Trabalho 2

Ronnypetson Souza da Silva

Introdução ao Processamento Digital de Imagem

Professor: Hélio Pedrini

IC - Instituto de Computação

UNICAMP - Universidade Estadual de Campinas

r211848@dac.unicamp.br / ronnypetson.silva@students.ic.unicamp.br / rsronnypetson4@gmail.com

1) Introdução

Neste relatório, são mostradas as atividades realizadas no Trabalho 2 da disciplina Introdução ao Processamento Digital de Imagem. O primeiro objetivo é, dados uma imagem e um arquivo de texto com uma mensagem na entrada do programa, inserir a mensagem na imagem através de um dos planos de bit (bit 0 ao bit 7) dos bytes do arquivo da imagem. A ideia para esconder a mensagem é inserir a codificação ASCII dos caracteres da mensagem em um plano de bits pouco significativos, para não causar diferenças na imagem perceptíveis ao olho humano. A representação em sequência de bits da mensagem é a concatenação da representação em bits dos caracteres. O segundo e último objetivo do trabalho é, dados um plano de bits e uma imagem, extrair uma suposta mensagem escondida no plano de bits informado. Além disso, também deve-se mostrar, para cada canal da imagem, o que existe nos planos de bit 0, 1, 2 e 7, a fim de identificar a existência de uma possível mensagem escondida em pelo menos um desses planos.

2) Os scripts codificar.py e decodificar.py

A implementação do trabalho é composta por dois scripts, um para codificação da mensagem e o outro para decodificação e exibição dos planos de bit 0, 1, 2 e 7. O script de codificação é o arquivo “codificar.py” e sua execução é chamada pelo seguinte protótipo de comando no terminal Linux:

```
$ python codificar.py <imagem_entrada> <texto_entrada> <plano_bits> <imagem_saida>
```

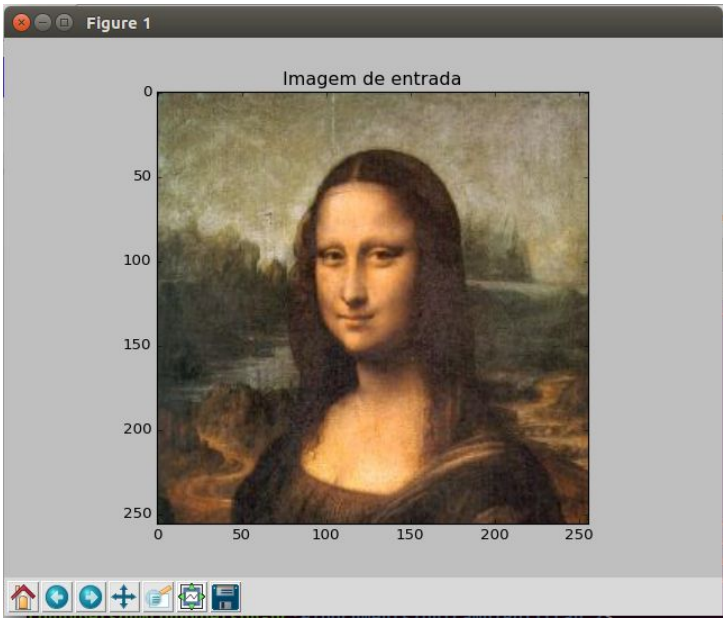
onde <imagem_entrada> é o nome de um arquivo .png contendo a imagem a ter uma mensagem inserida, <texto_entrada> é o nome de um arquivo de texto com o texto a ser inserido, <plano_bits> é um inteiro entre 0 e 7, inclusive, que é o plano de bits onde a mensagem será escondida, e <imagem_saida> é o nome de um arquivo .png que conterá a imagem da entrada com a mensagem embutida. Todos os arquivos indicados devem estar no mesmo diretório do script.

A seguir são descritas as etapas principais na execução desse script.

a) Definição das máscaras binárias para tornar um bit 1 ou 0. Dado um byte qualquer, para alterar o seu bit da posição *p* para o valor 1, a ideia é aplicar uma máscara ao byte com a operação *bitwise OR*. A máscara deve conter somente bits 0, exceto na posição *p*, que obviamente deve conter 1. Como o valor 0 é neutro em relação à operação lógica *OR*, todos os bits diferentes do bit da posição *p* permanecerão intactos, enquanto que o bit da posição *p* terá o valor 1. A representação decimal dessas máscaras é 1, 2, 4, 8, 16, 32, 64 e 128, na ordem do bit menos significativo para o mais significativo. Já para tornar o valor do bit 0, aplica-se uma máscara com a operação *bitwise AND*, onde a máscara consiste de

bits 1, exceto na posição que se deseja alterar. O raciocínio é análogo ao da máscara usada com *bitwise OR*, com a diferença de que o bit 1 que é neutro em relação à operação lógica *AND*. A representação decimal dessas máscaras é 254, 253, 251, 247, 239, 223, 191 e 127, do bit menos significativo ao mais significativo.

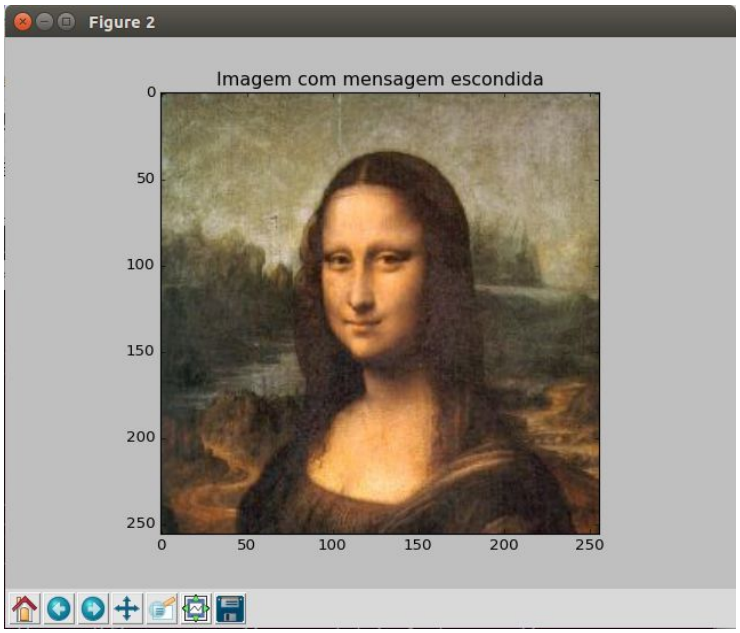
b) A imagem de entrada é carregada num array com duas ou três dimensões, dependendo da imagem ser monocromática ou ter vários canais, como é o caso das imagens RGB. Em seguida, a imagem é exibida para o usuário. A ilustração abaixo é a do arquivo "monalisa.png" baixado do diretório http://www.ic.unicamp.br/~helio/imagens_coloridas_png/, exibida pelo programa usando a biblioteca Matplotlib. Nota-se que, devido ao modo como a biblioteca Matplotlib interpreta as imagens RGB, que é invertido, é necessário realizar uma transformação - não *in-place* - para BGR antes.



c) A mensagem a ser inserida na imagem é carregada a partir do arquivo de texto indicado na entrada. A codificação dos caracteres da mensagem é ASCII.

d) Codificação da mensagem na imagem de entrada. Nesta etapa a mensagem carregada no passo anterior tem a sua representação em sequência de bits inserida no plano de bits indicado na entrada. A sequência de bits da mensagem é composta pela concatenação dos bits que representam seus caracteres, de acordo com o modelo ASCII. A sequência de bits de cada caractere fica invertida na sequência final, pois os bits menos significativos são extraídos primeiro, mas isso não atrapalha a codificação e decodificação da mensagem. A imagem é atravessada por cada canal de cada pixel para que um bit da sequência de bits da mensagem seja inserido no plano de bits definido pelo usuário. Desse modo, cada

pixel da imagem contém 3 bits de informação da mensagem. Daí o programa exibe a imagem resultante com a mensagem escondida, como ilustra a figura abaixo, e salva a imagem resultante no mesmo diretório do script com o nome de arquivo especificado pelo usuário na entrada.



O script de codificação é o arquivo “decodificar.py” e sua execução é chamada pelo seguinte protótipo de comando no terminal Linux:

```
$ python decodificar.py <imagem_a_ser_analisada> <plano_bits> <arquivo_texto_saida>
```

onde <imagem_a_ser_analisada> é o nome de um arquivo .png contendo a imagem que pode ter uma mensagem escondida, <plano_bits> é um inteiro entre 0 e 7, inclusive, que é o plano de bits de onde a mensagem será extraída e <arquivo_texto_saida> é o nome do arquivo de texto que conterá a mensagem extraída. Todos os arquivos indicados devem estar no mesmo diretório do script.

As etapas principais na execução do script “decodificar.py” estão descritas abaixo.

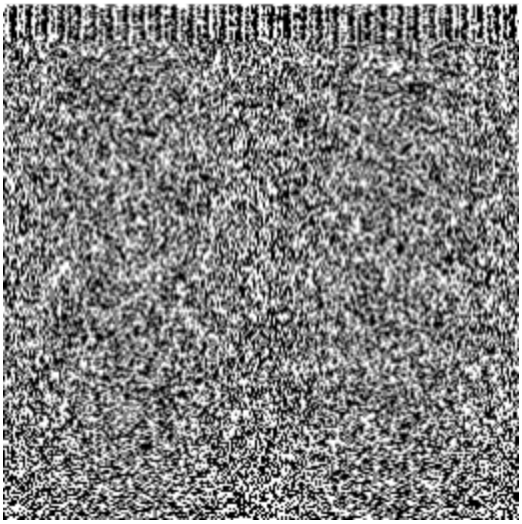
1) Aaa
    a) Definição das máscaras binárias para tornar um bit 1 ou 0. Idêntica à definição das máscaras no script “codificar.py”.
    b) Carrega e exibe a imagem da entrada de modo idêntico ao script de codificação, com a diferença de que nesse caso a imagem de entrada provavelmente passou por algum processo anterior de inserção de mensagem.

c) Decodificação da mensagem. Nesta etapa, o plano de bits indicado pelo usuário é percorrido em cada canal de cada pixel e a cada 8 bit lidos desse plano um novo caractere é formado. Então cada caractere extraído é concatenado em uma *string* que após a decodificação é escrita no arquivo de saída indicado. A ilustração abaixo mostra o conteúdo extraído da imagem de saída do script de codificação para o arquivo "monalisa.png". Nota-se a existência de uma parte não-legível, visto que a mensagem não ocupa toda a capacidade do plano de bits da imagem.

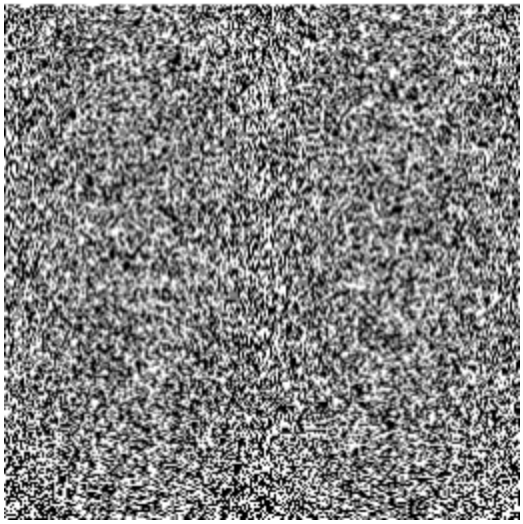
d) Exibição dos planos de bit 0, 1, 2 e 7 de cada canal. O conteúdo dos planos de bit 0, 1, 2 e 7 de cada canal é exibido como uma imagem binarizada, de modo que um bit 1 tem valor 255 e um bit 0 tem valor 0. As figuras abaixo ilustram esses planos de bit para a imagem "monalisa.png". Nota-se uma diferença de distribuição dos bits no começo das imagens do plano de bits 0, onde a mensagem foi escondida nesse exemplo. Também nota-se que os planos de bit menos significativos tem conteúdo mais parecido com ruído, devido ao fato que essses

planos de bitcarregam informações mais “finas” sobre os pixels. As imagens dos planos de bit são salvas, no mesmo diretório do script, em arquivos com nome <canal>_plano_<plano_bits>_<nome_img_entrada>.png, onde <canal> e <plano_bits> são o canal (“R”, “G”, “B” ou “Gray” para monocromático) e o número do plano de bits mostrado.

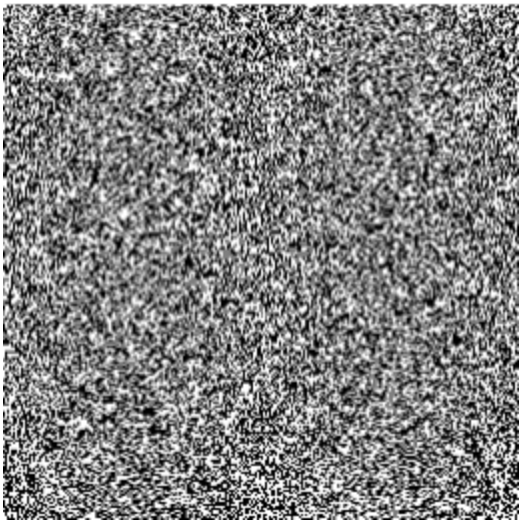
R plano 0



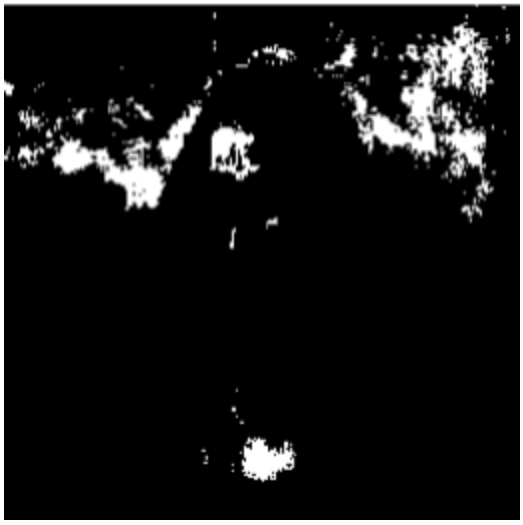
R plano 1



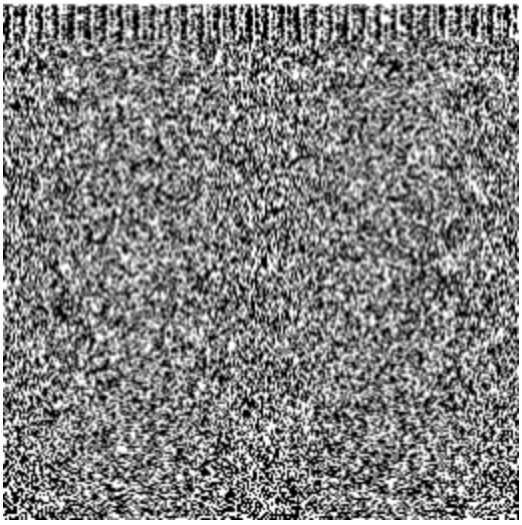
R plano 2



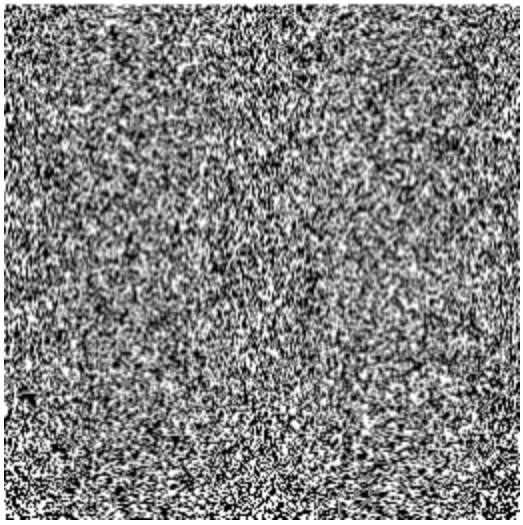
R plano 7



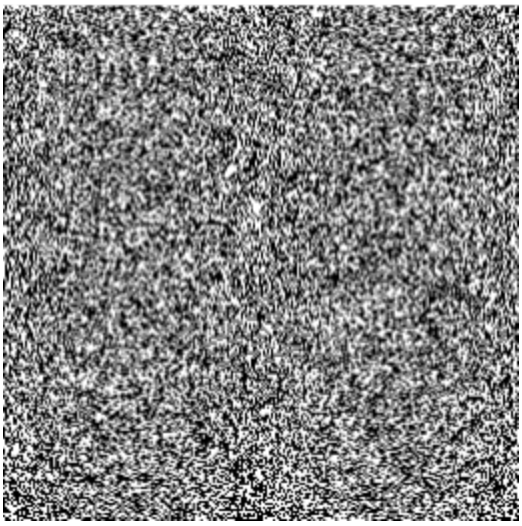
G plano 0



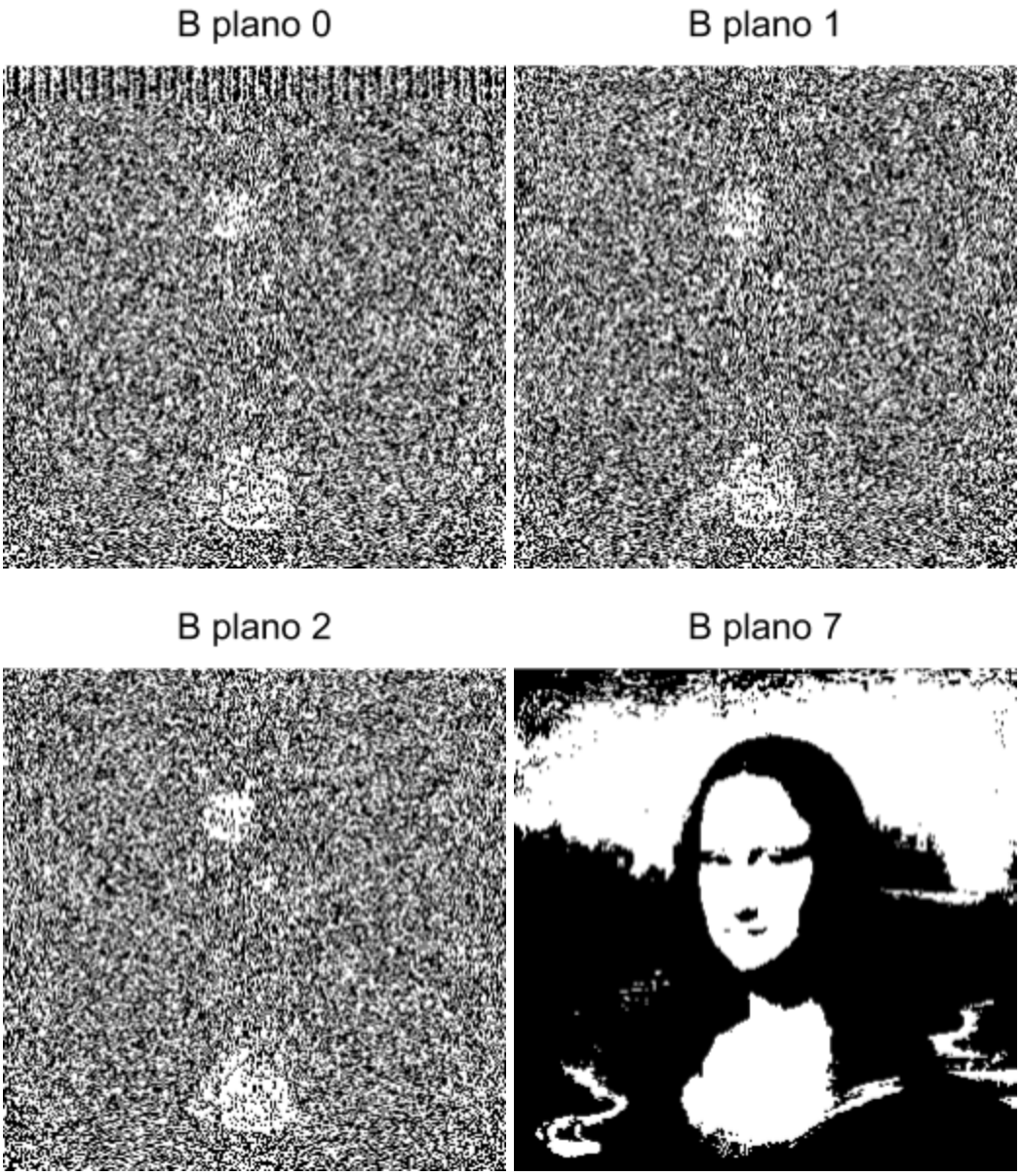
G plano 1



G plano 2



G plano 7



3) **As estruturas de dados** do trabalho são *arrays* de 1, 2 e 3 dimensões e estruturas de dados das bibliotecas usadas. Os arrays numpy possuem a propriedade *shape* das imagens, que são arrays de uma dimensão, enquanto que as imagens monocromáticas são arrays bidimensionais cujos valores vão de 0 à 255 (profundidade 8). As imagens de canais R, G e B são arrays de 3 dimensões, onde a última delas representa o canal de cores. As dimensões dos arrays bidimensionais são as dimensões da imagem, com o valor em uma linha e coluna da matriz representando o pixel da mesma linha e coluna da imagem.

4) **Considerações especiais**. Os scripts foram desenvolvidos para aceitar tanto imagens monocromáticas quanto imagens com mais de um canal. Os arquivos de texto com as mensagens extraídas devem ser abertos em um editor de texto, não pelo comando “cat”, pois as mensagens se encontram no início dos arquivos e o comando “cat” pode subir a mensagem na tela, não possibilitando a sua visualização.

5) **Testes**. Testes foram realizados com as imagens baixadas no endereço http://www.ic.unicamp.br/~helio/imagens_coloridas_png/ com planos de bit diferentes. Na imagem “watch.png” foram notados padrões de listras dentro de alguns planos de bit que não são perceptíveis na imagem original.

6) **Sobre as limitações**. Mensagens pequenas podem ser difíceis de detectar visualmente, pois seu rastro pode passar despercebido; No Fedora 27, pode ser necessário atualizar o gerenciador de arquivos para ver todas as imagens geradas no diretório. Outra limitação diz respeito ao diretório do arquivo de entrada, pois estes devem estar na mesma pasta do script sendo executado.