

Trabalho 4 de Introdução ao Processamento de Imagem Digital

Nome: Paulo Junio Reis Rodrigues - RA: 265674

30/12/2020

1 Introdução

O objetivo deste trabalho, será aplicar o método de esteganografia em imagens coloridas. Este método tem como objetivo ocultar mensagens ou arquivos dentro de imagens e vídeos. Esteganografia possui várias aplicações interessantes, tais como a inclusão de marcas para verificação de direitos autorais, a inclusão de mensagens ou arquivos sem a possibilidade do interceptador saber a informação "escondida".

Para aplicar esta técnica é preciso modificar os bits menos significativos da imagem de entrada, pois, foi percebido que bits de baixa ordem, quando alterados, não provocam uma modificação visual na figura, por esse motivo, esses bits são utilizados para ocultar arquivos nas imagens.

Portanto, o intuito do projeto é aplicar esta técnica em figuras coloridas no formato *RGB*, alterando somente os bits menos significativos de cada banda de cor de cada um dos pixels da imagem de entrada.

Junto com este relatório, está sendo enviado o arquivo trabalho_4_265674.zip que possui dentro dele todos os arquivos que irão ser citados neste relatório.

2 Programa

Todos os programas foram implementados usando Python 3.8.5, com as bibliotecas Numpy 1.19.2, OpenCV 4.4.0.42.

3 Como executar

Para este projeto foram criados dois scripts, um para codificar o arquivo dentro da imagem de entrada, e outro script para decodificar o arquivo dentro da figura gerada. O primeiro algoritmo tem como entrada os seguintes itens:

Imagen de entrada: A imagem colorida que será utilizada para codificação.

Arquivo de entrada: Arquivo que será codificado.

Plano de bit: Plano de bit que será guardado todos os bits do arquivo de entrada.

Diretório da imagem de saída: É o nome do diretório que será salvado a imagem de saída.

Com todos os dados enviados, podemos chamar o primeiro script desta maneira:

```
python trabalho_4_codificar.py baboon.png arquivo.txt 1 outputs/imagemSaida.png
```

Já o segundo algoritmo, tem como entrada os seguintes itens:

Imagen de saída: A imagem colorida que será utilizada para decodificação.

Plano de bit: Plano de bit que será pego todos os bits do arquivo codificado.

Diretório do arquivo de saída: É o nome do diretório que será salvado o arquivo de saída.

E pode ser chamado da seguinte maneira:

```
python trabalho_4_decodificar.py outputs/imagemSaida.png 1 outputs/arquivoSaida.txt
```

3.1 Entrada

Para a entrada do primeiro algoritmo, deve ser enviado uma imagem colorida no formato *RGB*, e logo depois o diretório do arquivo que será codificado dentro da imagem, podendo ser uma mensagem em formato texto, ou uma outra imagem. Em terceiro, deve ser enviado o plano de bit que será feito a esteganografia, e por último deve ser enviado o diretório que deverá ser salva a imagem de saída depois da aplicação do método.

Logo depois da codificação, temos as entradas do script de decodificação. A primeira entrada, é a imagem de saída do primeiro algoritmo, já segunda, é o plano de bit que o arquivo está ocultado, e por último temos o nome do diretório onde o arquivo de saída deverá ser armazenado.

3.2 Saída

O primeiro algoritmo tem como saída uma imagem colorida no formato *RGB* com o arquivo codificado dentro dela. Já o segundo script tem como saída o arquivo decodificado no formato que o usuário enviar como parâmetro durante a execução.

4 Parâmetros utilizados

Todas as imagens utilizadas para execução do programa estão presentes no diretório *imagens/*, e todas estão disponíveis em https://www.ic.unicamp.br/helio/imagens_coloridas/. Arquivos extras como textos e outras imagens, podem ser encontradas na pasta de arquivos/.

As saídas dos dois scripts executados terão o nome escolhido pelo usuário durante a execução de cada um deles. No caso, a primeira saída será a imagem com o arquivo codificado, e a segunda saída será o arquivo decodificado.

5 Solução do exercício

Como foi dito nas seções anteriores, para este projeto foram implementados dois scripts, o primeiro para realizar a codificação do arquivo na imagem de entrada, e logo depois foi criado outro algoritmo com o intuito de decodificar o arquivo para recuperá-lo.

5.1 Script de codificação

Durante a etapa de codificação, é feito primeiramente a leitura da imagem colorida de entrada utilizando a função *imread* do *OpenCV* a partir do diretório informado. Logo depois é carregado o arquivo que será codificado, porém ele é lido em formato binário e formatado para 8 bits utilizando a função *Format* do Python, para que durante a decodificação seja mais fácil para recuperar o arquivo codificado. Logo depois de todos estes processos, é retornado uma string que contém somente valores de 0 e 1.

Logo após o tratamento dos dados, é feita a esteganografia. Para isso, é colocado primeiramente um delimitador de parada no final da string retornada, para que durante a decodificação, seja feito corretamente a recuperação do arquivo, para que não haja perda ou ganho de informação. Após isso, para cada pixel da imagem é pego os valores de cada camada de cor, para que seja feita a codificação do arquivo em apenas um plano de bit escolhido no começo da execução do algoritmo. Tudo isso é feito utilizando os operadores *Slice* do *Python*.

Após todos estes processos a imagem com o arquivo codificado estará gerada, e ela será gravada em um diretório informado pelo usuário. Lembrando que, o algoritmo deixa o usuário escolher qualquer plano de bit da imagem, porém ele avisa se um plano de bit escolhido for mais significativo.

5.2 Script de decodificação

Nesta etapa, primeiramente é feita a leitura da imagem de saída do algoritmo de codificação utilizando a função *imread* do *OpenCV*, logo após deste processo, para cada pixel da imagem, será pego 1 bit para cada camada de cor do plano de bit escolhido no começo da execução do script. Ao final, é gerado uma grande string com somente valores de 0 e 1, porém essa variável ainda contém valores que são "inúteis" para a geração do arquivo codificado. Para corrigir isso, é feita

uma busca do delimitador colocado durante a codificação, para que somente sejam recolhidos os bits necessários para a geração do arquivo codificado.

Contudo, logo depois de todo este processo, todos os bits são novamente agrupados no formato de 8 bits, para que, seja feita a escrita (em formato binário) de um novo arquivo, que será gravado no diretório informado pelo usuário no começo da execução do script.

6 Resultados

6.1 Texto pequeno

O primeiro teste feito foi utilizado um texto pequeno de aproximadamente 80 bytes, e o plano de bit escolhido foi o 1, e toda a esteganografia foi feita na imagem do **peppers.png**. Como o arquivo utilizado foi pequeno, pode se notar que a imagem de saída gerada não possui quase nenhuma alteração em seu plano de bit 1, mostrando que para arquivos pequenos, esse método pode ser bastante imperceptível. A imagem gerada está representada na figura 1 e os planos de bits gerados podem ser observados na figura 2. No final de toda execução o texto foi recuperado corretamente pelo decodificador.



Figura 1: Aplicação do método de esteganografia na imagem **peppers.png** utilizando um arquivo de texto pequeno

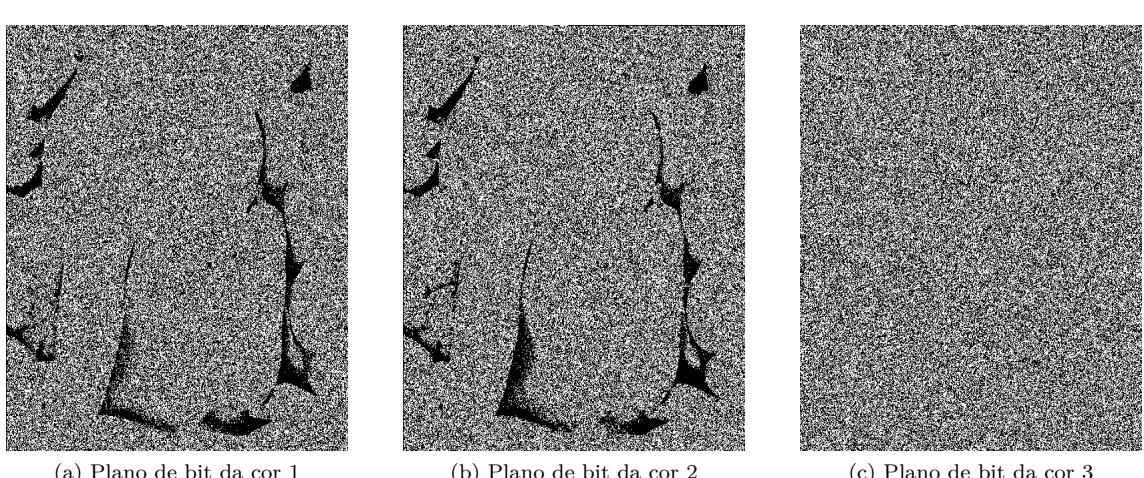


Figura 2: Planos de bits 1 da imagem gerada.

6.2 Texto grande

O segundo teste foi feito utilizando um texto maior, com aproximadamente 45.433 bytes, e a esteganografia foi feita no plano de bit 2. Depois da aplicação do método, a imagem gerada ainda continuava sem nenhuma modificação visual, porém, podemos notar uma grande diferença nos planos de bits da figura, aparentemente ela possui algum padrão diferente na parte superior do plano de bit, indicando que a imagem foi modificada por algum algoritmo. Contudo, caso essa imagem tivesse sido interceptada por alguém que conheça essa técnica, pode ser facilmente observado que ela está ocultando algo.

Portanto, para arquivos maiores a imagem gerada, mesmo com uma aparência visual idêntica a imagem de entrada, pode conter rastros que indicam que a figura foi alterada. A imagem gerada pode ser encontrada na figura 3 e todos os planos de bits 2 estão representados na figura 4. No final de toda execução o texto foi recuperado corretamente pelo decodificador.

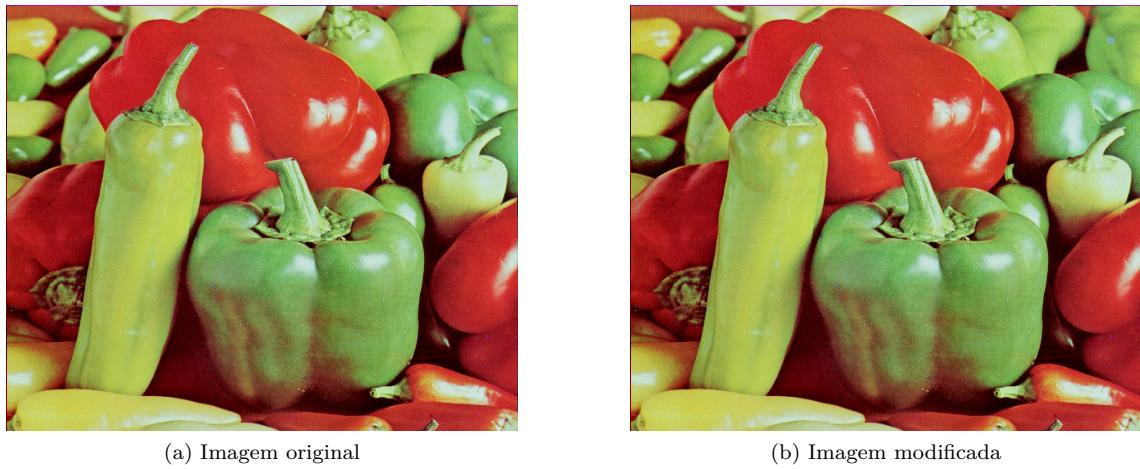


Figura 3: Aplicação do método de esteganografia na imagem **peppers.png** utilizando um arquivo de texto grande

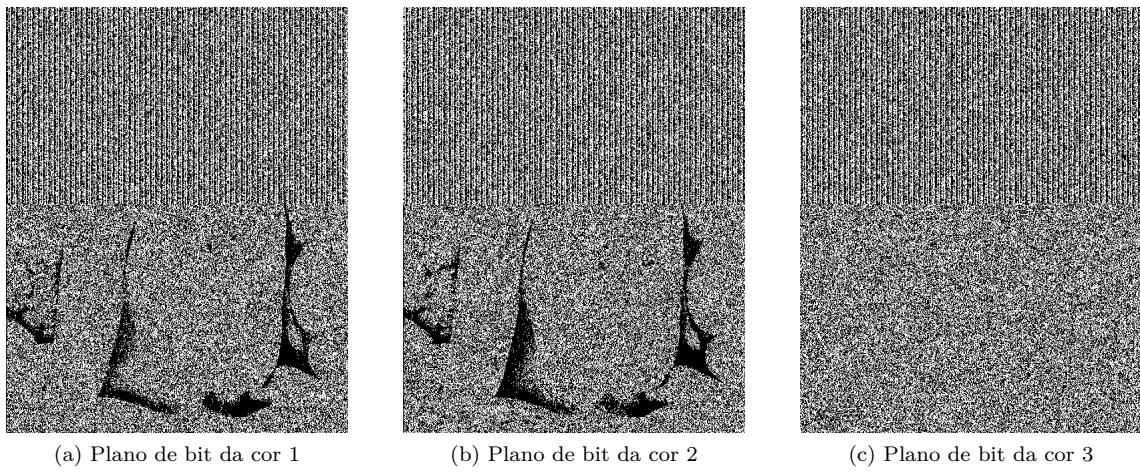


Figura 4: Planos de bits 2 da imagem gerada.

6.3 Imagem da Monalisa

O Terceiro teste foi feito utilizando uma imagem da **monalisa.png**. Para realizar esse teste a imagem **watch.png** foi utilizada como entrada e o plano de bit 2 foi escolhido como local para a codificação. Ao gerar a esteganografia, podemos observar que a imagem gerada possui uma pequena diferença visual comparada com a imagem de entrada (foram geradas algumas ondulações

na imagem na parte superior), mas nada que faça perceber de que tem algo de errado na imagem, provavelmente esse plano de bit tem a funcionalidade de retirar esse "ruído" na imagem. Porém, os planos de bits 2 gerados tiveram uma diferença perceptiva, mostrando o mesmo problema do resultado anterior. A imagem gerada pode ser observada na figura 5 e seus planos de bits 2 estão representados na figura 6.

Um outro teste foi feito, com os mesmos parâmetros do teste anterior, porém agora utilizando o plano de bit 7, para provar que bits mais significativos podem ser piores para guardar informações utilizando essa técnica. Podemos ver no resultado da figura 7, que a imagem gerada depois da esteganografia possui uma grande diferença visual da imagem original, mostrando que bits mais significativos não são ideais para aplicar essa técnica, os planos de bits 7 gerados podem ser observados na figura 8. No final das duas execuções a imagem foi recuperada corretamente pelo decodificador.

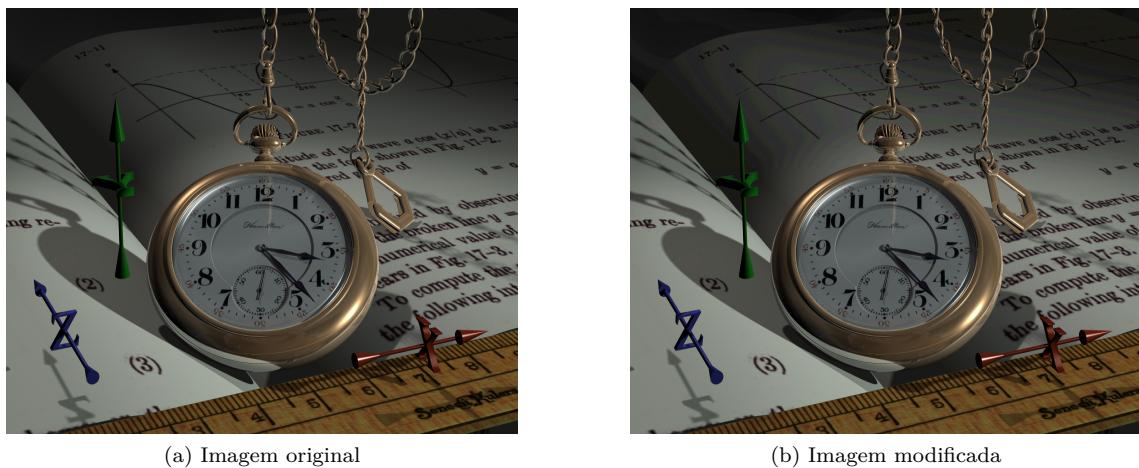


Figura 5: Aplicação do método de esteganografia na imagem **watch.png** utilizando a imagem da **monalisa.png**

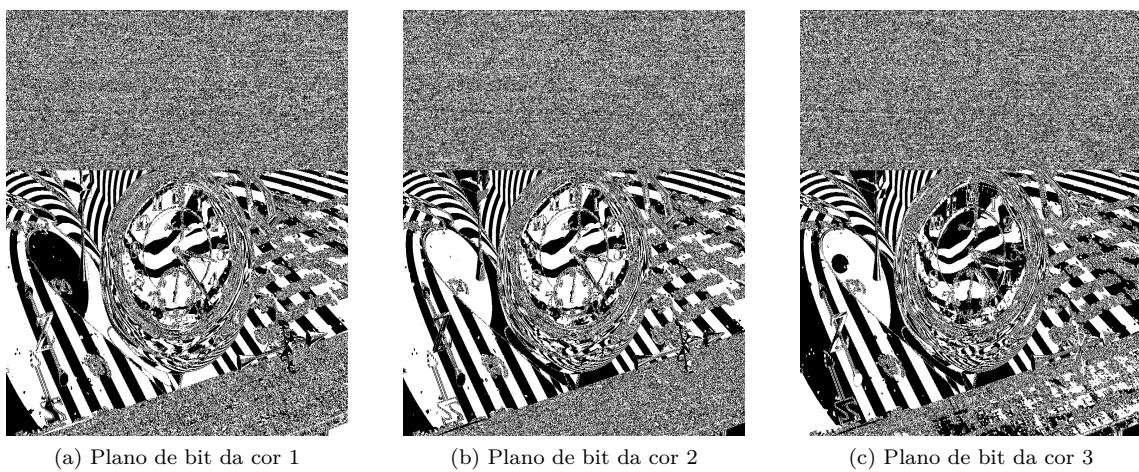
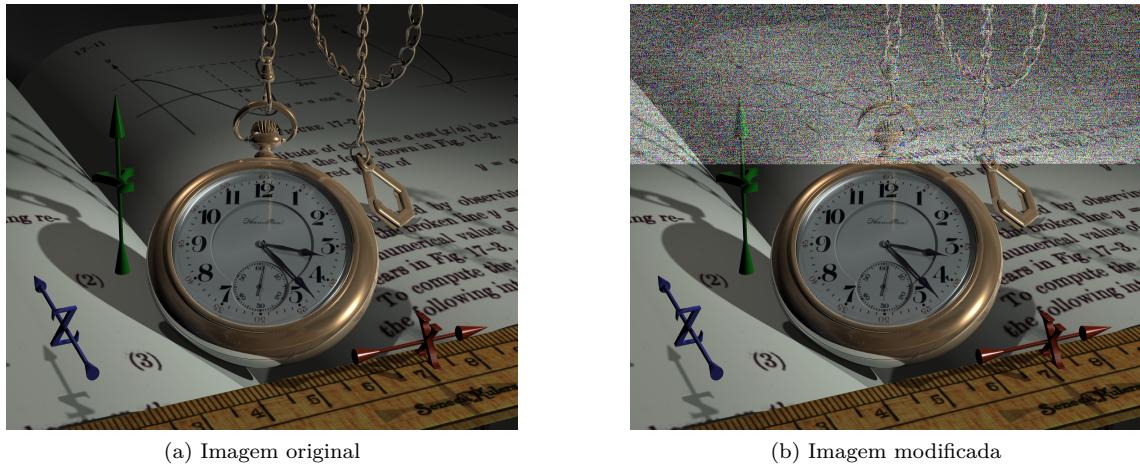


Figura 6: Planos de bits 2 da imagem gerada.

7 Conclusões

Pode-se perceber que todos os testes realizados foram bastante positivos, pois, o método conseguiu codificar e decodificar todos os arquivos, sem perder nenhuma informação. Mostrando que o método é super eficiente em transmitir informações dentro de imagens.



(a) Imagem original

(b) Imagem modificada

Figura 7: Aplicação do método de esteganografia na imagem **watch.png** utilizando a imagem da **monalisa.png**



Figura 8: Planos de bits 7 da imagem gerada.

Porém, caso uma pessoa que já conheça essa técnica, ocultar uma informação dentro de uma imagem pode ser uma tarefa difícil, pois ela pode ser facilmente percebida, pois, é nítido que os planos de bits são bastante modificados durante a aplicação da técnica, mostrando que a imagem foi modificada. Porém, pode se notar uma coisa bastante interessante, para imagens maiores (com uma maior amostragem), as informações podem estar mais “escondidas” dentro dos planos de bits, pois é bastante difícil de se perceber que os planos foram modificados por causa da grande quantidade de pixels.