MC920 - INTRODUÇÃO AO PROCESSAMENTO DIGITAL DE IMAGEM
2S2019
RELATÓRIO TÉCNICO - TRABALHO 4

Andressa Gabrielly Macedo Marçal
RA 262878

1. Especificação do Problema

O objetivo deste trabalho é implementar um Algoritmo de Esteganografia em imagens digitais.

1.1. O que é Esteganografia

A técnica de esteganografia consiste em ocultar uma mensagem dentro de uma imagem. A mensagem pode ser um texto ou mesmo uma outra imagem.

A esteganografia possui várias aplicações interessantes, tais como a divulgação de mensagens sem o conhecimento de um possível interceptador, a inclusão de marcas para verificação de direitos autorais, entre outras.

Uma técnica comum é a modificação de um ou mais bits que compõem cada pixel da imagem, de modo que a mensagem a ser oculta seja armazenada nesses bits modificados. Uma das técnicas mais conhecidas e utilizadas é o da alteração do bit menos significativo de cada pixel, ou seja, aquele se encontra mais à direita da palavra binária, é conveniente para ser modificado, pois produz alterações na imagem que não são normalmente perceptíveis à visão humana.

2. Descrição da Técnica Escolhida

Neste trabalho, a esteganografia deverá alterar os bits da mensagem a ser oculta nos bits menos significativos de cada um dos três canais de cores da imagem colorida(RGB).

O diagrama ilustrado logo a seguir, irá apresentar as principais etapas da esteganografia, em que a mensagem "MC" é oculta em uma imagem colorida por meio da alteração dos bits menos significativos dos pixels da imagem pertencentes a cada canal de cor.

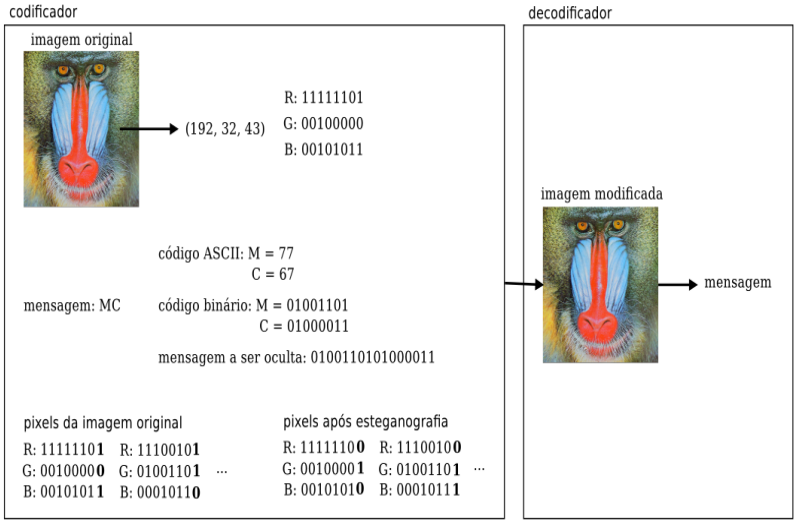

Figure 1. Ilustração da esteganografia em imagens coloridas.

2.1. Estrutura de uma Imagem Digital

Para entender essa técnica, alguns lembretes de alguns princípios básicos da imagem digital podem ser úteis para melhor entendimento.

- Uma imagem digital é composta por **X linhas** e por **Y colunas**;
- O pixel representa o menor elemento endereçável de uma imagem;
- Cada pixel é associado a uma cor, geralmente decomposta em três cores principais/primárias: Vermelho, Verde e Azul. É o que chamamos de modelo RGB;
- As intensidades de vermelho, verde e azul podem variar de 0 a 255;

EX: **Pixel BRANCO = (255,255,255) e Pixel PRETO = (0,0,0)**;

- Um pixel ocupa 3 bytes de memória, 1 para cada componente principal (daí o valor máximo de 255);
- Um byte consiste em 8 bits, representando um número binário (exemplo: 1010 0101);
- O valor mais alto que um byte pode receber é 1111 1111, que é igual a 255 em decimal;

2.2. Least Significant Bits(LSB)

Como o próprio nome sugere, a técnica de bit menos significativo(LSB) é baseada na ocultação de informações no bit menos significativo de cada byte da imagem.

Vamos pegar a seguinte representação de 1 Byte, onde o peso é anotado abaixo de cada bit:

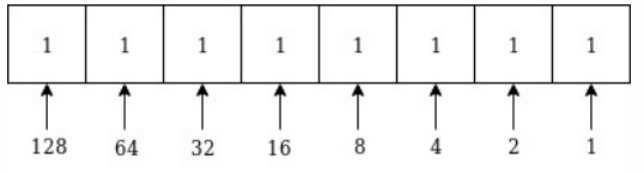

Figure 2. Significância de Bits

- O primeiro bit à esquerda é o de **maior peso**, já que é o que tem maior influência no valor do byte. Seu peso é 128.
- Na parte extrema da direita, seu peso é 1 e tem um impacto muito pequeno no valor do byte. De certa forma, esse bit é o **bit menos significativo(LSB)** desse byte.

O diagrama abaixo ilustra um exemplo da diferença de cores, quando o bit menos significativo do canal vermelho é modificado.

Figure 3. Comparativo da alteração de LSB no canal de cor vermelha

É notável que a modificação é praticamente imperceptível aos olhos humanos! Então, dessa forma, podemos modificar 3 bits por pixel sem que seja perceptível.

2.3. Plano de Bits

Imagens digitais em escala de cinza podem ser pensadas como uma matriz de pixels com intensidades(valores). Em uma imagem de 8 bits, esses valores variam de 0 a 255. Como por exemplo os planos de bits de uma imagem em escala de cinza de 8 bits podem ser vistos como a pilha de 8 imagens binárias, contendo apenas pixels brancos ou pretos. Cada valor de pixels é determinado com base na representação binária de sua intensidade. O primeiro plano de bits contém um conjunto de pixels cujos bits menos significativos estão ativados e o plano de 8 bits contém o conjunto de pixels cujos bits mais significativos estão ativados.

Existem representações variadas disso, mas eu vou utilizar o que peguei no livro Digital Image Processing por Gonzalez e Woods.

Na imagem abaixo é possível visualizar algumas imagens em escala de cinza, e o impacto visual de cada alteração em cada plano de bits e seus respectivos resultados.

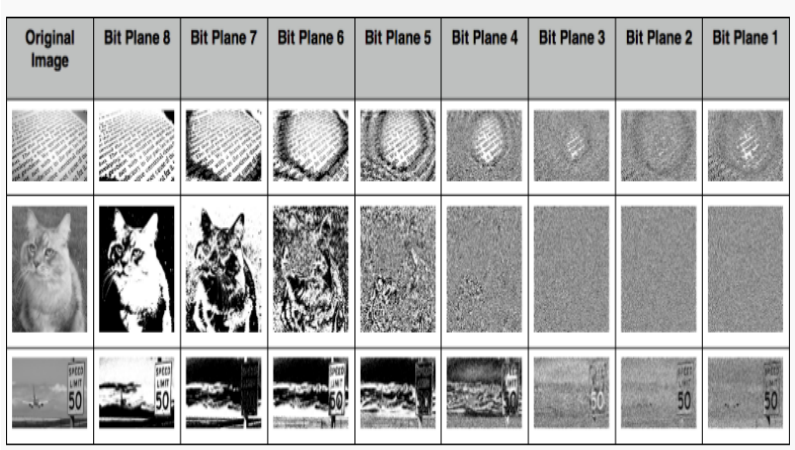

Figure 4. Resultado de mudança realizada por Plano de Bits

2.4. Como ocultar uma mensagem?

A mensagem é uma sequência de bits. Nesse caso a única limitação é que o tamanho da mensagem em bits deve ser inferior ao número de pixels na imagem multiplicado por 3(no caso se for RGB).

3. Codificando

O método que codifica o texto de string em código binário na imagem receberá como parâmetro uma lista de

entradas com a matriz de pixels da imagem a ser modificada, a representação binária do texto a ser codificado e o plano de bits em que o texto será codificado.

Seu retorno será a imagem codificado onde conterá a matriz de pixels da imagem com o texto codificado.

4. Decodificação

É um processo reverso direto. A mensagem secreta é detectada e extraída, pegando os bits menos significativos da matriz criptografada.

No script, o método para decodificar o texto dentro da imagem em uma sequência de texto, terá como parâmetros uma lista de entradas com a matriz de pixels da imagem a ser modificada e o plano de bits em que o texto está codificado.

A função irá retornar o texto de decodificação da imagem.

5. Execução do Script

1. Executar o script **trab4.py** diretamente no terminal do sistema Linux;

   - Comando: **python3 trab4.py**

2. As imagens que serão usadas para testes e as que serão salvas com a mensagem codificada ficaram armazenadas no diretório imagens;

3. Irá exibir no terminal um menu de opções, onde o usuário deverá digitar encode ou Encode para codificar, e decode ou Decode para a opção de decodificar;

   - **Execução do Menu do Script:**

     (a) **Encode ou encode:** Para encriptar o texto na imagem;

     (b) **Decode ou decode:** Para decriptar o texto que foi inserido na imagem;

     (c) **Texto de entrada(txtInput):** digitar o nome do arquivo de texto + extensão, que se encontra dentro do diretório de arquivos de texto chamado **inputTexts/'**

     (d) **Plano de Bits(bitPlane):** Inserir o numero do plano de bit que deseja ser modificado de 0 a 7;

     (e) **Imagem de Saída:(imageOutput)** Nome + extensão da imagem de saída;

```
(mestrado) andressa@andressa-Inspiron-7472:~/Documentos/mestrado/trabalho4$ python trab4.py

:: Esteganografia - Trabalho 4 ::

** Encode **

** Decode **

Digite o nome da opcao desejada: encode
Digite o nome da imagem (com sua extensão): monalisa.png
Insira o nome do arquivo de texto a ser inserido:(ex: nome.txt) test3.txt
Insira o numero do plano de bits que você deseja modificar(numero de 0 a 7): 0
Digite o nome da nova imagem(com sua extensão): mona_0.png

Lendo a imagem ...


Eliminando espaço e caracteres especiais do texto a ser codificado ...


.: Codificando o texto na imagem :.


Armazenando imagem modificada ...

Imagem salva com sucesso!
```

Figure 5. Encode na Prática

```
(mestrado) andressa@andressa-Inspiron-7472:~/Documentos/mestrado/trabalho4$ python trab4.py

:: Esteganografia - Trabalho 4 ::

** Encode **

** Decode **

Digite o nome da opcao desejada: decode
Digite o nome da imagem (com sua extensão): mona_0.png
Insira o plano de bits que você deseja modificar( de 1 a 8): 0
Insira o nome do arquivo em que deseja salvar o texto: mona_0.txt

Lendo a imagem ...


.: Decodificando :.


Escrevendo ...


 Concluído!
```

Figure 6. Decode na Prática

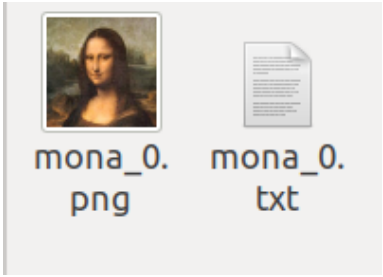

Figure 7. Resultado final com mensagem decriptada em arquivo de texto de saída

5.1. Bibliotecas Utilizadas

- OpenCV2: import cv2
- Numpy: import numpy as np
- RE: import re
- OS: import os

5.2. Funções Utilizadas

- **def readImage(filename):**

  Método para ler novas imagens

  - **@param1 filename:** O local do arquivo da imagem;
  - **@return img:** Uma lista de int's com a matriz de pixels da imagem;

- **def storeImage(filename, image_matrix):**

  Método para armazenar novas imagens

  - **@param1 filename:** o local do arquivo da imagem;
  - **@param2 image_matrix:** uma lista de entradas com a matriz de pixels da imagem;

- **def writePlans(img):**

  Método para mostrar planos específicos da imagem

  - **@param1 img:** Uma lista de entradas com a matriz de pixels da imagem;

- **def txtToBin(filename):**

  Método para converter um texto de sequência em código binário.

  - **@param1:** nome do arquivo: str
    O local/path do arquivo do texto;
  - **@return:** binarray: str
    A representação em cadeia do texto em código binário;

- **def changeNewLine(originalFile, destFile):**

  - **@param1 originalFile:** o local do arquivo do texto de entrada;
  - **@param2 destFile:** o local do arquivo do texto de saída;

- **def writeText(filename, data):**

  Método para escrever texto em um arquivo.

  - **@param1 filename:** str
    O local do arquivo do texto de entrada;
  - **@param2 data:** str
    O texto a ser gravado no arquivo;

- **def encodeImage(imageMatrix, binaryText, bitPlane):**

  Método para codificar um texto de sequencia de caracteres no código binário na imagem.

  - **@param1 imageMatrix:** list
    Uma lista de entradas com a matriz de pixels da imagem a ser modificada;
  - **@param2 binaryText:** str
    A representação binária do texto a ser codificado;
  - **@param3 bitPlane:** int
    O plano de bits em que o texto será codificado;
  - **@return encodedImage:** list
    Uma lista de entradas com a matriz de pixels da imagem com o texto codificado;

- **def decodeImage(imageMatrix, bitPlane):**

  Método para decodificar o texto dentro da imagem em uma sequência de texto.

  - **@param1 imageMatrix:** list
    Uma lista de entradas com a matriz de pixels da imagem a ser modificada;
  - **@param2 bitPlane:** int
    O plano de bits em que o texto está codificado;
  - **@return texto:** str
    O texto de decodificação da imagem;

- **def main():** Função principal, onde chama o menu de opções do script;

5.3. Entrada de Dados

As imagens de entrada estão no formato PNG (*Portable Network Graphics*). Alguns exemplos encontram-se disponíveis no diretório:

http://www.ic.unicamp.br/ helio/imagens_coloridas/

5.4. Leitura das Amostras

1. Das amostras de imagens utilizadas na implementação do projeto:

   - **monalisa.png**
   - **baboon.png**
   - **peppers.png**
   - **watch.png**

2. Das amostras de arquivos de texto, que foram geradas do site http://mussumipsum.com/, temos:

   - **test1.txt**
     - Tamanho do arquivo: 6,5kb
     - Quantidade de Caracteres: 6495
   - **test2.txt**

  - Tamanho do arquivo: 18,6kb
  - Quantidade de Caracteres: 18625
- **test3.txt**
  - Tamanho do arquivo: 359,2kb
  - Quantidade de Caracteres: 355175
  - Palavras: 58206
  - Linhas: 2999

No anexo do projeto, seguem todas as imagens com alterações feitas, inserindo texto e alterando seus planos de bits do 0 ao 7.

6. Resultados

Figure 8. Imagem Baboon

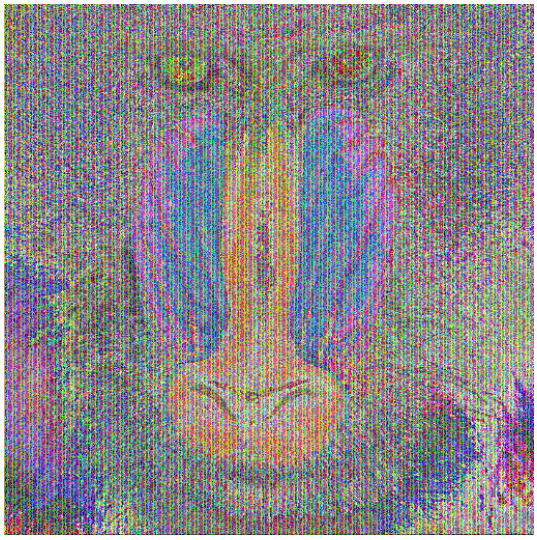

Figure 9. Plano de bit 7 + test3.txt

Figure 10. Plano de bit 6 + test3.txt

- O resultado de maior impacto e maior visibilidade é quando se insere algum contexto no bit 0.
- No bit 1 e 2 ainda é de fácil percepção que há contido algo, causando algum ruído na imagem.
- Nos bits menos significativos, no caso o 7, foram onde o resultado não foi percebido, não consegui notar alteração alguma no 0 e no 1, confesso que a partir do 4 já é mais perceptível o ruído na imagem, teste feito com um texto de 355175 caracteres.

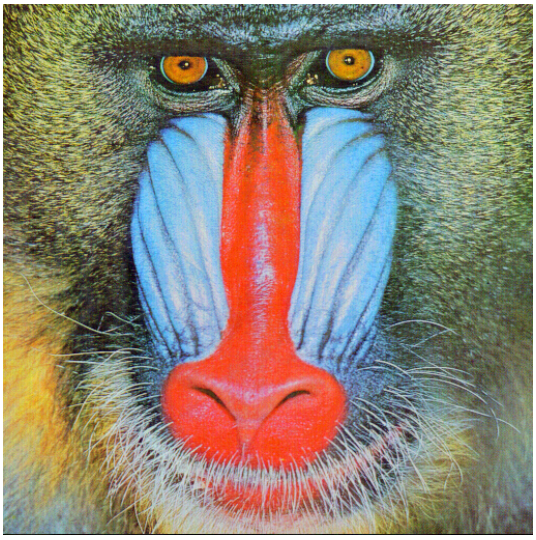

Figure 11. Plano de bit 3 + test3.txt

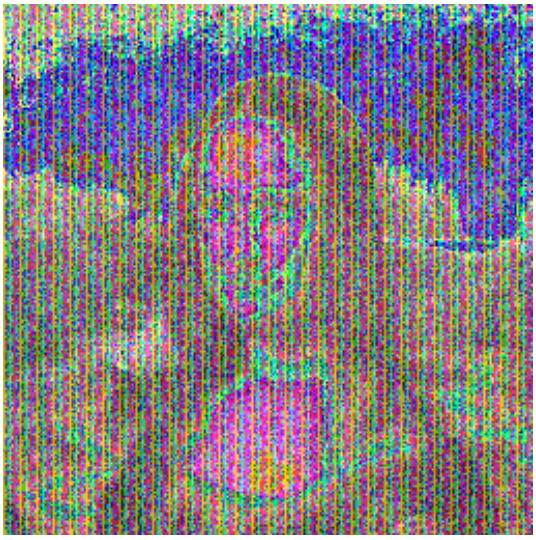

Figure 13. Plano de bit 7 + test3.txt

Figure 12. Imagem Monalisa

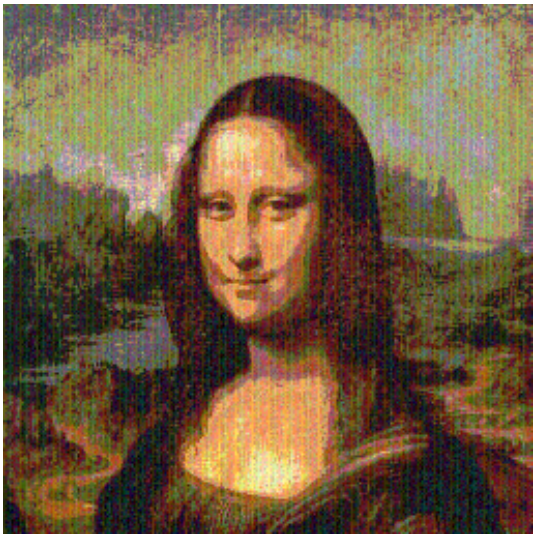

Figure 14. Plano de bit 5 + test3.txt

7. Dificuldades Encontradas

- Alguns testes deram problemas ao rodar, quando eu ia inserir um texto muito grande, demorava para processar e quando ia abrir o arquivo de texto, o aplicativo parava de executar. Tentei 9999 paragrafos e não deu certo.

8. Conclusão

Todas as imagens que foram geradas nesse Trabalho4 foram anexas no diretório **/imagens**.

O arquivo completo, com todos os arquivos, imagens processadas e relatórios irá compactado na extensão .zip, com nome **trabalho4.zip**.

- **Output:**
  - **/output:** Todas as imagens de teste processadas, ucada uma terá a aleração de cada imagem com base no plano de bits de 0 a 7(computando os 8 bits);
  - **/textos:** Onde está os textos usados para teste.
  - **imagens:** Todas as imagens coloridas usadas nos testes
  - Relatório Técnico em PDF;
  - script trab4.py

Figure 15. Plano de bit 0 + test3.txt

9. Referências

[1] https://www.boiteaklou.fr/Steganography-Least-Significant-Bit.html

[2] https://www.imageeprocessing.com/2017/11/hide-message-or-image-inside-image.html

[3] http://domnit.org/blog/2007/02/stepic-explanation.html

[4] https://www.gta.ufrj.br/grad/09_1/versao-final/stegano/tecnicas.html

[5] https://spin.atomicobject.com/2013/10/08/image-compression-bit-planes/

[6] https://en.wikipedia.org/wiki/Bit_plane

[7] https://en.wikipedia.org/wiki/Bit_plane

[8] https://theailearner.com/2019/01/25/bit-plane-slicing/

[9] https://spin.atomicobject.com/2013/10/08/image-compression-bit-planes/