Relatório - Atividade 2

Universidade Estadual de Campinas
Instituto de Computação
Introdução ao Processamento Digital de Imagens

Abril de 2018

Felippe Trigueiro Angelo
210479

1 Descrição geral da atividade 2

A atividade 2 da disciplina Introdução ao Processamento Digital de Imagens teve como objetivo a implementação da técnica de Esteganografia, que consiste em ocultar uma mensagem dentro de uma imagem, mensagem esta que pode ser um texto ou uma outra imagem.

Na atividade em questão o conteúdo de um arquivo de texto foi ocultado em uma dada imagem, onde ambos são selecionados pelo usuário. Cada caractere presente no texto foi convertido para seu respectivo código binário e cada um dos bits resultantes da conversão é ocultado nos bits de cada pixel da imagem. Para esta atividade foram utilizadas imagens coloridas, assim cada pixel possui a capacidade de guardar três bits, onde a posição dos bits no byte correspondente, foi passada pelo usuário. Por fim, a imagem codificada foi passada para um programa decodificador para a recuperação da mensagem.

2 Executando o Programa

2.1 Etapa de Codificação

Para executar o programa de codificação, entre no terminal do linux, navegue até a pasta onde os códigos estão localizados e digite o seguinte comando:

```
sudo python3 codificar.py [imagem-de-entrada] [texto-entrada] [plano-bits] [imagem-de-saída]
Ex: sudo python3 codificar.py monalisa.png text.txt 0 monalisa_enconde.png
```

2.2 Etapa de Decodificação

Para executar o programa de decodificação, entre no terminal do linux, navegue até a pasta onde os códigos estão localizados e digite o seguinte comando:

```
sudo python3 decodificar.py [imagem-de-saída] [plano-bits] [texto-saída]
Ex: sudo python3 decodificar.py monalisa_encode.png 0 out.txt
```

2.3 Dados de Entrada

Para a correta execução do programa é necessário que alguns cuidados sejam tomados com os dados de entrada:

- As imagens de entrada, tanto do codificador quanto do decodificador, devem estar na mesma pasta em que está contido o código-fonte;
- As imagens de entrada devem ser coloridas;
- O arquivo de texto contendo a mensagem deve estar na mesma pasta que está contido o código-fonte;
- Foi convencionado que o plano de bits deve ser um valor entre 0 e 2, selecionando assim os bits menos significativos. Caso um valor fora dessa faixa seja inserido, uma mensagem de erro será emitida;
- Por razões óbvias, o plano de bits passado para o codificador, deve ser o mesmo do decodificador;
- O formato da imagem de saída deve ser .png.

2.4 Dados de Saída

Após a etapa de codificação descrita no primeiro ítem deste texto, o programa gera como saída uma imagem codificada contendo a mensagem passada como entrada. Adicionalmente, é mostrado na tela um conjunto de quatro imagens contendo a mensagem passada como entrada, onde a codificação em cada uma delas é feita em plano de bits diferentes. São eles: 0, 1, 2, e 7.

Algumas informações adicionais não são exibidas na forma de imagens, mas são exibidas no terminal. São elas:

- Número de caracteres da mensagem;
- Número máximo de carateres que podem ser codificados na imagem.

Após a conclusão da etapa de decodificação, é gerado um arquivo de texto, cujo nome foi passado como entrada, contendo o texto ocultado na imagem.

3 Descrevendo o Código

3.1 Algoritmos e Estruturas de Dados utilizadas

3.1.1 Inserção de um caractere indicando fim de texto

Como a mensagem a ser codificada tem a liberdade de conter qualquer caracter existente, se fez necessária a inserção de um caractere especial para indicar o fim da mensagem. Foi escolhido para tal finalidade, o caractere \0, cuja representação binária é dada pelo byte 0. Assim, a operação em questão foi feita utilizando o seguinte código Python:

```
with open(text_message, 'r') as file_object:
        stri = file_object.read()
stri += '\0'
```

A vantagem de se utilizar *with as* é que ao fim do código presente em seu interior, o arquivo apontado por file_object é fechado.

3.2 Conversão dos caracteres da mensagem para bits

Para a realização da esteganografia, a mensagem de entrada foi convertida de caracteres para bits por meio do código Python abaixo:

```
stri_bin = ''.join(format(ord(x), '08b') for x in stri)
```

A função *format()* converte a representação de uma string para algum formato especificado em seus parâmetros. Nesse caso cada caracter da mensagem é passado como argumento e convertido para um formato de oito bits, onde os bits mais significativos são completados com o bit 0, caso necessário. Por fim, a função *join()* junta todos os bits convertidos em uma única string, convertendo assim a mensagem de entrada para um conjunto de bits.

3.3 Inserção da mensagem na imagem de entrada

A inserção dos caracteres da mensagem na imagem de entrada foi feita por meio da criação da função change_image_bits presente no arquivo esteg.py, cujo código é mostrado abaixo:

```
import numpy as np

def change_image_bit(image, bit_plane, row, column, stri_bin):
    i = 0
    flag_r = 0
    flag_c = 0
    imag = np.copy(image)

    for r in range(row):
        if flag_r == 0:
            for c in range(column):
```

```
12              if flag_c == 0:
13                  for bp in range(3):
14                      bin_pixel_layer = list(format(imag[r, c, bp], '08b'))
15                      bin_pixel_layer[7 - bit_plane] = stri_bin[i]
16                      bin_pixel_layer = int(''.join(bin_pixel_layer), 2)
17                      imag[r, c, bp] = bin_pixel_layer
18                      i = i + 1
19                      if(i == len(stri_bin)):
20                          flag_c = 1
21                          flag_r = 1
22                          break
23                      else:
24                          break
25              else:
26                  break
27
28      return imag
```

Os três laços *for* encadeados, percorrem os pixels da imagem de entrada a partir do pixel superior esquerdo, ou pixel (0, 0). Assim, cada camada R, G, B é percorrida e o bit correspondente de cada camada é mudado de acordo com o valor passado por bit_plane. É importante citar que o valor 0 de bit_plane corresponde ao valor do bit menos significativo, e o valor 7 corresponde ao valor do bit mais significativo, daí a necessidade de se utilizar a expressão (7 - bit_plane), presente na linha 15, que faz a operação de se selecionar o bit correto.

Para se converter a string binária para um valor inteiro foi utilizada a função *int()* presente na linha 16, onde a base numérica do número presente no primeiro argumento é passado como segundo argumento.

Quando todos os bits dos caracteres da mensagem de entrada forem codificados na imagem, os flags *flag_c* e *flag_r* são acionados e os laços *for* são encerrados.

3.4 Extração da mensagem oculta na imagem

Como parte da atividade 1, foi pedido para rotular cada região da imagem de entrada, assim foi criada uma imagem contendo o rótulo de cada objeto contido na imagem. Para tal foi utilizo o trecho de código abaixo.

```
1  for r in range(row):
2      for c in range(column):
3          for bp in range(3):
4              bin_pixel_layer = format(img[r, c, bp], '08b')
5              seq_bit_plane.append(bin_pixel_layer[7 - bit_plane])
6
7  for i in range(len(seq_bit_plane)):
8      list_bit_plane.append(seq_bit_plane[(8*i):(8*i+8)])
9      list_bit_plane[i] = ''.join(list_bit_plane[i])
10
11 end_position = list_bit_plane.index('00000000')
12 list_bit_plane = list_bit_plane[0:end_position]
13
14 for i in range(len(list_bit_plane)):
15     decod_text.append(chr(int(list_bit_plane[i], 2)))
16 decod_text = ''.join(decod_text)
```

Para a extração dos bits da mensagem contidos na imagem de saída do codificador e conversão nos caracteres correspondentes, o programa decodificador foi dividido em quatro partes.

A primeira parte consiste em criar uma lista de tipo string contendo todos os bits da imagem que estão na posição fornecida como plano de bit. Em seguida esses bits são agrupados em blocos de 8 bits, formando assim os bytes que compões cada um dos caracteres da mensagem. Em terceiro lugar, são retirados da lista os bytes que não correspondem efetivamente a mensagem, estes bytes correspondem àqueles que estão após o caractere de fim de mensagem \0. Por fim, os bytes são convertidos para caracteres, formando assim a mensagem decodificada.

3.5 Estruturas de dados utilizadas

No código, foram utilizadas as estruturas de dados array, tupla e lista no armazenamento das informações desejadas. No caso das imagens foram utilizados os arrays providos pela biblioteca *numpy*, gerando arrays da classe *ndarray*. Como estamos utilizando imagens coloridas, elas são representadas como arrays do tipo MxNx3, onde cada matriz MxN representa um canal do RGB.

A tupla é utilizada no armazenamento das dimensões das imagens, por meio da extração do atributo *.shape*.

As listas são utilizadas principalmente para se alterar o valor da string de bits, assim a variável string, que não permite alteração, é convertida para uma lista, podendo assim alterar-se o seu valor. As listas também foram utilizadas para o armazenamento da conjunto de bytes que correspondem aos caracteres da mensagem. A sua utilização facilita a manipulação pois permite a alteração do seu conteúdo, permitindo assim a eliminação dos bytes que não correspondem aos caracteres da mensagem.

4 Testes adotados e e Limitações do programa

Foram feitos testes com as imagens contidas no endereço da página da disciplina. Tais imagens podem ser vistas na figura 1. Portanto, foram testadas imagens coloridas, quadradas e retangulares no formato .png.

Convém citar novamente aqui, que foi adotada a convenção de que as imagens de entrada devam ser coloridas, entretanto do ponto de vista teórico, nada impediria a realização de uma esteganografia em imagens monocromáticas, entretanto, esse procedimento implica em limitações como redução em três vezes no número de caracteres que podem ser ocultados na imagem.

(a) Baboon

(b) Monalisa

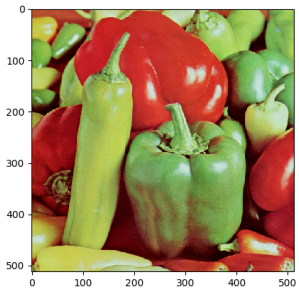
(c) Peppers

(d) Watch

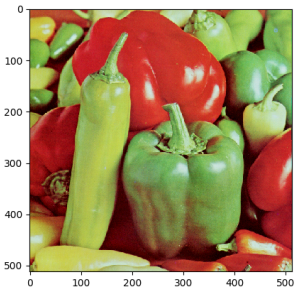
Figure 1: Imagens utilizadas como teste.

Também foram feitos testes onde o valor do plano de bits foi alterado para se analisar o comportamento da imagem codificada. No teste em questão foi utilizado um texto contendo

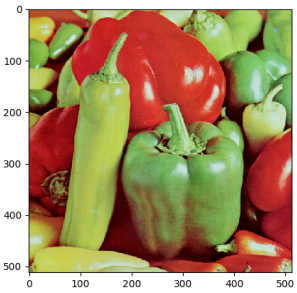
394264 caracteres e valores de planos de bit iguais a 0, 1, 2 e 7. Os resultados do teste feito com a imagem peppers.png, podem ser vistos na figura 2.



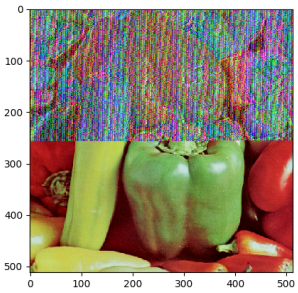
(a) Plano de Bit 0



(b) Plano de Bit 1



(c) Plano de Bit 2



(d) Plano de Bit 7

Figure 2: Imagens utilizadas como teste.

A partir dos resultados dos testes é possível afirmar que à medida que se aumenta o valor do plano de bits, maior é a degradação da imagem e maior é a facilidade para se descobrir que existe uma mensagem contida na imagem. Entretanto, é possível afirmar que os valores de plano de bits 0, 1 e 2, pouco influênciam no resultado final. Isso ocorre pois nesses planos os valores dos pixels em cada camada, são alterados por no máximo o valor 4, já o plano 7, altera o valor do pixel em 128, permitindo assim uma indicação da existência da informação.

5 Referências Bibliográficas

1. **The Python Standard Library.** Disponível em: https://docs.python.org/3/library/index.html
2. **The Matplotlib API Pyplot.** Disponível em: https://matplotlib.org/api/pyplot_summary.html