Sprawozdanie
Inżynieria Obrazów
Laboratorium 2

Katsiaryna Kolyshko 276708

Dr. Inż. Jan Nikodem

Wstęp Teoretyczny

Stenografia obrazowa (ang. image steganography) to dziedzina kryptografii ukrytej, która zajmuje się ukrywaniem informacji w obrazach cyfrowych w sposób niewidoczny dla ludzkiego oka. W przeciwieństwie do kryptografii, która szyfruje wiadomości czyniąc je nieczytelne, stenografia ukrywa sam fakt istnienia tajnej komunikacji.

Podstawy teoretyczne

Obrazy cyfrowe składają się z pikseli, gdzie każdy piksel w modelu RGB jest reprezentowany przez trzy wartości (czerwony, zielony, niebieski) w zakresie od 0 do 255. Ta struktura stwarza idealne środowisko do ukrywania dodatkowych danych poprzez wprowadzanie subtelnych modyfikacji, które nie wpływają znacząco na wygląd obrazu.

Główne metody stenografii obrazowej

1. **Metoda LSB (Least Significant Bit)**

Najbardziej popularna metoda ze względu na prostotę i skuteczność:
- Polega na zastępowaniu najmniej znaczących bitów wartości pikseli bitami ukrywanej wiadomości
- Modyfikacja LSB powoduje zmianę wartości piksela maksymalnie o 1, co jest praktycznie niewidoczne
- Przykładowo: wartość 200 (11001000) z zastąpionym LSB może zmienić się na 201 (11001001)

2. **Metody domenowe**

Bardziej zaawansowane techniki wykorzystujące transformacje:
- DCT (Discrete Cosine Transform) - modyfikacja współczynników transformaty kosinusowej
- DWT (Discrete Wavelet Transform) - ukrywanie informacji w różnych pasmach częstotliwościowych
- Te metody są bardziej odporne na kompresję i przetwarzanie obrazu

3. **Metody statystyczne**

Wykorzystują właściwości statystyczne obrazu:
- Modyfikacja histogramu obrazu
- Użycie technik rozpraszania widma (Spread Spectrum)
- Wykazują wyższą odporność na analizy statystyczne

Aspekty inżynierii obrazów w stenografii

Pojemność ukrywania
Zależy od kilku czynników:
- Rozdzielczość obrazu (więcej pikseli = więcej miejsca na ukryte dane)
- Użyta metoda (LSB pozwala ukryć więcej danych niż metody domenowe, ale jest mniej odporna)
- Wymagana jakość wizualna (mniejsza tolerancja zniekształceń = mniejsza pojemność)

Niewykrywalność

Kluczowy aspekt stenografii:
- Analiza wizualna - obraz nie powinien wykazywać widocznych artefaktów
- Analiza statystyczna - rozkład wartości pikseli nie powinien odbiegać od typowego
- Analiza histogramu - nie powinno być widocznych anomalii w histogramie

Odporność
Określa zdolność ukrytych danych do przetrwania:
- Kompresji obrazu (szczególnie stratnej jak JPEG)
- Operacji przetwarzania obrazu (filtry, przeskalowanie)
- Intencjonalnych ataków steganalitycznych

Procedura implementacji stenografii LSB

Ukrywanie wiadomości
1. Przygotowanie nośnika:
    - Wybór obrazu o odpowiedniej złożoności i rozmiarze
    - Preferowane są obrazy z dużą ilością szczegółów i różnorodnymi kolorami
2. Przygotowanie wiadomości:
    - Konwersja wiadomości na ciąg bitów
    - Ewentualne szyfrowanie przed ukryciem (podwójna ochrona)
3. Ukrywanie:
    - Iteracja przez piksele obrazu
    - Zastępowanie LSB wartości R, G lub B odpowiednimi bitami wiadomości
    - Zachowanie metadanych o długości wiadomości (ukryte w początkowych pikselach)
4. Zapisywanie:
    - Zapisanie zmodyfikowanego obrazu w formacie bezstratnym (PNG, BMP)
    - Unikanie formatów z kompresją stratną (np. JPEG)

Odzyskiwanie wiadomości
1. Odczytanie metadanych:
    - Ekstrakcja informacji o długości ukrytej wiadomości
2. Ekstrakcja bitów:
    - Odczytanie LSB z wartości R, G, B kolejnych pikseli
    - Składanie odczytanych bitów w bajty wiadomości
3. Rekonstrukcja:
    - Konwersja wyodrębnionych bitów na oryginalną formę wiadomości
    - Ewentualne odszyfrowanie, jeśli wiadomość była dodatkowo zabezpieczona

Wyzwania i ograniczenia
1. Kompromis między pojemnością a niewykrywalnością:
    - Zwiększanie ilości ukrytych danych zwiększa ryzyko wykrycia
2. Wrażliwość na przetwarzanie obrazu:
    - Kompresja stratna może zniszczyć ukryte dane
    - Nawet proste operacje jak zmiana jasności mogą wpłynąć na ukrytą wiadomość
3. Rozwój stegoanalizy:
    - Coraz doskonalsze metody wykrywania stenografii
    - Konieczność stosowania bardziej wyrafinowanych technik ukrywania

Zastosowania

1. Bezpieczna komunikacja:
    - Przekazywanie tajnych informacji w sytuacjach monitoringu
    - Omijanie cenzury w krajach o ograniczonej wolności komunikacji
2. Znaki wodne:
    - Ochrona własności intelektualnej
    - Oznaczanie autorstwa obrazów cyfrowych
3. Medycyna:
    - Ukrywanie danych pacjenta w obrazach medycznych
    - Zapewnienie prywatności przy jednoczesnym zachowaniu kontekstu
4. Bezpieczeństwo danych:
    - Dodatkowa warstwa ochrony dla krytycznych informacji
    - Ukrywanie kluczy kryptograficznych

Zadanie 1

Korzystając z podanych wyżej funkcji ukryć dowolną wiadomość w dowolnym obrazku.

Na oparciu mieliśmy kod z punktu „Ukryjmy tekst w obrazku”.

Oryginalny kod z zadania używa prostej metody LSB (Least Significant Bit) bezpośrednio na wartościach pikseli obrazu, natomiast program zawiera bardziej zaawansowaną implementację stenografii w domenie DCT (Discrete Cosine Transform) wykorzystywanej w formacie JPEG.

Użyliśmy metody opartej na DCT (Discrete Cosine Transform) zamiast klasycznej metody LSB z kilku powodów:

1. Odporność na kompresję JPEG - Standardowa metoda LSB działa bezpośrednio na wartościach pikseli, przez co ukryta informacja zostaje całkowicie zniszczona podczas kompresji JPEG. DCT działa w tej samej domenie co kompresja JPEG, więc ukryta wiadomość ma znacznie większe szanse na przetrwanie.

2. Większa niewykrywalność - Modyfikacja współczynników DCT jest trudniejsza do wykrycia przez metody stegoanalizy niż bezpośrednia zmiana LSB pikseli, ponieważ wpływa na charakterystykę częstotliwościową obrazu, a nie na bezpośrednie wartości pikseli.

3. Kompatybilność z JPEG - Widzimy w kodzie, że program zapisuje obrazy również jako JPEG. Klasyczna metoda LSB jest całkowicie nieskuteczna przy zapisie do formatu JPEG, ponieważ kompresja JPEG znacząco zmienia wartości pikseli, niszcząc ukryte informacje.

4. Lepsza integracja z procesem przetwarzania JPEG - Wykorzystanie tych samych mechanizmów co JPEG (bloki 8x8, przestrzeń kolorów YCrCb, transformata DCT) pozwala na bardziej naturalne wkomponowanie ukrytych danych w strukturę obrazu.

Program nie będzie działać efektywnie z metodą LSB dla plików JPEG. Jest to związane z fundamentalną naturą kompresji JPEG:

1. Dla plików PNG (format bezstratny): Metoda LSB działa doskonale, ponieważ format PNG zachowuje dokładnie te same wartości pikseli po zapisaniu. Ukryta wiadomość pozostanie nienaruszona.

2. Dla plików JPEG (format stratny): Metoda LSB jest praktycznie bezużyteczna, ponieważ proces kompresji JPEG znacząco modyfikuje wartości pikseli. Nawet niewielka kompresja JPEG prawie zawsze zniszczy informacje ukryte metodą LSB.

Dlatego, jeśli chcemy używać wyłącznie plików PNG, metoda LSB jest wystarczająca i prostsza w implementacji. Jeśli potrzebna obsługa plików JPEG, konieczne jest zastosowanie metody działającej w domenie DCT, która modyfikuje współczynniki DCT przed kompresją JPEG.

Jak działa program:

Program pyta użytkownika, czy on chcę zakodować czy odkodować tekst z foto.

```
===== JPEG LSB Steganography =====
1. Hide a message in an image
2. Extract a message from an image
3. Exit

Enter your choice (1-3):
```

Następnie, jak jest wybrana opcja 1, na przykład, pyta osobę o pliku i o kodowanym tekście.

Niżej są przykładowe działania programu:

```
Enter your choice (1-3): 1
Enter the path to the input image: input.jpg
Enter the path for the output image: output.jpg
Enter the message to hide: LA PASSION

Processing...
Original image dimensions: (981, 736, 3)
Message length in bits: 80
Successfully encoded all 80 bits
Message verified in DCT domain: LA PASSION
Image saved to output.jpg

Message successfully hidden!
Remember this number for extraction: 80 bits

===== JPEG LSB Steganography =====
1. Hide a message in an image
2. Extract a message from an image
3. Exit
```

Original Image

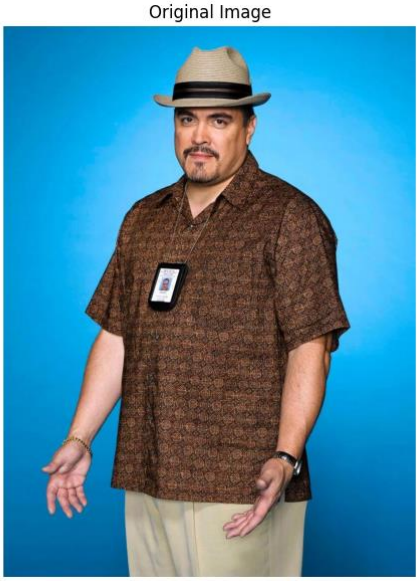
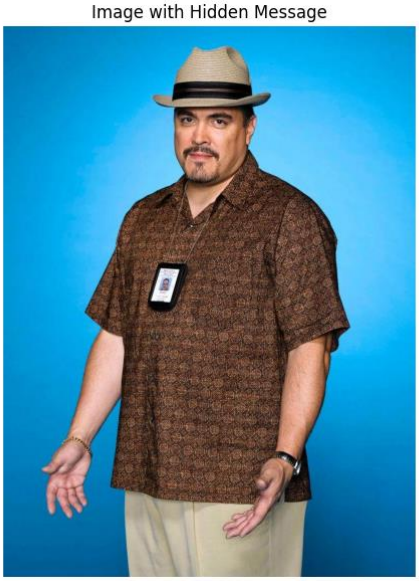


Image with Hidden Message



Teraz spróbujmy odkodować foto.

```
===== JPEG LSB Steganography =====
1. Hide a message in an image
2. Extract a message from an image
3. Exit

Enter your choice (1-3): 2
Enter the path to the image with hidden message: output.jpg
Enter the message length in bits: 80

Extracting message...

Extracted message: LA PASSION
```

Jak widać, program działa poprawnie.

Czym są różnice z przykładowym kodem:

1. Metoda ukrywania: z prostego LSB na wartościach pikseli na modyfikację współczynników DCT
2. Reprezentacja obrazu: z RGB na przestrzeń YCrCb (lepiej dostosowaną do kompresji JPEG)
3. Przetwarzanie: dodano transformatę kosinusową (DCT) i jej odwrotność (IDCT)
4. Struktura danych: dodano operacje na blokach 8x8 (charakterystyczne dla JPEG)
5. Interfejs: dodano interaktywne menu zamiast bezpośredniego wywoływania funkcji

Opis kodu

Niżej jest krótki opis funkcji z programu:

- **dct2/idct2**: Wykonują dwuwymiarową transformatę kosinusową i odwrotną, przekształcając dane z domeny przestrzennej do częstotliwościowej i odwrotnie.
- **split_channel_to_blocks/merge_blocks_to_channel**: Dzielą kanał obrazu na bloki 8x8 i łączą bloki z powrotem w kanał, co jest kluczowe dla przetwarzania JPEG.

- **encode_as_binary_array/decode_from_binary_array**: Konwertują tekst na ciąg bitów i odwrotnie, umożliwiając ukrywanie tekstowych wiadomości.
- **hide_message**: Ukrywa wiadomość binarną w blokach DCT, modyfikując LSB niezerowych współczynników, co minimalizuje widoczne zmiany w obrazie.
- **reveal_message**: Odczytuje ukrytą wiadomość z bloków DCT, wyodrębniając LSB z tych samych współczynników.
- **y_to_dct_blocks/dct_blocks_to_y**: Przekształcają kanał Y obrazu do bloków DCT z kwantyzacją i odwrotnie, symulując proces kompresji JPEG.
- **load_image/save_image**: Obsługują wczytywanie i zapisywanie obrazów z odpowiednią konwersją przestrzeni kolorów.
- **embed_message_in_image**: Główna funkcja ukrywająca wiadomość - konwertuje obraz do YCrCb, przetwarza kanał Y przez DCT, ukrywa wiadomość i składa obraz z powrotem.
- **extract_message_from_image**: Wykonuje proces odwrotny do ukrywania - wczytuje obraz, konwertuje do YCrCb, przetwarza przez DCT i odczytuje ukrytą wiadomość.
- **main**: Interaktywne menu pozwalające użytkownikowi wybrać między ukrywaniem wiadomości, jej odczytywaniem lub wyjściem z programu.

Zadanie 2

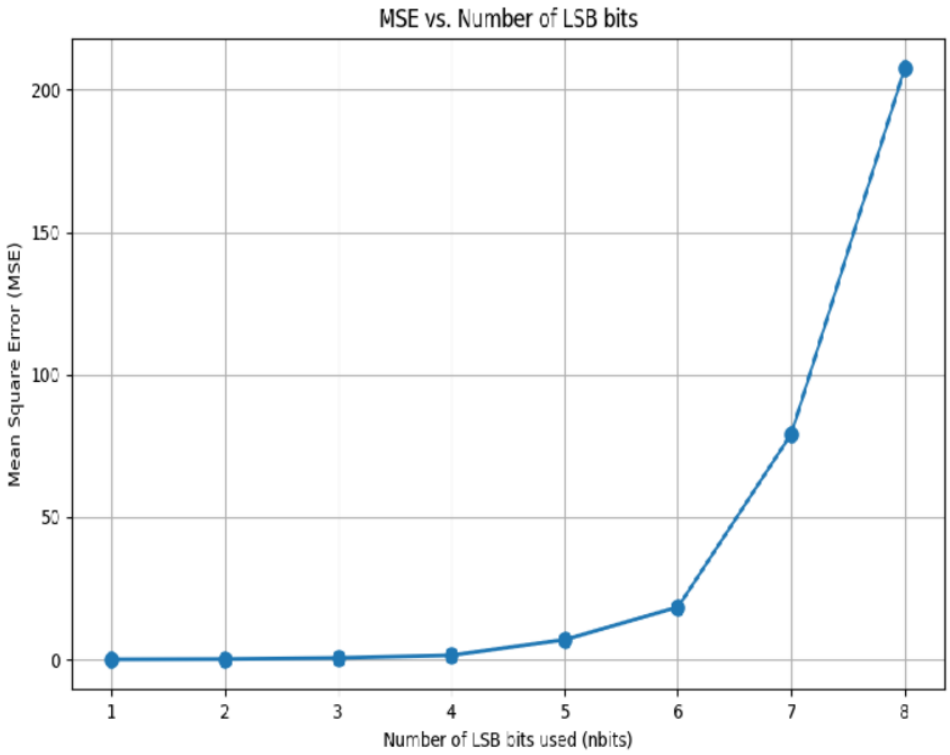
2. Korzystając z kodu znajdującego się w punkcie `Ukryjmy tekst w obrazku` wygenerować i wyświetlić szerego obrazków z ukrytą wiadomością:
   a. Wiadomość dla nbits=1 ma zajmować znakomitą większość obrazka (powyżej 75%).
   b. Każdy kolejny obrazek należy generować z większą wartością nbits (do 8) - będzie 8 obrazków z wiadomością.
   c. Dla każdego obrazka należy policzyć MSE z oryginałem.
   d. Wartości MSE należy umieścić na wykresie, w którym oś x stanowić będzie nbits.
   e. Źródło tekstu jest dowolne, ale proponuję użyć czegoś w stylu (https://pypi.org/project/lorem/).
   f. Do rysowania wykresu można wykorzystać funkcję `matplotlib.pyplot.plot`

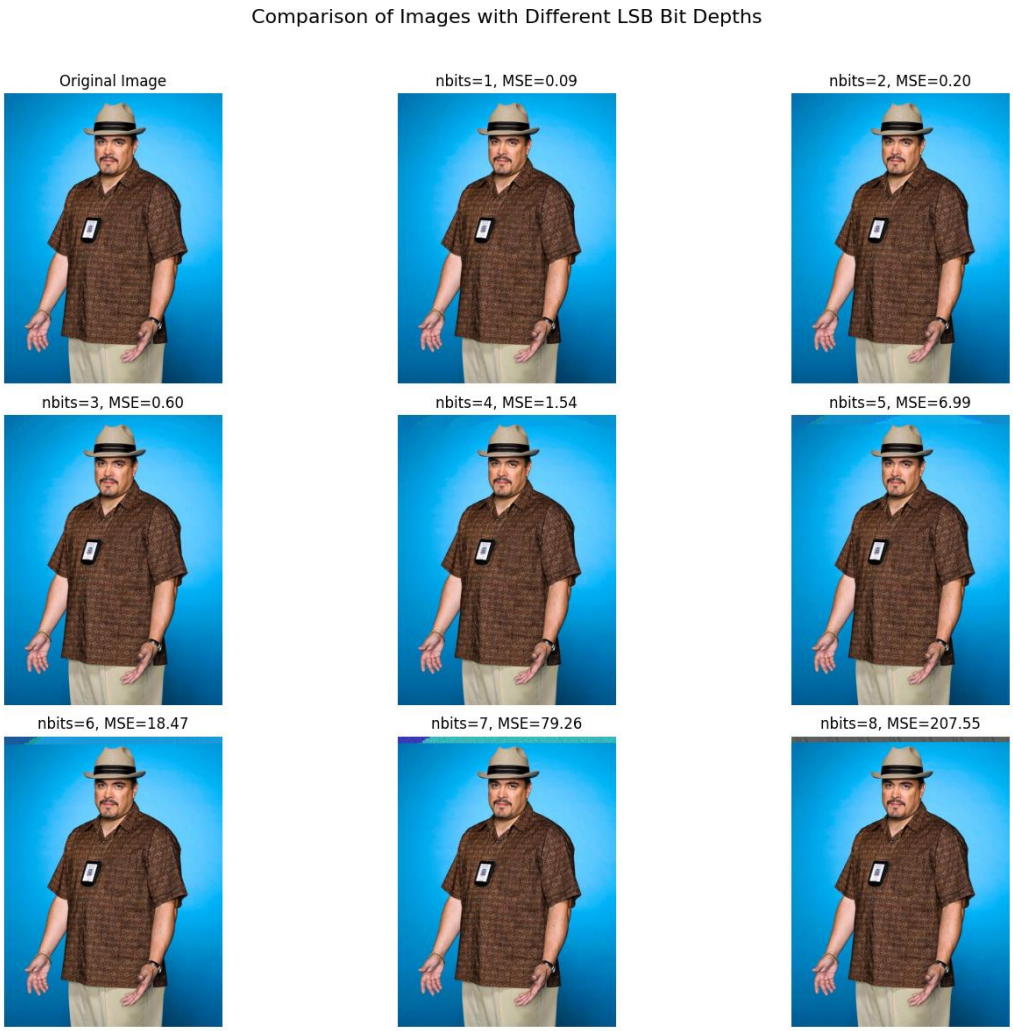
Program prosi użytkownika o ścieżkę do obrazu, generuje długą wiadomość Lorem Ipsum, ukrywa ją w obrazie za pomocą różnej liczby bitów LSB (od 1 do 8), zapisuje wszystkie obrazy z ukrytą wiadomością, oblicza MSE dla każdego z nich w porównaniu z oryginałem, tworzy wykres MSE i zestawienie wszystkich obrazów, a następnie wyświetla wyniki.

Wyniki działania programu:

```
Please enter the path to the image file: input.jpg
Original image dimensions: (981, 736, 3)
Generated message length: 47339 characters
Binary message length: 378712 bits
Using 17.48% of the image to hide the message with 1 bits
Image saved to output_nbits_1.png
nbits=1, MSE=0.09
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
Using 8.74% of the image to hide the message with 2 bits
Image saved to output_nbits_2.png
nbits=2, MSE=0.20
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
Using 5.83% of the image to hide the message with 3 bits
Image saved to output_nbits_3.png
nbits=3, MSE=0.60
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
Using 4.37% of the image to hide the message with 4 bits
Image saved to output_nbits_4.png
nbits=4, MSE=1.54
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
```

```
--------------------------------------------------
Using 3.50% of the image to hide the message with 5 bits
Image saved to output_nbits_5.png
nbits=5, MSE=6.99
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
Using 2.91% of the image to hide the message with 6 bits
Image saved to output_nbits_6.png
nbits=6, MSE=18.47
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
Using 2.50% of the image to hide the message with 7 bits
Image saved to output_nbits_7.png
nbits=7, MSE=79.26
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
Using 2.19% of the image to hide the message with 8 bits
Image saved to output_nbits_8.png
nbits=8, MSE=207.55
Original start: Lorem ipsum dolor sit amet, consectetur adipiscing
Extracted start: Lorem ipsum dolor sit amet, consectetur adipiscing
--------------------------------------------------
MSE plot saved to 'mse_plot.png'
All images comparison saved to 'all_images_comparison.png'
Displaying MSE Plot. Press any key to continue...
```


MSE vs. Number of LSB bits
Mean Square Error (MSE)
0
50
100
150
200
1
2
3
4
5
6
7
8
Number of LSB bits used (nbits)



Procent wykorzystania obrazu zależał od liczby bitów LSB:

- nbits=1: 17.40% obrazu
- nbits=2: 8.74% obrazu
- nbits=3: 5.83% obrazu
- nbits=4: 4.37% obrazu
- nbits=5: 3.50% obrazu
- nbits=6: 2.91% obrazu
- nbits=7: 2.50% obrazu
- nbits=8: 2.19% obrazu

Widać wyraźny trend: im więcej bitów LSB wykorzystujemy do ukrycia wiadomości, tym mniejszy procent obrazu jest potrzebny, ponieważ każdy piksel może przechować więcej informacji.

Wartości Mean Square Error (MSE) wskazują na stopień zniekształcenia obrazu:

- nbits=1: MSE=0.09 (praktycznie niewidoczne)
- nbits=2: MSE=0.20 (nadal niewidoczne)

- nbits=3: MSE=0.60 (minimalne zniekształcenia)
- nbits=4: MSE=1.54 (lekkie zniekształcenia)
- nbits=5: MSE=6.99 (widoczne zniekształcenia)
- nbits=6: MSE=18.47 (znaczne zniekształcenia)
- nbits=7: MSE=79.26 (silne zniekształcenia)
- nbits=8: MSE=207.55 (drastyczne zniekształcenia)

Wykres MSE pokazuje wykładniczy wzrost błędu przy zwiększaniu liczby modyfikowanych bitów. Między nbits=6 a nbits=8 następuje dramatyczny wzrost MSE, co wskazuje na punkt krytyczny, gdzie zniekształcenia stają się bardzo widoczne.

Porównując obrazy z różnymi wartościami nbits:

- nbits=1-4: Obrazy są praktycznie nieodróżnialne od oryginału gołym okiem. Zniekształcenia są minimalne, a MSE pozostaje poniżej 2.0. Są to optymalne wartości dla ukrytej stenografii.
- nbits=5-6: Zaczynają pojawiać się subtelne zniekształcenia, szczególnie w obszarach jednolitego koloru (jak niebieskie tło). MSE wzrasta do poziomów 6.99-18.47, ale obraz wciąż wygląda względnie normalnie.
- nbits=7-8: Widoczne są wyraźne zniekształcenia. Przy nbits=8 obraz ma wręcz "ziarnisty" wygląd z zakłóceniami przypominającymi szum. MSE osiąga bardzo wysokie wartości (79.26-207.55).

Wnioski:

1. Kompromis pojemność-niewykrywalność: Istnieje wyraźny kompromis między ilością danych, które można ukryć, a niewykrywalnością stenografii. Dla tego obrazu, użycie 1-4 bitów LSB zapewnia dobrą równowagę.
2. Wykładniczy wzrost MSE: Wzrost MSE nie jest liniowy - każdy dodatkowy bit LSB powoduje coraz większy wzrost błędu, co widać na wykresie.
3. Optymalny punkt operacyjny: Dla tego obrazu, nbits=4 wydaje się być punktem granicznym - zapewnia względnie dużą pojemność (wykorzystanie tylko 4.37% obrazu) przy wciąż akceptowalnym MSE=1.54.
4. Skuteczność ekstrakcji: We wszystkich przypadkach wiadomość była poprawnie odzyskiwana, co potwierdza niezawodność metody LSB, nawet przy wyższych wartościach nbits.

Eksperymenty pokazują, że metoda LSB jest efektywna i elastyczna, ale wymaga starannego doboru parametrów, by zachować balans między pojemnością a niewykrywalnością.

Opis kodu

Opisy głównych funkcji:

1. encode_as_binary_array/decode_from_binary_array: Konwertują tekst na ciąg bitów i z powrotem, umożliwiając ukrywanie tekstowych wiadomości.
2. load_image/save_image: Funkcje pomocnicze do wczytywania i zapisywania obrazów z odpowiednią konwersją przestrzeni kolorów między RGB i BGR (format używany przez OpenCV).

3. hide_message: Ukrywa binarną wiadomość w obrazie, modyfikując określoną liczbę najmniej znaczących bitów (nbits) każdego piksela. Funkcja wypełnia piksele obrazu bitami wiadomości, zaczynając od początku obrazu.
4. reveal_message: Odzyskuje ukrytą wiadomość poprzez ekstrahowanie określonej liczby najmniej znaczących bitów z pikseli obrazu.
5. calculate_mse: Oblicza błąd średniokwadratowy między obrazem oryginalnym a zmodyfikowanym, co pozwala ocenić stopień zniekształcenia obrazu.
6. generate_lorem_ipsum: Generuje tekst Lorem Ipsum o określonej liczbie akapitów, służący jako długa wiadomość testowa.
7. main: Główna funkcja programu, która:
    - Wczytuje obraz
    - Generuje wiadomość Lorem Ipsum
    - Konwertuje wiadomość na format binarny
    - Ukrywa wiadomość w obrazie przy różnych wartościach nbits (1-8)
    - Oblicza MSE dla każdego obrazu
    - Tworzy i zapisuje wykres MSE
    - Tworzy i zapisuje zestawienie wszystkich obrazów
    - Wyświetla wyniki

Program spełnia wszystkie wymagania zadania 2, wykorzystując metodę LSB do ukrycia wiadomości w obrazie i analizując wpływ liczby używanych bitów na widoczność zmian w obrazie poprzez pomiar MSE.

Zadanie 3

3. Przerobić funkcje `reveal_message` i `hide_message` z punktu `Zmiana wartości pikseli` w taki sposób, aby pozwalały zapisywać i odczytywać wiadomość zaczynając od zadanej pozycji (funkcje można napisać od nowa, ale wyjście ma pozostać niezmienione). Sygnatury funkcji powinny wyglądać następująco (spos to pozycja początkowa):
    a. `def hide_message(image, message, nbits=1, spos=0)`
    b. `def reveal_message(image, nbits=1, length=0, spos=0):`

W zadaniu 3 mamy przerobić funkcje hide_message i reveal_message tak, aby można było określić pozycję początkową (parametr spos), od której zacznie się ukrywanie/odczytywanie wiadomości w obrazie.

Oryginalna funkcja:

```python
def hide_message(image, message, nbits=1):
    """Hide a message in an image (LSB).
    nbits: number of least significant bits
    """
    # ... (kod)
    for i, chunk in enumerate(chunks):
        byte = "{:08b}".format(image[i])
        new_byte = byte[:-nbits] + chunk
        image[i] = int(new_byte, 2)
```

Zmodyfikowana funkcja:

```python
def hide_message(image, message, nbits=1, spos=0):
    """Hide a message in an image (LSB) starting from a specific position.
    nbits: number of least significant bits
    spos: starting position in the flattened image array
    """
    # ... (kod)
    # Make sure starting position is within the image bounds
    spos = clamp(spos, 0, len(image) - 1)

    # Check if message will fit in the image from the starting position
    if len(message) > (len(image) - spos) * nbits:
        raise ValueError("Message is too long for the specified starting position :(")

    for i, chunk in enumerate(chunks):
        # Add spos to position to start from the specified position
        pos = spos + i

        # Make sure we're still within the image bounds
        if pos >= len(image):
            break
```

W oryginalnej wersji funkcji hide_message, wiadomość była zawsze ukrywana od początku obrazu. Proces ukrywania wyglądał następująco:

1. Spłaszcz obraz do jednowymiarowej tablicy pikseli
2. Podziel wiadomość binarną na fragmenty o długości nbits
3. Dla każdego fragmentu:
    - Weź piksel o indeksie i (zaczynając od 0)
    - Zamień jego najmniej znaczące bity na bity wiadomości
    - Przejdź do kolejnego piksela

Taki podejście jest proste, ale ma istotne ograniczenie - wiadomość zawsze zaczyna się od pierwszego piksela obrazu.

W nowej wersji funkcji dodano parametr spos (starting position), który określa, od którego piksela rozpocząć ukrywanie wiadomości. Proces został zmodyfikowany:

1. Spłaszcz obraz do jednowymiarowej tablicy pikseli
2. Sprawdź, czy spos mieści się w granicach obrazu, a jeśli nie - ogranicz go
3. Sprawdź, czy wiadomość zmieści się w obrazie zaczynając od pozycji spos
4. Podziel wiadomość binarną na fragmenty o długości nbits
5. Dla każdego fragmentu:
    - Weź piksel o indeksie spos + i (zaczynając od spos)
    - Zamień jego najmniej znaczące bity na bity wiadomości
    - Przejdź do kolejnego piksela

Parametr spos wprowadza kilka istotnych korzyści:

1. **Elastyczność umieszczania wiadomości**
    - Możemy ukryć wiadomość w dowolnym miejscu obrazu, nie tylko na początku
    - Pozwala to na ukrywanie wielu różnych wiadomości w różnych częściach tego samego obrazu
2. **Zwiększone bezpieczeństwo**
    - Standardowe narzędzia stegoanalizy często zakładają, że dane są ukryte od początku pliku
    - Umieszczenie wiadomości w nieoczywistym miejscu utrudnia jej wykrycie
    - Pozycja początkowa może być używana jako swego rodzaju "klucz" - bez znajomości spos trudno odzyskać wiadomość
3. **Omijanie obszarów krytycznych obrazu**
    - Niektóre części obrazu (np. jednorodne obszary) są bardziej podatne na zmiany
    - Parametr spos pozwala ukryć wiadomość w mniej widocznych lub bardziej chaotycznych obszarach
4. **Optymalizacja stenografii**
    - Możemy omijać nagłówki plików i metadane, które są bardziej podatne na analizę
    - Umożliwia łatwiejsze ominięcie obszarów obrazu, które mogą zostać uszkodzone przy kompresji

Następnie zmieniliśmy funkcje „reveal_message”

Oryginalna wersja:

```python
def reveal_message(image, nbits=1, length=0):
    """Reveal the hidden message.
    nbits: number of least significant bits
    length: length of the message in bits.
    """
    # ... (kod)
    i = 0
    while i < length_in_pixels:
        byte = "{:08b}".format(image[i])
        message += byte[-nbits:]
        i += 1
```

Zmodyfikowana wersja z parametrem spos:

```python
def reveal_message(image, nbits=1, length=0, spos=0):
    """Reveal the hidden message starting from a specific position.
    nbits: number of least significant bits
    length: length of the message in bits
    spos: starting position in the flattened image array
    """
    # ... (kod)
    # Make sure starting position is within the image bounds
    spos = clamp(spos, 0, len(image) - 1)

    # Calculate how many pixels we need to check
    if length <= 0:
        length_in_pixels = len(image) - spos
    else:
        length_in_pixels = math.ceil(length/nbits)

    # Make sure we don't go beyond the image bounds
    if spos + length_in_pixels > len(image):
        length_in_pixels = len(image) - spos

    i = 0
    while i < length_in_pixels:
        # Add spos to position to start from the specified position
        pos = spos + i

        # Make sure we're still within the image bounds
        if pos >= len(image):
            break

        byte = "{:08b}".format(image[pos])
        message += byte[-nbits:]
        i += 1
```

W oryginalnej wersji funkcji reveal_message, wiadomość była zawsze odczytywana od początku obrazu. Proces odczytywania wyglądał następująco:

1. Spłaszcz obraz do jednowymiarowej tablicy pikseli
2. Określ, ile pikseli należy przeanalizować (na podstawie parametru length)
3. Dla każdego piksela (zaczynając od indeksu 0):
    - Odczytaj nbits najmniej znaczących bitów
    - Dodaj te bity do odczytanej wiadomości

- Przejdź do kolejnego piksela, aż do osiągnięcia wymaganej długości

To podejście działało pod warunkiem, że wiadomość zaczynała się od pierwszego piksela obrazu.

W nowej wersji funkcji dodano parametr spos (starting position), który określa, od którego piksela rozpocząć odczytywanie wiadomości. Proces został zmodyfikowany:

1. Spłaszcz obraz do jednowymiarowej tablicy pikseli
2. Sprawdź, czy spos mieści się w granicach obrazu, a jeśli nie - ogranicz go
3. Określ, ile pikseli należy przeanalizować, biorąc pod uwagę:
    - Parametr length (jeśli został podany)
    - Dostępną przestrzeń od pozycji spos do końca obrazu
4. Dla każdego piksela (zaczynając od indeksu spos):
    - Odczytaj nbits najmniej znaczących bitów
    - Dodaj te bity do odczytanej wiadomości
    - Przejdź do kolejnego piksela, aż do osiągnięcia wymaganej długości

Kluczowe różnice i udoskonalenia

1. Początek odczytu
    - Oryginał: zawsze od pierwszego piksela (indeks 0)
    - Nowa wersja: od piksela o indeksie spos
2. Obsługa braku podanej długości (length=0)
    - Oryginał: próbuje odczytać całą długość obrazu
    - Nowa wersja: odczytuje od pozycji spos do końca obrazu
3. Bezpieczeństwo przed wyjściem poza granice
    - Nowa wersja dodaje zabezpieczenia sprawdzające:
        - Czy spos jest w granicach obrazu
        - Czy suma spos + length_in_pixels nie przekracza długości obrazu
4. Precyzyjniejsza kontrola zakresu odczytu
    - Nowa wersja uwzględnia zarówno pozycję początkową, jak i dostępną przestrzeń

Zadanie 4

4. Napisać program, który odzyska obrazek z obrazka zapisanego w punkcie 'Ukryjmy obrazek w obrazku'. Funkcja dekodująca (odnajdująca wiadomość) powinna przyjmować następujące parametry:
    a. image - obrazek z ukrytym obrazkiem
    b. length - długość ukrytego obrazka
    c. nbits - liczba najmłodszych bitów użyta do zakodowania obrazka

Główne funkcje programu

1. load_image(image_path)

Ta funkcja wczytuje obraz z pliku i konwertuje go z przestrzeni kolorów BGR (używanej przez OpenCV) do RGB (używanej przez matplotlib). Jest to konieczne, ponieważ OpenCV i matplotlib używają różnych porządków kanałów kolorów.

2. reveal_message(image, length, nbits=1)

Funkcja ta wyodrębnia ukrytą wiadomość binarną z obrazu:

1. Tworzy maskę bitową na podstawie liczby bitów LSB (nbits) używanych do ukrycia danych: mask = (1 << nbits) - 1

Ta formuła generuje maskę z odpowiednią liczbą jedynek binarnych. Na przykład dla nbits=2 maska będzie 0b11 (3 dziesiętnie).

2. Spłaszcza obraz do jednowymiarowej tablicy, co upraszcza iterację przez wszystkie wartości pikseli.
3. Iteruje przez wartości pikseli, wyodrębniając nbits najmniej znaczących bitów z każdego piksela:

   extracted_bits = pixel_value & mask

Operacja bitowa AND (&) z maską zachowuje tylko nbits najmniej znaczące bity, zerując pozostałe.

4. Konwertuje wyodrębnione bity na ciąg znaków binarnych i dodaje je do wiadomości.
5. Kontynuuje process aż osiągnie wymaganą długość (length) i zwraca dokładnie określoną liczbę bitów.

3. extract_hidden_image (image, length, nbits=1)

Ta funkcja używa reveal_message do wyodrębnienia danych binarnych, a następnie przekształca je w plik obrazu:

1. Wyodrębnia ciąg binarny używając funkcji reveal_message.
2. Konwertuje ciąg binarny na format heksadecymalny, przetwarzając po 8 bitów naraz:

byte = binary_message[i:i + 8]

hex_data += format(int(byte, 2), '02x')

3. Konwertuje dane heksadecymalne na obiekt bytes:

   bytes_data = bytes.fromhex(hex_data)

4. Zapisuje dane jako plik obrazu.
5. Wczytuje zapisany obraz i zwraca go do wyświetlenia.

4. main()

Funkcja główna programu, która:

- Pobiera od użytkownika informacje o pliku z ukrytym obrazem, długości ukrytych danych i liczbie użytych bitów LSB
- Wczytuje obraz-nośnik
- Wyodrębnia ukryty obraz

- Wyświetla oba obrazy obok siebie

Kluczowe aspekty implementacji

1. **Użycie maski bitowej**: Zastosowanie operacji (1 << nbits) - 1 to efektywny sposób tworzenia maski, która wyodrębnia określoną liczbę najmniej znaczących bitów. Jest to szybsze niż podejście konwersji do łańcucha znaków i z powrotem.
2. **Spłaszczenie obrazu**: Przekształcenie wielowymiarowej tablicy pikseli w jednowymiarową tablicę upraszcza przetwarzanie i pozwala na sekwencyjne odczytywanie danych.
3. **Obsługa różnych wartości nbits**: Program jest elastyczny i może obsługiwać różne scenariusze ukrywania danych - od użycia tylko pojedynczego bitu LSB (najmniej widoczne zmiany) do wielu bitów (większa pojemność).
4. **Konwersja binarna → hex → bytes**: Ta sekwencja konwersji jest kluczowa, ponieważ:
    - Najpierw odczytujemy dane jako bity binarne
    - Następnie grupujemy je w bajty i konwertujemy na format heksadecymalny
    - Na końcu przekształcamy w rzeczywiste bajty, które mogą być zapisane jako plik
5. **Obsługa błędów**: Program zawiera zabezpieczenia przed typowymi błędami, takimi jak nieprawidłowa ścieżka pliku czy niepoprawne wartości wejściowe.

Metoda dekodowania

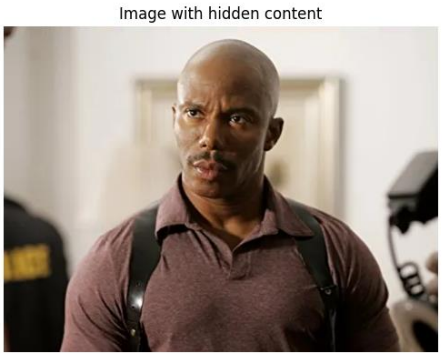
Program wykorzystuje klasyczną metodę steganografii LSB, która polega na:

1. Uznaniu, że najmniej znaczące bity obrazu-nośnika zostały zastąpione bitami ukrytego obrazu
2. Sekwencyjnym odczytaniu tych bitów od początku obrazu
3. Rekonstrukcji oryginalnych bajtów danych z odczytanych bitów
4. Zapisaniu odzyskanych danych jako plik obrazu

Ta metoda jest odwrotnym procesem do metody ukrywania obrazu, gdzie najmniej znaczące bity oryginalnego obrazu zostały zastąpione bitami ukrywanego obrazu.

Taka implementacja zapewnia prostotę, wydajność i elastyczność – trzy kluczowe cechy dobrego programu steganograficznego.

Wyniki działania programu:



Przykładowe parametry dla programu:

```
=== Hidden Image Extraction Program ===
Enter image file with hidden content: stego_image.png
Enter length of hidden image (in bits): 41712
Enter number of least significant bits used for encoding: 3
Loading image from stego_image.png...
Extracting hidden image...
Extracted image saved to extracted_image.jpg
```

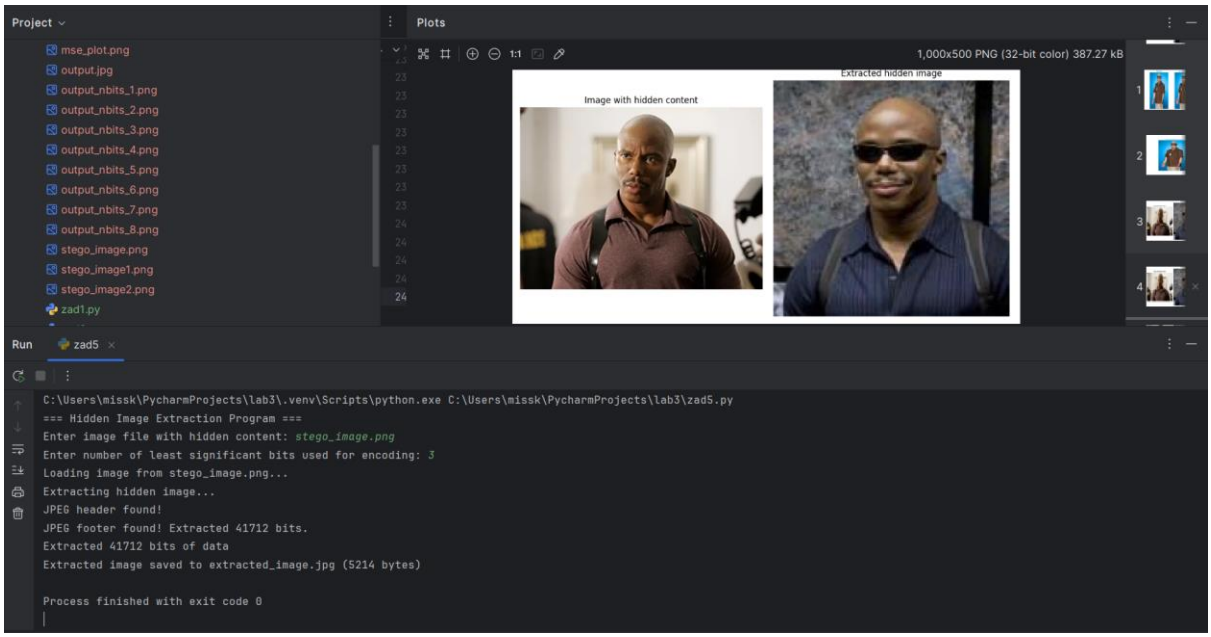
Dodatkowo był napisany program, który może ukryć w sobie inny obraz. Jest to plik 1.py, lub opcja w menu [6]. Wygląda ona tak:

```
Enter the path to the cover image: image1.png
Enter the path to the secret image to hide: image2.png
Output will be saved as: stego_image2.png
Use default settings for image size and encoding bits? (y/n): n
Enter maximum width for secret image (e.g., 150): 150
Enter maximum height for secret image (e.g., 150): 150
Enter number of LSB bits to use (1-4 recommended): 3
Cover image loaded: (375, 500, 3)
Secret image loaded: (400, 400, 3)
Available space in cover image: 1687500 bits
Resized secret image to: (150, 150, 3)
Compressed secret image size: 5214 bytes
Binary data length: 41712 bits
Image saved to stego_image2.png
Successfully hidden 41712 bits of image data
To extract the hidden image, remember this data length: 41712
Or use the automatic extraction method that looks for JPEG markers
Display the original and steganography images? (y/n): n
```

Zadanie 5

5. Przekazywanie długości ukrytej wiadomości do funkcji odzyskującej nie jest ładnym rozwiązaniem. W przypadku tajnego tekstu można by było zapisywać na jego końcu znak `\0`, który oznaczałby koniec ciągu znaków. W przypadku ukrytych obrazków można by było wykorzystać standardowy nagłówek i stopkę pliku (i tego dotyczy to zadanie).
   Należy zmodyfikować program napisany w poprzednim punkcie tak, aby funkcja dekodująca nie potrzebowała parametru `length`
   Uwagi:
   - w zadaniu wymagane jest jedynie szukanie stopki (plik może być na początku obrazka)
   - istnieje mała szansa, że stopka będzie występowała naturalnie w danych - należy wziąć pozycję pierwszego wystąpnienia (w tym i większości przypadków to zadziała, ale normalnie powinno się dokładniej przeanalizować nagłówek).

Wyniki działania:



Kluczową różnicą między zadaniem 4 i 5 jest sposób określania końca ukrytego obrazu:

- W zadaniu 4: Trzeba podać dokładną długość (length) ukrytych danych
- W zadaniu 5: Program automatycznie znajduje koniec obrazu przez wykrycie stopki pliku JPEG

Zmiany w kodzie:

1. Nowa funkcja reveal_message_until_footer

Zamiast funkcji reveal_message, która wymagała parametru length, dodano funkcję reveal_message_until_footer, która:

- Definiuje znaki nagłówka JPEG (FFD8FF) i stopki (FFD9) w formie heksadecymalnej i binarnej
- Ekstrahuje dane bit po bicie, stale szukając nagłówka i stopki JPEG
- Po znalezieniu nagłówka, zaczyna budować właściwą wiadomość
- Po znalezieniu stopki, kończy ekstrakcję i zwraca tylko dane między nagłówkiem a stopką

2. Zmiany w funkcji extract_hidden_image

Funkcja została zmodyfikowana, aby:

- Nie wymagać parametru length
- Używać nowej funkcji reveal_message_until_footer zamiast reveal_message
- Lepiej obsługiwać przypadki, gdy ekstrakcja nie powiodła się całkowicie

3. Zmiany w funkcji main

- Usunięto prośbę o podanie długości ukrytego obrazu
- Dodano dodatkowe sprawdzanie, czy wyodrębniony obraz można poprawnie wczytać

Jak działa nowe podejście

1. Identyfikacja plików JPEG - Program zna standardowe sygnatury plików JPEG:
    - Nagłówek: FFD8FF (Start of Image)
    - Stopka: FFD9 (End of Image)
2. Strategia wyszukiwania:
    - Początkowo szuka nagłówka JPEG w ekstrahowanych danych
    - Po znalezieniu nagłówka, rozpoczyna poszukiwanie stopki
    - Po znalezieniu obu, zwraca tylko dane zawarte między nimi
3. Zabezpieczenia:
    - Jeśli nie znajdzie nagłówka i stopki w typowej sekwencji, próbuje przeszukać całą ekstrakcję
    - Nawet jeśli znajdzie tylko nagłówek (bez stopki), zwraca dane od nagłówka do końca
    - Jeśli nie znajdzie ani nagłówka, ani stopki, zwraca wszystkie dane