

Introdução ao Processamento Digital de Imagem

Trabalho 01

Filipe Alves Sampaio
RA 249092

I. Introdução

Este trabalho tem como objetivo explicar o funcionamento, implementação e resultados do trabalho 01, desenvolvido na disciplina MO443. O trabalho tem como objetivo implementar um algoritmo de esteganografia em imagens digitais.

A seção II inclui exemplos de execução do programa, bem como os parâmetros necessários para sua execução e uma descrição, tanto dos parâmetros de entrada como das saídas geradas pelo script.

II. Execução

Dentro do *zip* referente ao trabalho, encontra-se dois arquivos: *codificar.py* e *decodificar.py*. Ambos os arquivos são códigos em *Python* desenvolvido para realizar as tarefas especificadas na descrição do trabalho. Nessa seção serão apresentados:

Dependências necessárias para a execução.

Funcionalidades implementadas e suas ações sobre as tarefas exigidas no trabalho.

Exemplos de execução.

A) Dependências

É opcional, mas aconselha-se a utilizar um ambiente de desenvolvimento para isolar as bibliotecas utilizadas nesse trabalho.

Assim, como orientado pelo professor da disciplina, utilize o Anaconda ou

semelhante e crie um ambiente separado. Após isso, instale as seguintes dependências:

- a) Python: Foi utilizado a versão 3.12.2. Essa versão é opcional, dado que os scripts desenvolvidos são simples e não utilizam algoritmos sofisticados que não existam em versões anteriores do Python.
- b) Numpy: Usada para manipular vetores e otimizar o *encode* da esteganografia.
- c) Sys: Usada para manipular argumentos passados por linha de comando.
- d) Opencv: Biblioteca usada para manipular imagens.
- e) Matplotlib: Biblioteca usada para gerar histogramas das imagens.

B) Funcionalidades

Essa seção tem como finalidade explicar brevemente o funcionamento do que foi implementado. Uma análise mais profunda dos algoritmos usados será descrita na seção III e uma análise dos resultados será discutida na seção IV.

Como descrito pelo trabalho, o objetivo é implementar um algoritmo de esteganografia em imagens digitais. Para isso, deve ser utilizado um processo de codificação, onde uma mensagem, sendo texto ou imagem, deve ser escondida dentro de uma imagem informada. Após isso, deve ser criado um processo de

decodificação dessa mensagem oculta na imagem, de tal forma a recuperar totalmente toda a informação.

C) Parâmetros

Neste tópico, é tratado os parâmetros possíveis para executar os *scripts* implementados. Seguindo o que foi informado pela descrição do trabalho, foi utilizado os seguintes parâmetros:

imagem_entrada.png: imagem, existente, no formato PNG em que será utilizada para ocultar a mensagem.

imagem_saida.png: nome e formato da imagem (deve ser sempre no formato PNG) com a mensagem embutida.

texto_entrada.txt: arquivo de texto contendo a mensagem a ser ocultada dentro da imagem.

texto_saida.txt: nome e formato do arquivo de texto (deve ser sempre .txt) contendo a mensagem recuperada após decodificação.

plano_bits: três planos de bits menos significativos representados pelos valores 0, 1 ou 2. (Pode ser informado mais de um).

Além desses parâmetros, é informado na seguinte lista, os arquivos que serão utilizados para execução do programa:

codificar.py: programa que oculta mensagem de texto na imagem.

decodificar.py: programa que recupera mensagem de texto da imagem.

D) Exemplos de Execução

No diretório do programa há duas pastas, que são: **/imgs** e **/tests**. Essas pastas

contêm, respectivamente, imagens de teste utilizadas na implementação do programa e arquivos de texto para codificar nas imagens de teste.

Ao finalizar a execução dos scripts *codificar.py* e *decodificar.py*, serão gerados imagens (resultantes do processo de codificação) e o texto de saída (resultante do processo de decodificação).

O programa pode ser executado da seguinte forma:

(a) Para codificação da mensagem na imagem:

```
python codificar.py  
<imagem_entrada.png>  
<texto_entrada.txt> [<plano_bits>]  
<imagem_saida.png>
```

Onde:

<imagem_entrada.png>: é a imagem, em formato .png, de entrada que será usada na esteganografia.

<texto_entrada.txt>: é o arquivo de texto no formato .txt que será usado para ser escondido na imagem.

[plano_bits]: é uma lista de bits a serem usados na codificação do texto. Logo, é possível mandar mais de um valor.

<imagem_saida.png>: é o nome da imagem, em .png, de saída da esteganografia, já com o texto escondido na imagem.

Veja a seguir alguns exemplos. Suponha que você deseja codificar utilizando os planos de bits 0 e 2 para uma determinada imagem, utilizando uma imagem de entrada chamado “*imagem_entrada.png*”:

```
python codificar.py
imagem_entrada.png texto_entrada.txt 0
2 imagem_saida.png
```

Executando esse comando, o resultado será a codificação do texto em binário dentro da imagem utilizando apenas os planos de bits 0 e 2 da imagem.

Após executar o comando, será plotado uma imagem contendo a imagem de entrada e a imagem codificada. Ao pressionar qualquer tecla, a imagem sairá e será *plotado* uma imagem contendo 6 colunas de imagens e 4 linhas, onde as 3 primeiras colunas são os canais R, G e B da imagem de entrada e as últimas 3 colunas representam os canais R, G e B da imagem codificada. As 4 linhas são, respectivamente, os planos de bit 0, 1, 2 e 7.

Após isso, basta pressionar qualquer tecla novamente para que a imagem suma. Por fim, será mostrado um histograma contendo uma contagem de pixels em tons de cinza das imagens de entrada e codificada. Para encerrar o processo, basta fechar a janela do *plot* do histograma e o programa se encerrará.

Todas as imagens *plotadas* na tela serão salvas no diretório raiz do programa. Isso foi feito para uma melhor análise do processo de codificação, caso seja necessário.

É possível visualizar no próprio terminal os parâmetros obrigatórios. Basta executar o comando:

```
python codificar.py
```

Aparecerá no terminal as orientações dos parâmetros obrigatórios.

(b) Para decodificação da mensagem na imagem

Para decodificar a mensagem na imagem, basta executar o seguinte comando:

```
python decodificar.py
<imagem_saida.png> [<plano_bits>]
<texto_saida.txt>
```

Onde:

<imagem_saida.png>: é a imagem de saída da codificação, ou seja, a imagem esteganografada.

[plano_bits]: é uma lista de bits a serem usados na decodificação do texto na imagem. Tenha certeza de usar os mesmos bits da codificação.

<texto_saida.txt>: é o arquivo .txt que salvará o texto oculto da imagem, após o processo de decodificação.

Veja agora alguns exemplos. Suponha que você deseja decodificar uma determinada mensagem que possui uma mensagem oculta na imagem. Para realizar a decodificação tenha em mente quantos e quais foram os planos de bits utilizados na codificação. Por exemplo, suponha que você precisa decodificar uma mensagem que foi codificada utilizando os planos 1 e 2. Com isso, basta executar:

```
python decodificar.py
imagem_saida.png 1 2 texto_saida.txt
```

O resultado ao executar esse comando será um *plot* da imagem que você informou. No caso do exemplo, é a imagem_saida.png. Ao pressionar qualquer tecla, a imagem será fechada e o processo de decodificação se iniciará. Basta acompanhar no terminal o processo se encerrar.

Também é possível visualizar, pela linha de comando, os parâmetros obrigatórios. Para isso, basta executar:

python decodificar.py

Aparecerá no terminal uma orientação dos parâmetros.

III. Implementação

Nessa seção serão descritos os passos e motivações que levaram ao estado final da implementação do programa. Juntamente, será feita uma descrição do funcionamento de cada etapa.

A) Código de codificação

Abrindo o arquivo *codificar.py*, será possível visualizar algumas funções implementadas para realizar a esteganografia.

ReadImage(): é a função utilizada para ler a imagem informada via linha de comando.

ShowImage(): é a função utilizada para *plotar* uma imagem na tela. Ela utiliza a biblioteca *opencv*.

SavelImage(): é a função utilizada para salvar uma imagem. Toda imagem salva ficará na raiz do programa.

ReadText(): função responsável por ler um arquivo texto.

EncodeText(): função responsável por codificar uma string (texto de entrada, vinda de um arquivo .txt) em sequência binária.

DecodeText(): função utilizada para decodificar uma string com binários de volta para texto.

CalculaCapacidadeEsteganografia(): função responsável por calcular quantos bits uma imagem dada consegue armazenar, em bits. O cálculo utilizado consiste em

multiplicar largura, altura, canais e planos de bits utilizados no processo de esteganografia.

ExtrairPlanoBit(): função utilizada para extrair apenas um plano de *bit* escolhido. Essa função é utilizada apenas para extrair o plano e *plotar* para visualização.

PrintPlanos(): função responsável por plotar o agrupamento dos planos de bits requisitados pelo trabalho (0, 1, 2 e 7) em uma imagem só.

Histograma(): função responsável por *plotar* histograma das imagens de entrada e codificadas, calculando a média dos pixels para tons de cinza e *plotar* apenas um canal.

esteganografia_v1(): função que implementa a primeira versão do algoritmo. Complexidade $O(n^4)$.

esteganografia_v2(): função que refatora a função *esteganografia_v1()*, utilizando vetorização. Complexidade $O(n^2)$.

Main(): função principal que orquestra o script.

Neste trabalho, a ideia para realizar a esteganografia vem da estratégia de salvar todos os binários do texto de forma sequencial, por pixel, conforme percorre-se os canais da imagem.

Para melhor exemplificar, tome uma imagem com 512x512 pixels e 3 canais, R, G e B. O algoritmo de esteganografia percorrerá todos os pixels da imagem, e para cada pixel, este será convertido em binário, para obter sua representação de 8 bits.

Com o pixel selecionado, será feito uma iteração sobre os planos de bits escolhidos

pelo usuário. Por exemplo, assuma que um usuário escolheu os bits 0, 1 e 2. Assim, será substituído os bits 0, 1 e 2 do pixel selecionado pelos bits disponíveis do texto, sequencialmente.

Só após preencher todo o pixel, o algoritmo seleciona um próximo pixel e refaz o processo, enquanto houver bits de texto. Quando acabar os bits de texto, o algoritmo pára e monta a imagem esteganografada.

Para realizar a decodificação da esteganografia corretamente, é preciso utilizar alguma técnica para saber quantos bits ler quando for realizada a decodificação da imagem codificada.

Primeiramente foi abordado o uso de um delimitador. A ideia é simplista, onde consistia em adicionar no final dos bits do texto um caractere especial que denota-se o final do texto codificado na imagem.

Porém, essa estratégia não funciona quando há textos muito grandes, por exemplo, com vários caracteres especiais. O algoritmo acaba pulando o delimitador.

Assim, para solucionar esse problema, foi implementado uma solução mais robusta, que adiciona no início da mensagem uma sequência de 32 bits, referentes ao tamanho da mensagem.

Isso é necessário, pois para que, na decodificação, seja possível saber a posição do vetor de bits da imagem que deve ser cortado para extrair os bits exatos da mensagem codificada.

Foi implementado duas versões do algoritmo para codificação da esteganografia, uma sem vetorização e outra com. Notou-se uma significativa diferença durante a execução devido a utilização de vetorização, além da redução da complexidade algorítmica.

Além disso, foi implementado, na codificação do texto, palavras de 16 bits. Inicialmente foram utilizados apenas 8 bits, mas percebeu-se que, dependendo do tamanho da mensagem, essa codificação com apenas 8 bits não era o suficiente, ocorrendo então perda de dados na etapa de conversão de texto para binário. Assim, para solucionar esse problema, o texto agora é codificado para 16 bits, porém, se for utilizado mensagens ainda maiores (por exemplo, maior que o arquivo *tests/revolucao_portuguesa_05_de_outubro.txt*, será necessário aumentar o tamanho da palavra).

Em experimentos, notou-se que esse aumento do tamanho da palavra afeta drasticamente a qualidade da imagem, tornando bem perceptível o ocultamento de mensagem na mensagem. Na seção IV será apresentado exemplos de testes onde isso ocorre.

B) Código de decodificação

Agora, abrindo o arquivo *decodificar.py*, será possível visualizar algumas funções implementadas. Muitas delas são iguais às funções do arquivo *codificar.py*. São elas:

ReadImage(): é a função utilizada para ler a imagem informada via linha de comando.

ShowImage(): é a função utilizada para *plotar* uma imagem na tela. Ela utiliza a biblioteca *opencv*.

SaveText(): função responsável por salvar o texto em um arquivo .txt na raiz do programa.

EncodeText(): função responsável por codificar uma string (texto de entrada, vinda de um arquivo .txt) em sequência binária.

`DecodeText()`: função utilizada para decodificar uma string com binários de volta para texto.

`decodificar_esteganografia_v1()`: é a função responsável por decodificar uma imagem esteganografada. Possui complexidade $O(n^4)$.

`Main()`: função principal que orquestra o script.

Na etapa de decodificação da esteganografia, é possível visualizar o processo na função `decodificar_esteganografia_v1()`. Nessa função, que não possui vetorização, recebe como entrada a imagem codificada pelo arquivo `codificar.py`, e então realiza o processo reverso das funções `esteganografia_v1()` e `esteganografia_v2()`.

O seu funcionamento parte do princípio de percorrer todos os pixels da imagem de entrada, no caso sendo a imagem codificada, e, para cada pixel, percorrer os planos de bits escolhidos pelo usuário e coletar, na sequência da leitura, os bits do plano de bit.

Ao final, é calculado os 32 bits iniciais referentes ao tamanho da mensagem, utilizando então essa informação para então realizar um corte no vetor de bits lidos da imagem até o comprimento total da mensagem.

Ou seja, nos primeiros 32 bits da imagem, haverá um comprimento da mensagem ocultada. Após obter esse comprimento, basta percorrer os bits da imagem até completar os 32 bits e a mensagem ocultada será extraída, sem nenhuma perda.

C) Validação e Apresentação dos Dados

Durante a codificação, é feito um cálculo para avaliar a capacidade da imagem e validar se ela tem espaço suficiente para armazenar todos os bits da mensagem sem perder nada. Isso é feito no arquivo `codificar.py`, nas linhas 205 a 216. O cálculo consiste na multiplicação das dimensões da imagem pela quantidade de canais da mesma e pela quantidade de planos de bits a serem usados na codificação da mensagem. Se isso não for feito, a mensagem pode não ser totalmente armazenada na imagem, pois como o trabalho exige a utilização apenas no planos 0, 1 e 2, dependendo da situação, isso pode não ser o suficiente para armazenar toda a mensagem, resultando então em perda da própria mensagem no momento da decodificação. Veja um exemplo abaixo na figura 1:

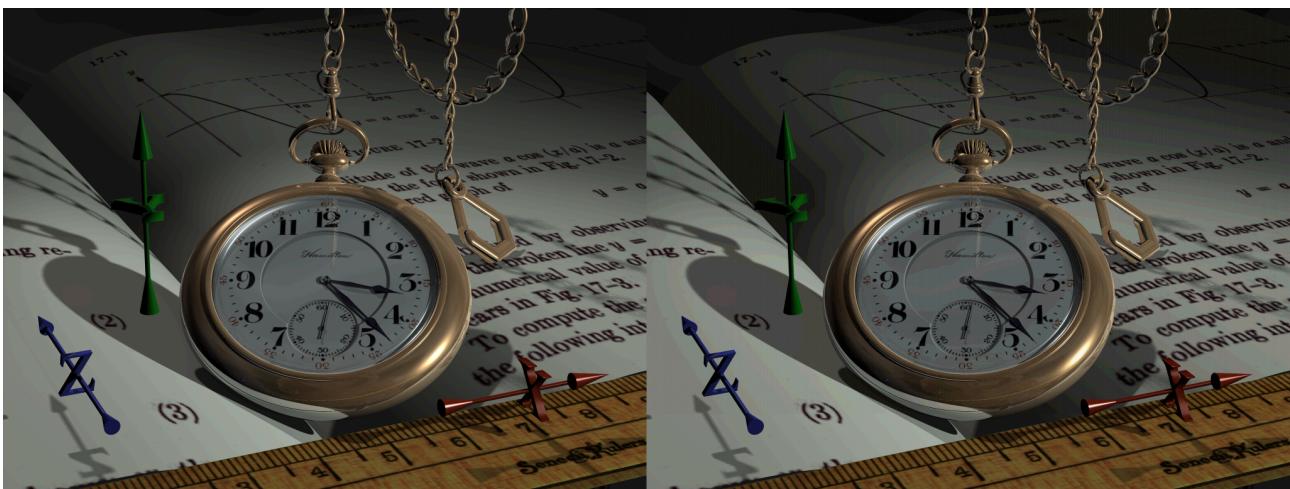
```
== ESTEGANOGRAFIA -> CODIFICADOR ==
-----
Bits que cabem na imagem de entrada usando os planos_bits [0 1 2] informados: 2359296
Bits do texto a ser escondido: 928
-----
```

Figura 1. Print do terminal mostrando cálculo de capacidade da imagem e da mensagem.

IV. Resultados

Nessa seção é apresentado alguns experimentos realizados durante a implementação. Observe na figura 2 e 3 o resultado da codificação de uma mensagem que contém o livro “Revolução Portuguesa 05 de outubro”, na pasta `tests/` na imagem `watch.png`. Para visualizar com mais detalhes as imagens geradas, execute o comando:

```
python codificar.py imgs/watch.png
tests/revolucao_portuguesa_05_de_outubro.txt 0 1 2 imagem_saida.png
```



Figuras 2 e 3. A esquerda, imagem original. A direita, imagem codificada com esteganografia.

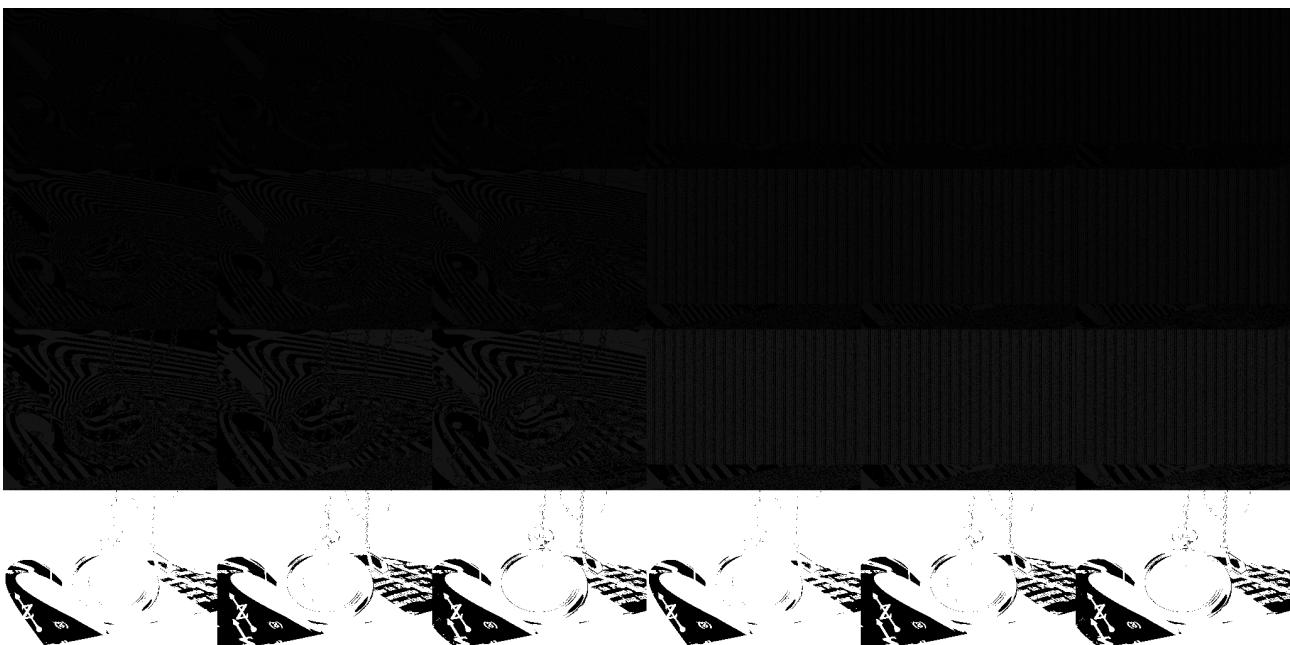


Figura 4. Planos de bits nas linhas (0, 1, 2 e 7) e canais RGB nas colunas. As três primeiras colunas são os canais RGB da imagem de entrada, e as três últimas são da imagem codificada.

O processo salvará as imagens `entrada_e_saida.png`, `histograma.png`, `imagem_saida.png` e `planos_de_bits.png` na pasta do programa.

É possível observar o efeito da degradação dos bits 0, 1 e 2 nesse exemplo, onde na figura 3, na região acima do relógio, há suaves ondulações, o que não havia na figura 2 original.

Isso se deve ao fato da mensagem de entrada preencher quase todos os bits dos planos 0, 1 e 2 da imagem. Pode-se ver isso na figura 4, apresentando os planos de bits em cada um dos canais R, G e B.

Na figura 4, observa-se que os bits menos significativos tendem a não ser relevantes para representar as informações na imagem. Isso porque,

após a esteganografia na figura 3, não houve degradação na imagem a ponto de ser visivelmente diferente da original.

É possível visualizar também que, quando os planos de bits menos significativos são plotados, os pixels são bem escuros. A imagem de plano de bits menos significativo geralmente parece quase totalmente preta porque os bits menos significativos contêm a menor contribuição para a intensidade de cor de um pixel.

Assim, as diferenças entre os valores dos bits menos significativos podem ser muito pequenas e difíceis de serem percebidas a olho nu.

Devido a isso, foi aumentado o contraste dessas imagens para aumentar o detalhe na figura 4, para que fosse possível visualizar com mais detalhes a diferença.

Observe que, as três colunas da figura 4 estão mantendo um certo padrão, com ondulações. Porém, após a esteganografia, aparece linhas verticais preenchendo quase todas as imagens dos planos 0, 1 e 2. O plano de *bit* 7 não é afetado, e nunca será, pois devido a codificação do algoritmo, é garantido que somente os bits 0, 1 e 2 serão usados na codificação.

Com isso, para aumentar os detalhes na observação das imagens de entrada e codificadas, podemos observar na figura 5, o histograma das figuras 2 e 3.

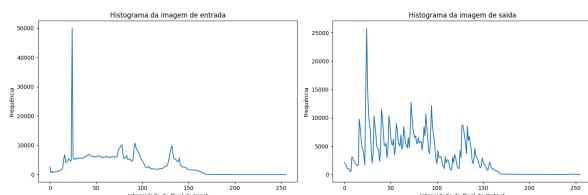


Figura 5. Histograma da imagem de entrada (a esquerda) e imagem codificada (a direita).

Percebe-se claramente uma perturbação nos pixels devido a codificação da mensagem na imagem de entrada, confirmando então que houve escrita de bits na imagem.

Executando um outro experimento, agora com uma mensagem menor, na pasta *tests/revolucao_portuguesa_05_de_outubro-medio.txt* e a imagem *peppers.png*, foi realizado os mesmos experimentos para avaliação. Segue o comando utilizado para realizar esse experimento:

```
python codificar.py imgs/peppers.png
tests/revolucao_portuguesa_05_de_outubro-resumo.txt 0 1 2
imagem_saida.png
```

Nesse exemplo, não é visível nenhuma alteração nas imagens, conforme é observado nas figuras 6 e 7.

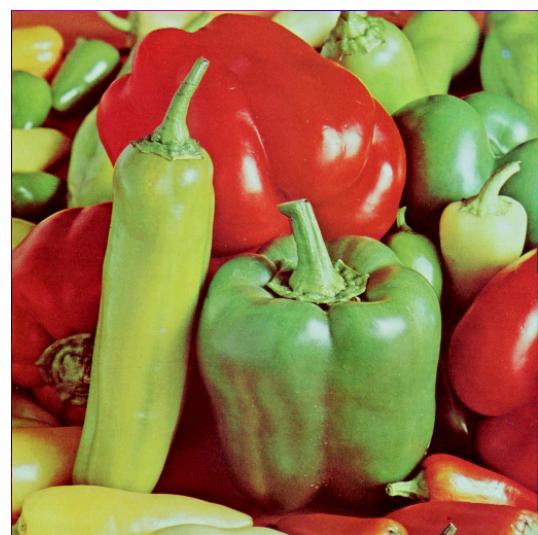


Figura 6. Imagem de entrada (sem esteganografia).

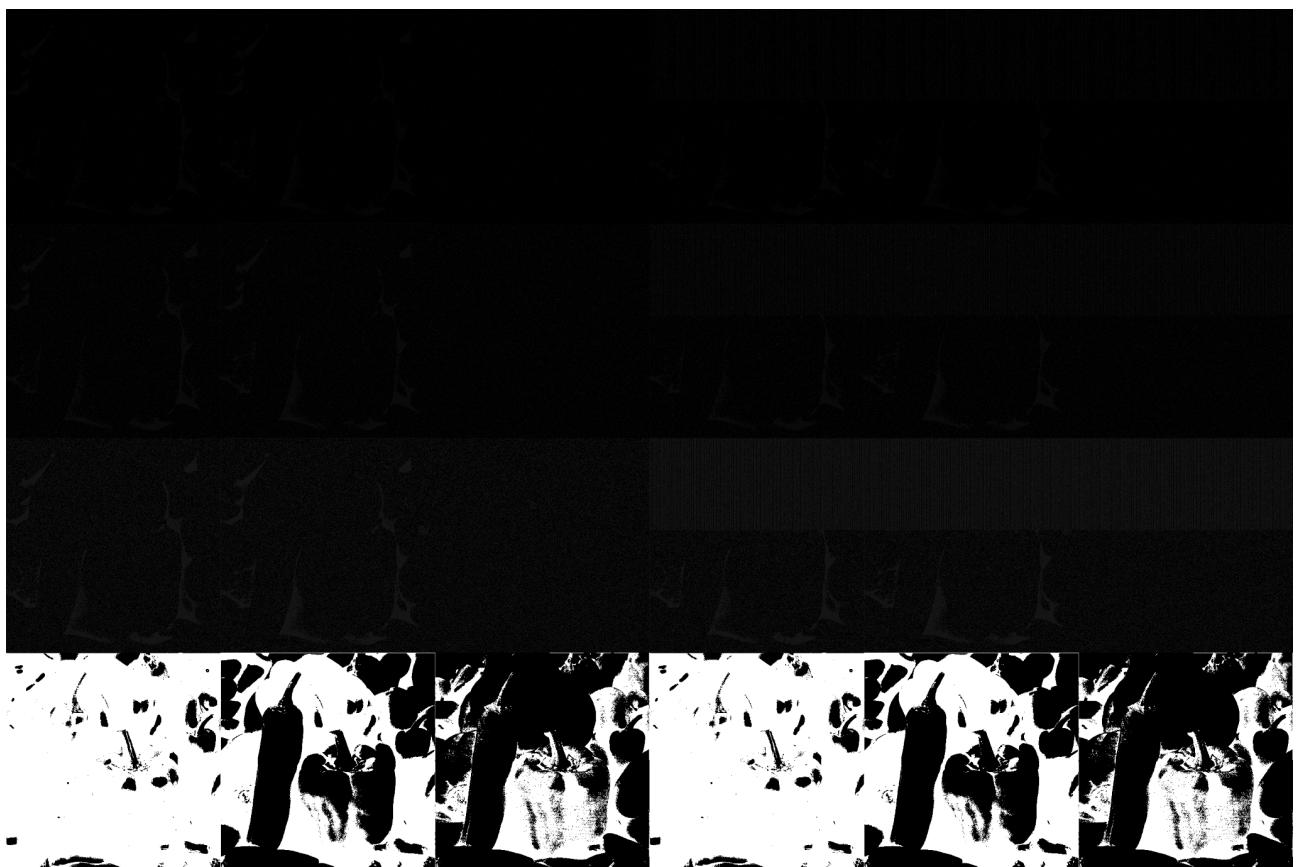


Figura 8. Planos de bits nas linhas (0, 1, 2 e 7) e canais RGB nas colunas. As três primeiras colunas são os canais RGB da imagem de entrada, e as três últimas são da imagem codificada.

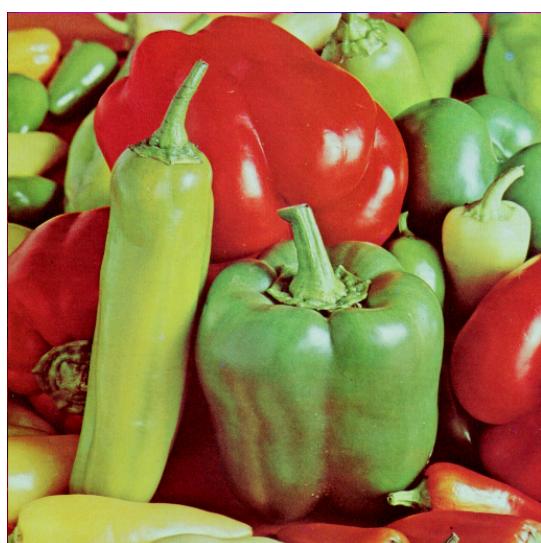


Figura 7. Imagem codificada (com esteganografia).

Porém, quando são avaliados os planos de bits 0, 1, 2 e 7, é possível notar que

houve modificação, conforme vê-se na figura 8. Observe que há linhas verticais nos bits 0, 1 e 2, porém no bit 7, novamente não houve alterações.

Reitera-se que, seria possível a esteganografia manipular o bit 7 de forma indireta, com a manipulação dos bits 0, 1 e 2. Para isso, a codificação da esteganografia deve obrigatoriamente adicionar os bits da mensagem dentro da imagem, começando pelos bits menos significativos. O trabalho pede apenas que os bits 0, 1 e/ou 2 sejam substituídos.

Devido a isso, o bit 7 jamais será manipulado. Mas, caso fosse feita a

adição ao invés da substituição, seria possível, além de manipular indiretamente o bit 7, recuperar a imagem defeituosa da codificação.

No histograma da figura 9 pode-se visualizar novamente a perturbação dos pixels, onde as linhas apresentam-se mais pontudas, ao contrário da imagem original, onde é mais suave.

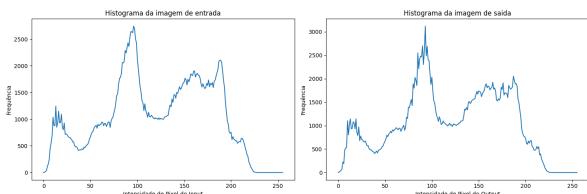


Figura 9. Histograma da imagem de entrada (a esquerda) e imagem codificada (a direita).

É levantado uma discussão sobre os efeitos que ocorrem nos planos de bits. Como comentado na seção III, devido a utilização de palavras de 16 bits na mensagem, é notável uma maior degradação da imagem devido ao tamanho da mensagem. Foram realizados testes com palavras de 8 bits que colocavam metade das linhas verticais nas figuras 4 e 8.

Porém, como mencionado anteriormente, foi definido palavras de 16 bits para conseguir codificar mensagens maiores dentro das imagens sem perder informações.

Por fim, na decodificação, para o primeiro exemplo apresentado nessa seção, execute o comando:

```
python decodificar.py  
imagem_saida.png 0 1 2  
texto_saida.txt
```

Aparecerá na tela a imagem de entrada, que deve ser a imagem de saída do codificador. Com isso, será gerado um arquivo .txt com a mensagem original utilizada. O mesmo pode ser feito para o segundo exemplo, obtendo o mesmo resultado.

V. Conclusões

Neste trabalho foi possível explorar como realizar manipulação em bits da imagem. Esse processo consistiu de realizar esteganografia em uma imagem, tendo como resultado final, perturbações nos pixels da imagem.

Mesmo manipulando bits menos significativos, dependendo do tamanho da mensagem e do tamanho da palavra de bits da mensagem, pode ser facilmente visível a manipulação na imagem colorida.