

Escondendo uma mensagem em uma imagem (trabalho 1)

Introdução ao Processamento de Imagem Digital

Randerson A. Lemos (103897) 2022-1S

1 Introdução

A esteganografia é a área de conhecimento que se dedica a estudar técnicas de ocultação de informações (por exemplo, mensagens) em imagens sem que estas sofram alterações perceptíveis ao olho humano. A esteganografia, neste trabalho, será aplicada por meio da substituição dos bits menos significativos dos pixels das imagens escolhidas pelos bits das mensagens de interesse que se deseja ocultar. A seguir teremos três seções, a de Solução, a de Resultado e a de Conclusão. Na seção de Solução, detalhes técnicos, de usabilidade e de decisões da solução proposta são fornecidos. Na seção de Resultado, os principais resultados são apresentados. Na seção de Conclusão, há não apenas a apresentação de conclusões, mas também de discussões.

2 Solução

A solução utiliza a linguagem de programação Python e conta com o auxílio do gerenciador de projetos e pacotes Conda. Assumindo que o usuário tenha o Conda instalado em sua máquina, a configuração do projeto pode ser feita pela execução do comando `conda env create -f environment.yml` a partir da pasta do projeto **trab1**. Esse comando cria o ambiente de trabalho **mc920-trab1** e instala os seguintes módulos: opencv, numpy, scipy, pandas, matplotlib. Finalizada a configuração do ambiente de trabalho em questão, o usuário deve executar o comando `source source.sh`¹ para carregar as variáveis de ambiente adequadas e, assim, poder usar os programas do projeto dentro do próprio ambiente de trabalho recém configurado.

Dos arquivos presentes na pasta do projeto **trab1**, destacam-se as pastas **png**, **tex**, **txt** e os programas **codificar.py**, **decodificar.py**, **mostrar_planos.py**. A pasta **png** contém imagens no formato png que podem ser utilizadas como esconderijos das mensagens a serem ocultadas. A pasta **tex** contém exemplos de textos que podem ser utilizados no processo de ocultação de suas mensagens. A pasta **tex** contém os arquivos Latex deste relatório. As informações pertinentes dos programas **codificar.py**, **decodificar.py**, **mostrar_planos.py** são detalhadas a seguir.

2.1 Codificar.py

O programa **codificar.py** é responsável por ocultar uma mensagem de interesse em uma imagem escolhida que deve ser colorida e estar no formato png. Para ser executado, esse programa precisa receber os parâmetros **imagem_entrada**, **texto_entrada**, e **planos_bits**:

- ao parâmetro **imagem_entrada** deve-se fornecer o nome do arquivo da imagem a ser utilizada como ‘bau’ da mensagem que se deseja esconder;
- ao parâmetro **texto_entrada** deve-se fornecer o nome do arquivo do texto que contém a mensagem que se deseja esconder;
- ao parâmetro **planos_bits** deve-se fornecer os planos de bits menos significativos dos pixels da imagem escolhida que serão utilizados para guardar os bits da mensagem de interesse. Os valores esperados para esse parâmetro são: 0, ou 1, ou 2, ou combinações desses valores separados por ‘:’. Quando mais de um plano de bits são passados ao programa pelo parâmetro **planos_bits**, ocorre a ordenação em ordem crescente desses planos de modo que a utilização dos bits dos pixels da imagem se dê sempre do plano de bits menos significativo para o mais significativo.

O texto de entrada que contém a mensagem a ser escondida na imagem escolhida deve apresentar um código de *end of file* para que o processo subsequente de decodificação consiga identificar o final de mensagem. Aqui,

¹O comando que configura o ambiente de trabalho mc920-trab1 precisa ser executado apenas um vez. Assim sendo, depois que este ambiente está configurado, o usuário precisa apenas executar o comando `source source.sh`

o código escolhido é !@#FIM#@!. A combinação dos caracteres desse código foi pensada de modo que uma possível aparição não intencional de tal código ao longo do texto seja significativamente improvável.

Exemplos de como utilizar o programa **codificar.py** utilizando os recursos contidos dentro do próprio projeto são:

```
python3 codificar.py -imagem_entrada=png/watch.png -texto_entrada=txt/texto1.txt -planos_bits=2;
python3 codificar.py -imagem_entrada=png/watch.png -texto_entrada=txt/texto1.txt -planos_bits=1:2;
python3 codificar.py -imagem_entrada=png/watch.png -texto_entrada=txt/texto1.txt -planos_bits=0:1:2.
```

Após executado, o programa **codificar.py** gera uma imagem de saída. Esse imagem nada mais é que a imagem de entrada contendo a mensagem do texto de entrada oculta nos seus bits menos significativos de acordo com os planos de bits passados pelo usuário. Essa imagem de saída é salva automaticamente na pasta **out** que está dentro da pasta do projeto **trab1**. Se a imagem de entrada tem o nome **img_ent.png** a imagem de saída apresentará o nome **img_entm_planoX.png**, onde o letra X é um *placeholder* para os números dos planos de bits escolhidos pelo usuário.

Para demais informações acerca das implementações ou algoritmos utilizados, consultar o código fonte. Lá existem comentários que abrangem esse tipo de informação.

2.2 Decodificar.py

O programa **decodificar.py** é responsável por recuperar a mensagem escondida em um imagem previamente codificada com esse mensagem pelo programa **codificar.py**. Para ser executado, esse programa precisa receber os parâmetros **imagem_entrada** e **planos_bits**:

- ao parâmetro **imagem_entrada** deve-se fornecer o nome do arquivo da imagem que possui uma mensagem escondida cujo conteúdo se deseja recuperar;
- ao parâmetro **planos_bits** deve-se fornecer os valores numéricos dos planos de bits nos quais os bits da mensagem escondida foram guardados. Essa parâmetro aceita os seguintes valores: 0, ou 1, ou 2, ou combinações desses valores separados por ‘:’.

Exemplos de como executar o programa **decodificar.py** utilizando os recursos² contidos dentro do próprio projeto são:

```
python3 decodificar.py -imagem_entrada=out/watchm_plano2.png -planos_bits=2;
python3 decodificar.py -imagem_entrada=out/watchm_plano01.png -planos_bits=0:1;
python3 decodificar.py -imagem_entrada=out/watchm_plano012.png -planos_bits=0:1:2.
```

Após executado, o programa **decodificar.py** é responsável por gerar um arquivo de texto contendo a mensagem que estava escondida na imagem repositório. Esse texto é salvo automaticamente na pasta **out** que está dentro da pasta do projeto **trab1**. Se a imagem com a mensagem escondida tiver o nome **img_entm_planoX.png** o arquivo texto da mensagem terá o nome **img_entm_planoX.txt**.

Para demais informações acerca das implementações ou algoritmos utilizados, consultar o código fonte. Lá existem comentários que abrangem esse tipo de informação.

2.3 Mostrar_planos.py

O programa **mostrar_planos.py** é responsável por gerar mapas de bits dos três canais (RGB) de uma imagem selecionada. Para funcionar esse programa precisa receber os parâmetros **imagem_entrada** e **planos_bits**:

- ao parâmetro **imagem_entrada** deve-se fornecer o nome do arquivo da imagem que se deseja visualizar os planos de bits;
- ao parâmetro **planos_bits** deve-se fornecer os valores numéricos dos planos de bits que se deseja visualizar. Essa parâmetro aceita valores de 0 (referente ao plano de bits menos significativo) até 7 (referente ao plano de bits mais significativo). Para gerar imagens de múltiplos planos de bits, basta passar os valores dos planos de interesse separados por ‘:’.

Exemplos de como executar o programa **mostrar_planos.py** utilizando os recursos² contidos dentro do próprio projeto são:

```
python3 mostrar_planos.py -imagem_entrada=out/watchm_plano12.png -planos_bits=3,
python3 mostrar_planos.py -imagem_entrada=out/watchm_plano12.png -planos_bits=0:1:2:7.
```

Após executado, o programa **mostrar_planos.py** gera imagens dos planos de bits passados pelo usuário dos canais RGB da imagem selecionada. Essa imagem de saída é salva automaticamente na pasta **out** que está dentro da pasta do projeto **trab1**. Se a imagem de entrada tem o nome **img_entm_planoX.png**, as imagens de

²É necessário que o comando adequado do programa codificar.py tenha sido executado.

saída apresentarão nome segundo o padrão **img_entm_planoX_plano_bits_Y_Z.png** em que Y é o *placeholder* para o número do plano de bits apresentado na imagem e Z é o *placeholder* para os canais (R, G ou B) dos planos de bits.

Para demais informações acerca das implementações ou algoritmos utilizados, consultar o código fonte. Lá existem comentários que abrangem esse tipo de informação.

3 Resultado

Os resultados levantados são provenientes da aplicação dos programas **codificar.py**, **decodificar.py**, **mositar_planos.py** sobre a imagens apresentadas na Figura 1.

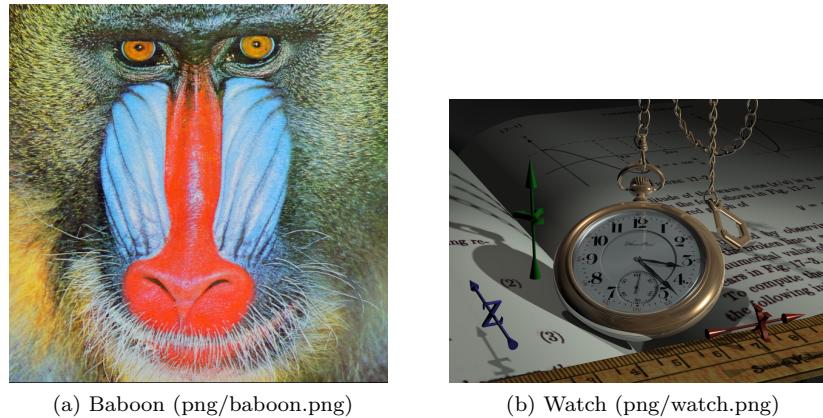


Figura 1: Imagens utilizadas para aplicação da solução de esteganografia.

Ambas as imagens estão no formato requerido (png) e estão localizadas na pasta **png** que está dentro da pasta do projeto **trab1**. A imagem Baboon apresenta dimensões de 512x512. A imagem Watch apresenta dimensões de 1024x768.

Duas mensagens serão aplicadas em cada uma desses imagens (em momentos diferentes). Uma mensagem é **TAMO JUNTO!!!** que está armazenada no arquivo **texto1.txt**, localizado na pasta **txt** que está dentro da pasta do projeto **trab1**. Importante salientar que nessa mensagem também há o conjunto de caracteres **!@#FIM#@!** que codificam o final de mensagem. A outra mensagem é uma sequência de letras 'a's cujo tamanho é suficiente para encher um plano de bits dos três canais completamente da imagem Baboon. Essa mensagem está armazenada no arquivo **texto_longo_1.txt** que está também localizada na pasta **txt**.

Os comandos utilizados para ocultação da mensagem contida no arquivo **texto1.txt** foram:

```
python3 codificar.py -imagem_entrada=png/baboon.png -texto_entrada=txt/texto1.txt -planos_bits=0;
python3 codificar.py -imagem_entrada=png/watch.png -texto_entrada=txt/texto1.txt -planos_bits=0.
```

As imagens resultantes com a mensagem escondida estão apresentadas na Figura 2.

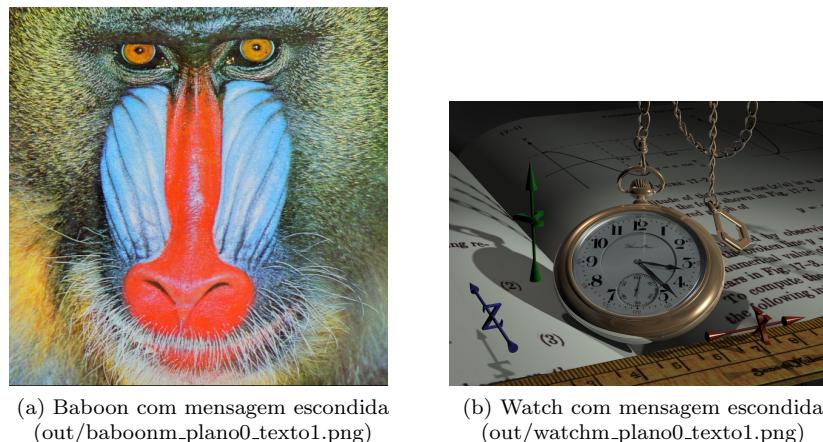
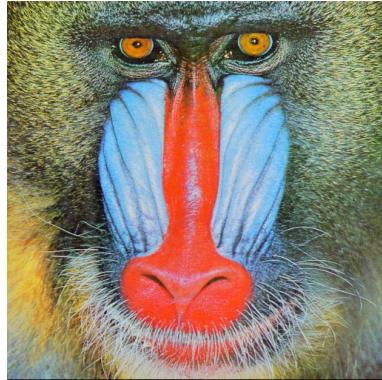


Figura 2: Imagens que apresentam mensagem escondida do arquivo **texto1.txt**.

Os comandos utilizados para ocultação de mensagem contida no arquivo **texto_longo_1.txt** foram:

```
python3 codificar.py -imagem_entrada=png/baboon.png -texto_entrada=txt/texto_longo_1.txt -planos_bits=0;
python3 codificar.py -imagem_entrada=png/watch.png -texto_entrada=txt/texto_longo_1.txt -planos_bits=0.
```

As imagens resultantes com a mensagem escondida estão apresentadas na Figura 3.



(a) Baboon com mensagem escondida
(/out/babo-
onm_plano0_texto_longo_1.png)



(b) Watch com mensagem escondida
(out/wat-
chm_plano0_texto_longo_1.png)

Figura 3: Imagens que apresentam mensagem escondida do arquivo do **texto_longo_1.txt**.

Nota-se que visualmente, essas imagens, apesar de modificadas pelos bits da mensagem escondida, não apresentam alterações visuais perceptíveis ao olho humano. Vamos visualizar os planos 0, 1, 2 e 7 dessas imagens para cada uma das duas mensagens **texto1.txt** e **texto_longo_1.txt**.

3.1 Planos de bits da imagem Baboon e Watch modifica pela mensagem contida no arquivo **texto1.txt**

Começaremos visualizando os planos de bits da imagem do Baboon modificada. Os planos apresentados serão os 0, 1, 2, e 7 (0 menos significativo, 7 o mais significativo). As imagens dos planos de bits podem ser visualizadas na Figura 4.

Para as imagens apresentadas na Figura 4, notamos que não é possível visualmente verificar que a imagem foi modificada a partir da análise visual dos seus planos de bits. Essa condição deve ser propiciada tanto pela ruído apresentado nos planos de bits menos significativos quanto pelo tamanho da mensagem que é pequeno. Também podemos perceber que os planos bits menos significativos carregam menos informações da composição da imagem do que os planos de bits mais significativos. Logo, é por isso que modificar esse bits (os menos significativos) não altera perceptivelmente a imagem de saída.

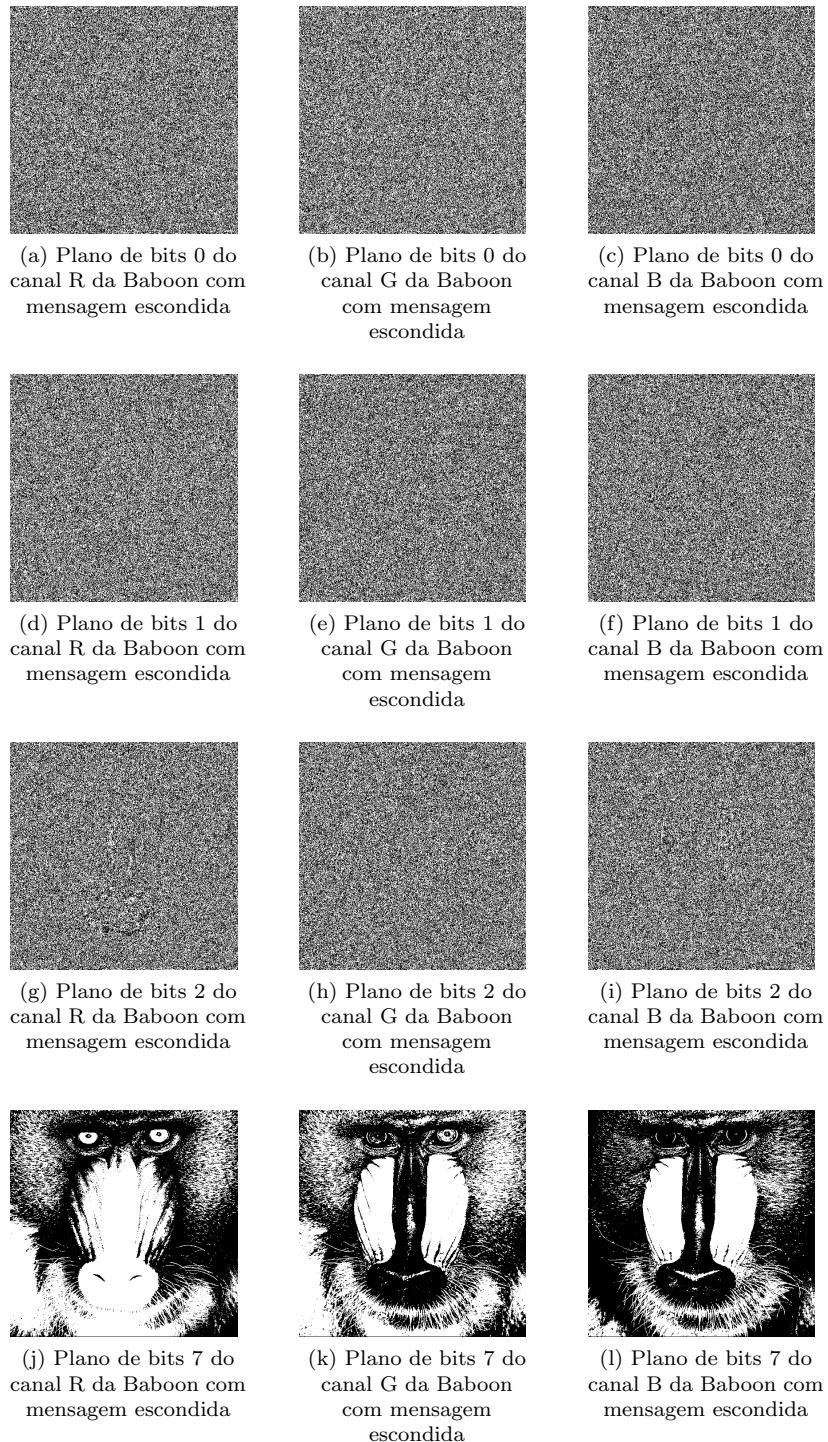


Figura 4: Planos de bits da imagem do Baboon modificada pelos bits da mensagem do arquivo **texto1.txt**.

Agora vamos verificar os planos de bits 0, 1, 2 e 7 da imagem Watch modificada. Esses planos de bits estão apresentados na Figura 5.

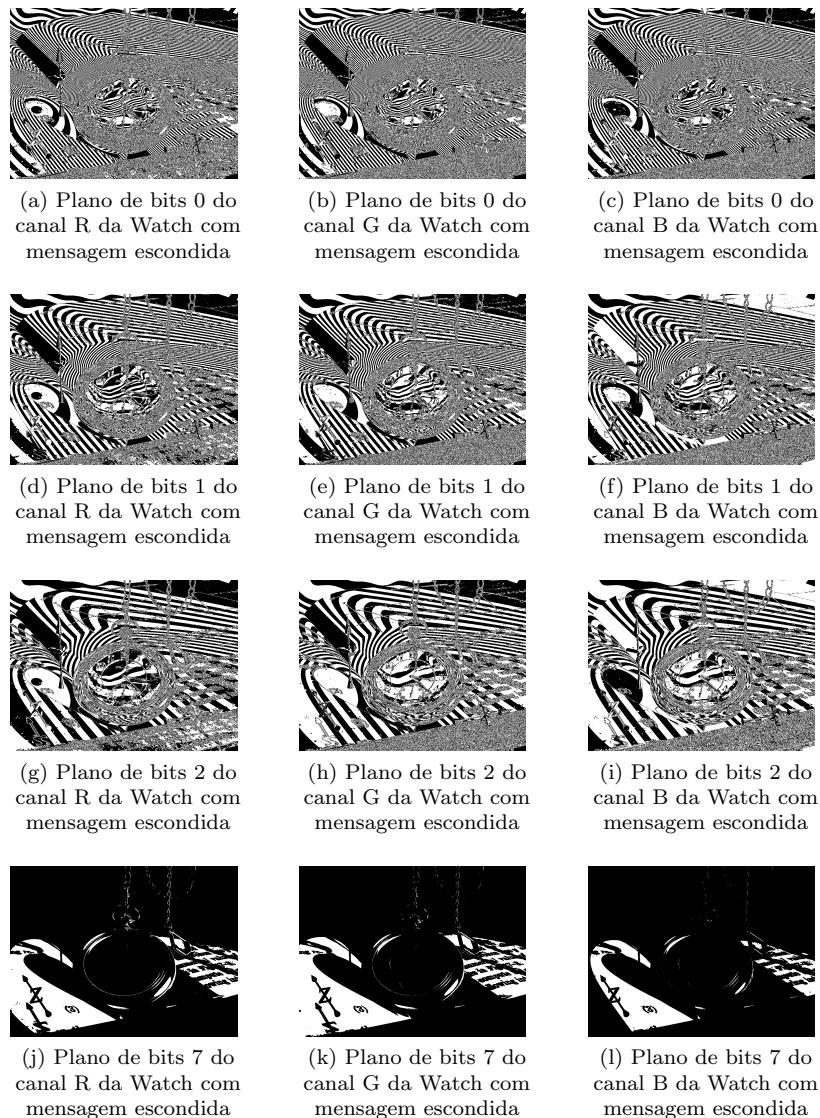


Figura 5: Planos de bits da imagem do Watch modificada pelos bits da mensagem do arquivo **texto1.txt**.

No caso dessa imagem, é possível notar uma alteração dos padrões de bits apresentados no seu plano de bits 0 (que é o plano onde a mensagem foi inserida). A visualização dessa quebra de padrão está dificuldade pelo tamanho da imagem, mas um olhar atento ainda é capaz de verificar (olhar canto superior esquerdo da primeira linha de imagens).

3.2 Planos de bits da imagem Baboon e Watch modifica pela mensagem contida no arquivo **texto_longo_1.txt**

Começaremos visualizando os planos de bits da imagem do Baboon modificada. Os planos apresentados serão o 0, 1, 2, e 7 (0 menos significativo, 7 o mais significativo). As imagens dos planos de bits podem ser visualizadas na Figura 6.

Para as imagens apresentadas na Figura 6, notamos que é possível visualmente verificar que a imagem foi modificada a partir da análise visual dos seus planos de bits. Logo, para mensagens muito grandes, pode-se constatar que é possível verificar que uma imagem foi modificada pela análise dos padrões dos seus planos de bits.

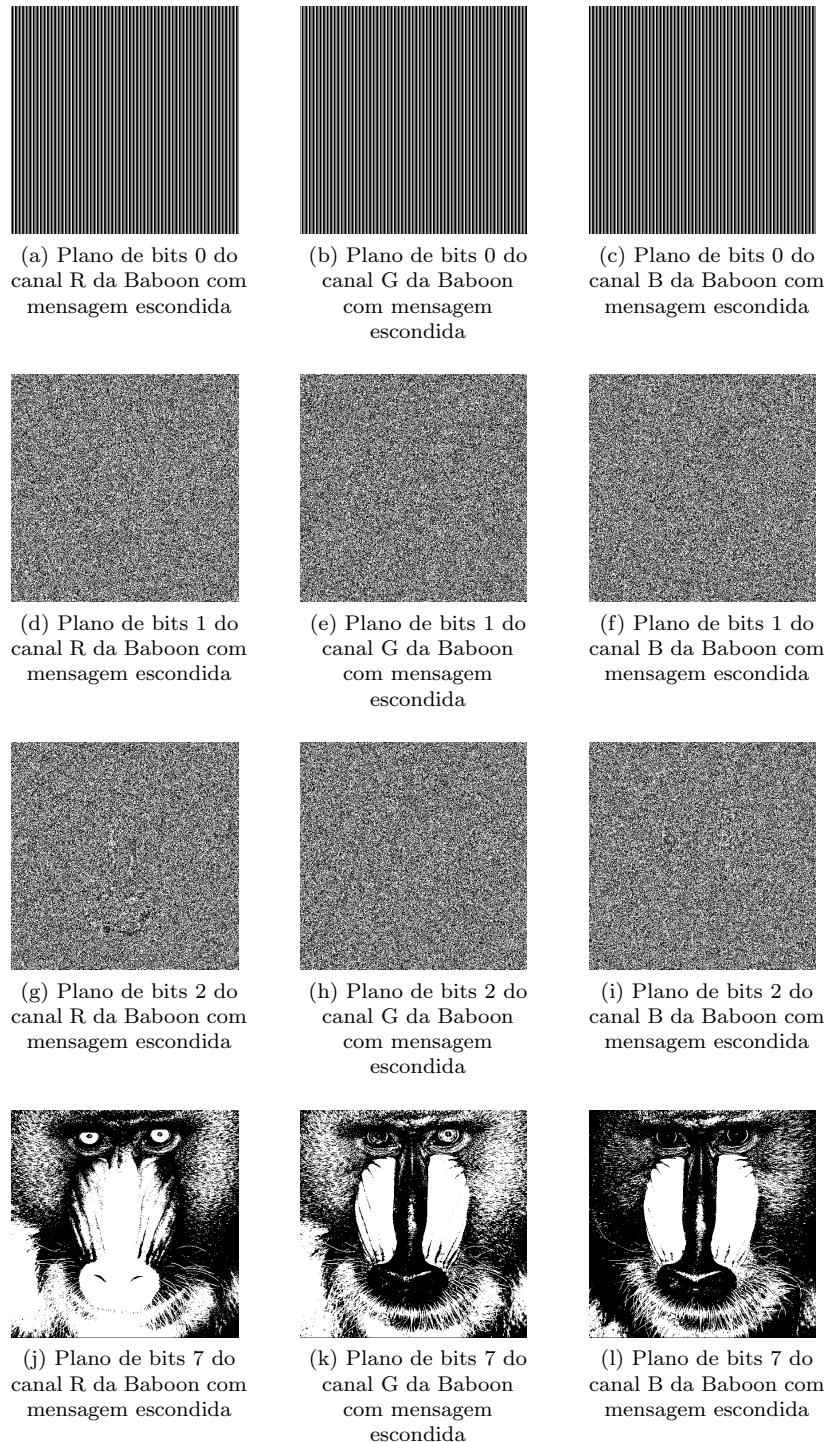


Figura 6: Planos de bits da imagem do Baboon modificada pelos bits da mensagem do arquivo `texto_longo_1.txt`.

Agora vamos verificar os planos de bits 0, 1, 2 e 7 da imagem Watch modificada. Esses planos de bits estão apresentados na Figura 7.

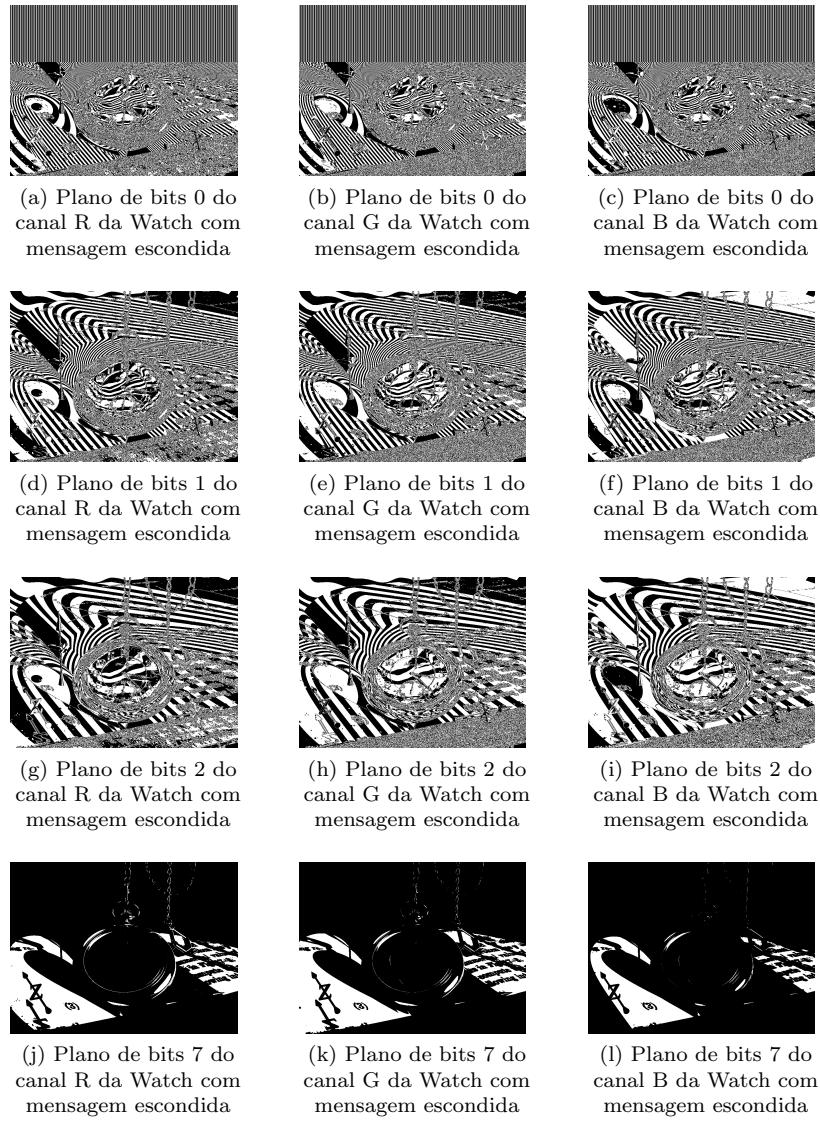


Figura 7: Planos de bits da imagem do Watch modificada pelos bits da mensagem do arquivo `texto_longo_1.txt`.

Neste caso, assim como no anterior, é possível notar uma alteração dos padrões de bits apresentados no seu plano de bits 0 (que é o plano onde a mensagem foi inserida).

3.3 Recuperação das mensagens

A recuperação das mensagens escondidas pode ser feita pela execução do programa `decodificar.py`. Para as codificações realizadas anteriormente os seguintes comandos para decodificar podem ser utilizados:

```
python3 decodificar.py -imagem_entrada=out/baboonm_plano0_texto1.png -planos_bits=0;
python3 decodificar.py -imagem_entrada=out/watchm_plano0_texto1.png -planos_bits=0;
python3 decodificar.py -imagem_entrada=out/baboonm_plano0_texto_longo_1.png -planos_bits=0;
python3 decodificar.py -imagem_entrada=out/watchm_plano0_texto_longo_1.png -planos_bits=0.
```

As mensagens recuperadas das imagens são salvas na pasta `out` que está dentro da pasta do projeto `trab1`.

4 Conclusão

Do apresentado neste trabalho, podemos concluir que a esteganografia é uma técnica interessante e promissora de ocultação de mensagens em imagens sem que estas apresentem modificações visuais ao olho humano. De qualquer maneira, mesmo que visualmente as imagens não sejam alteradas, verificamos que ainda sim é possível identificar que a imagem está modificada, assim, possivelmente com alguma mensagem escondida. Neste trabalho,

essa verificação foi feita pela análise dos planos de bits das imagens modificadas. Nesse verificação, foi possível visualizar uma alteração do padrão de bits do plano de bits 0 (plano onde foi inserida a mensagem) em ambas as imagens Baboon e Watch utilizadas. A única exceção aconteceu com a imagem Baboon e a mensagem escondida **texto1.txt** que se deu, principalmente, porque a mensagem utilizada é muito pequena.