image.fromarray(np.clip(image_array, 0, 255).astype(np.uint8))
    img.save(path)

def downscale_image(img):
    return img[::2, ::2]

def upscale_inp(original):
    h, w = original.shape
    new_h, new_w = h * 2 - 1, w * 2 - 1
    result = np.zeros((new_h, new_w), dtype=np.uint8)
```

```
    for i in range(h):
        for j in range(w):
            result[2 * i, 2 * j] = original[i, j]

    for i in range(0, new_h, 2):
        for j in range(1, new_w, 2):
            left = result[i, j - 1]
            right = result[i, j + 1] if j + 1 < new_w else left
            result[i, j] = (int(left) + int(right)) // 2

    for i in range(1, new_h, 2):
        for j in range(0, new_w, 2):
            top = result[i - 1, j]
            bottom = result[i + 1, j] if i + 1 < new_h else top
            result[i, j] = (int(top) + int(bottom)) // 2

    for i in range(1, new_h, 2):
        for j in range(1, new_w, 2):
            tl = result[i - 1, j - 1]
            tr = result[i - 1, j + 1] if j + 1 < new_w else tl
            bl = result[i + 1, j - 1] if i + 1 < new_h else tl
            br = result[i + 1, j + 1] if (i + 1 < new_h and j + 1 <
new_w) else tl
            result[i, j] = (int(tl) + int(tr) + int(bl) + int(br)) // 4

    return result

def get_code_and_index(d, k):
    M = 2 ** (k - 1)
    if d < M:
```

```
        return 0, d - M  # center
    else:
        return 1, d - M - (2 ** (k - 1))  # shifted for index 1

def get_symbol_from_code(index, code, k):
    M = 2 ** (k - 1)
    if index == 0:
        return code + M
    else:
        return code + M + (2 ** (k - 1))

def embed_secret(cover, secret_bits, k=4):
    h, w = cover.shape
    stego = cover.copy().astype(np.int16)
    i = 0
    embedded_bits = 0

    for y in range(0, h - 2, 4):
        for x in range(0, w - 2, 4):
            if i + 4 * k > len(secret_bits):
                return np.clip(stego, 0, 255).astype(np.uint8),
embedded_bits

            symbols = []
            for j in range(4):
                start = i + j * k
                end = i + (j + 1) * k
                if end > len(secret_bits):
                    bits = secret_bits[start:] + "0" * (end -
len(secret_bits))
```

```
                    symbols.append(int(bits, 2))
                    embedded_bits += len(secret_bits[start:])
                    i = len(secret_bits)
                    break
                else:
                    bits = secret_bits[start:end]
                    symbols.append(int(bits, 2))
                    embedded_bits += k
            else:
                i += 4 * k

            codes = []
            indexes = []
            for s in symbols:
                index, code = get_code_and_index(s, k)
                codes.append(code)
                indexes.append(index)

            index_bin = "".join(map(str, indexes))
            I = int(index_bin, 2)
            M = 2**k
            Pc = int(cover[y + 2, x + 2])
            stego[y + 2, x + 2] = Pc + (I - M)

            positions = [(y, x + 1), (y + 1, x), (y + 1, x + 2), (y + 2,
x + 1)]
            for pos, code in zip(positions, codes):
                stego[pos] += code

    return np.clip(stego, 0, 255).astype(np.uint8), embedded_bits
```

```
def extract_secret(stego, cover, total_bits, k=4):
    h, w = stego.shape
    recovered_bits = []
    count = 0

    for y in range(0, h - 2, 4):
        for x in range(0, w - 2, 4):
            if count >= total_bits:
                break

            positions = [(y, x + 1), (y + 1, x), (y + 1, x + 2), (y + 2,
x + 1)]
            codes = [int(stego[pos]) - int(cover[pos]) for pos in
positions]

            Pc = int(cover[y + 2, x + 2])
            Pc_ = int(stego[y + 2, x + 2])
            I = Pc_ - Pc + 2**k
            I = max(0, min(15, I))
            index_bin = format(I, f"0{4}b")[-4:]
            indexes = list(map(int, index_bin))

            for idx, code in zip(indexes, codes):
                if count >= total_bits:
                    break
                s = get_symbol_from_code(idx, code, k)
                s = max(0, min(2**k - 1, s))
                recovered_bits.append(format(s, f"0{k}b"))
                count += k
```

```
    return "".join(recovered_bits)[:total_bits]
```