

Fábio Takahashi Tanniguchi - RA 145980

Relatório - Trabalho 2

1. Introdução

O trabalho proposto consiste em implementar um algoritmo de esteganografia em imagens digitais e implementar um algoritmo que carregue a imagem resultante da operação anterior e crie uma nova imagem digital contendo apenas um determinado plano de bits. A separação de um plano de bits é uma operação essencial para encontrar a mensagem inserida por esteganografia.

2. Execução, entradas e saídas

Conforme explicado na seção anterior, há dois fluxos de execução:

- A. Implementação do algoritmo de esteganografia aplicado em uma imagem digital com o conteúdo de um arquivo-texto.
- B. Geração de uma nova imagem digital correspondente a apenas um determinado plano de bits da imagem passada como entrada.

Para o fluxo A, é necessário executar o arquivo *codificar.py* passando os respectivos parâmetros:

- caminho para o arquivo contendo a imagem digital (em formato PNG) que será usada;
- caminho para o arquivo-texto contendo o texto que será inserido na imagem;
- número inteiro de 0 a 7 representando o plano de bits a ser utilizado na esteganografia;
- caminho desejado para o arquivo de saída contendo a imagem (em formato PNG) após a esteganografia.

Exemplo de comando para execução do fluxo A:

python codificar.py imagem_entrada.png texto_entrada.txt plano_bits imagem_saida.png

Para o fluxo B, é necessário executar o arquivo *decodificar.py* passando os respectivos parâmetros:

- caminho para o arquivo contendo a imagem digital (em formato PNG) a ser carregada;
- número inteiro de 0 a 7 representando o plano de bits a ser isolado;
- caminho desejado para o arquivo de saída contendo a imagem (em formato PNG) representando o plano de bits requisitado.

Exemplo de comando para execução do fluxo B:

python decodificar.py imagem_saida.png plano_bits imagem_plane_bits.png

3. Descrição da solução

Para a implementação de ambos os fluxos foi utilizada a linguagem Python e as bibliotecas *sys*, *SciPy* e *BitArray*. Nas subseções seguintes serão explicadas as soluções adotadas para ambos os fluxos.

3.1. Fluxo A (Codificação)

Inicialmente, para o fluxo A, foi utilizada a biblioteca *sys* para leitura dos parâmetros passados na execução.

Em seguida, é lido o arquivo contendo a imagem digital a ser carregada, por meio do método *imread* da biblioteca *SciPy*. Além disso, é lido o arquivo-texto contendo a mensagem a ser armazenada na imagem utilizando métodos-padrão da linguagem.

Todo o texto lido do arquivo é convertido em um *bitarray*, da biblioteca de mesmo nome, ou seja, um vetor de bits. Este vetor é convertido pelo programa em um *ndarray* da biblioteca *SciPy*, de modo a possibilitar realizar uma série de operações em *n*-dimensões.

Então, o *ndarray* contendo os bits é redimensionado para as dimensões (*shape*) da imagem digital carregada inicialmente. Caso o tamanho total do *shape* da imagem seja menor que o tamanho do vetor, automaticamente são cortados do vetor redimensionado os bits em excesso. E caso o tamanho total do *shape* da imagem seja maior que o tamanho do vetor, então o vetor redimensionado será completado com zeros. Isso é facilmente provido pelo método *reshape*.

Tendo um *ndarray* de bits com as dimensões da imagem carregada, podem ser realizadas operações *bitwise* para realizar o processo de escrita dos bits no plano desejado da imagem. Primeiramente, todos os bits da matriz são deslocados à esquerda pelo número do plano de bits passado como parâmetro de execução. Feito isso, é realizada uma operação *bitwise-OR* entre cada pixel e canal da imagem e o respectivo valor no *ndarray*.

Com o texto inserido no plano de bits desejado da imagem, ela é salva no caminho passado como parâmetro de execução utilizando a função *imsave* da biblioteca *SciPy*.

3.2. Fluxo B (Decodificação)

No fluxo B, assim como no anterior, também é utilizada a biblioteca *sys* para leitura dos parâmetros de execução.

Feito isso, é realizada a leitura do arquivo contendo a imagem digital a ser carregada, da qual será extraído o plano de bits passado como parâmetro de execução.

Para a extração do plano de bits, são realizadas operações *bitwise* da linguagem. Todos os valores da imagem (para cada pixel e para cada canal) sofrem uma operação *bitwise-AND* com um byte contendo apenas um bit ativado, que é o bit do plano de bits desejado. Depois, os valores da imagem são deslocados para a direita pelo número do plano de bits.

Como a biblioteca *SciPy* e seu método *imread* não conseguiram detectar uma imagem em RGB binário e salvá-la, então os valores da imagem são multiplicados por 255.

Dessa forma, ao invés de haver valores 0 e 1, a imagem conterá valores 0 e 255 apenas, analogamente a um campo de tipo binário. Com isso, é salva uma imagem contendo apenas o plano de bits desejado.

4. Testes e resultados obtidos

Foram realizados testes com cinco imagens digitais e dois textos.

As imagens utilizadas são:

- “*baboon.png*” (vide *Figura 1*): imagem fornecida previamente de dimensões 512x512 pixels contendo o rosto de um babuíno;
- “*monalisa.png*” (vide *Figura 3*): imagem fornecida previamente de dimensões 256x256 pixels contendo uma representação dessa pintura de Leonardo da Vinci;
- “*peppers.png*” (vide *Figura 4*): imagem fornecida previamente de dimensões 512x512 pixels contendo algumas pimentas;
- “*watch.png*” (vide *Figura 5*): imagem fornecida previamente de dimensões 1024x768 pixels contendo um desenho de um relógio sobre um livro;
- “*DSC05354.png*” (vide *Figura 2*): imagem de autoria do próprio redator deste texto, de dimensões 4608x3456 pixels, contendo uma perspectiva de um trecho da Rodovia dos Bandeirantes, Estado de São Paulo, Brasil.

Os textos utilizados são:

- texto completo do livro “The Adventures of Sherlock Holmes”, de Arthur Conan Doyle, oferecido pelo The Project Gutemberg EBook, baixado de <https://norvig.com/big.txt>;
- texto apenas do primeiro capítulo da obra acima.

Como o texto completo possui dimensões impertinentes a este relatório (possui 128.458 linhas e 6.488.666 caracteres), ele não será reproduzido aqui.

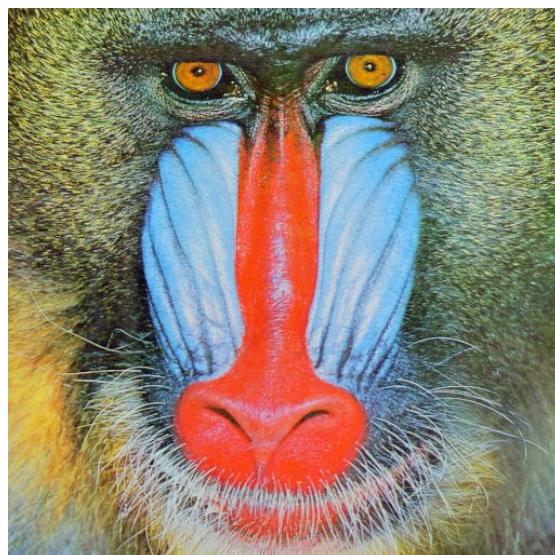


Figura 1 - Imagem “baboon.png”

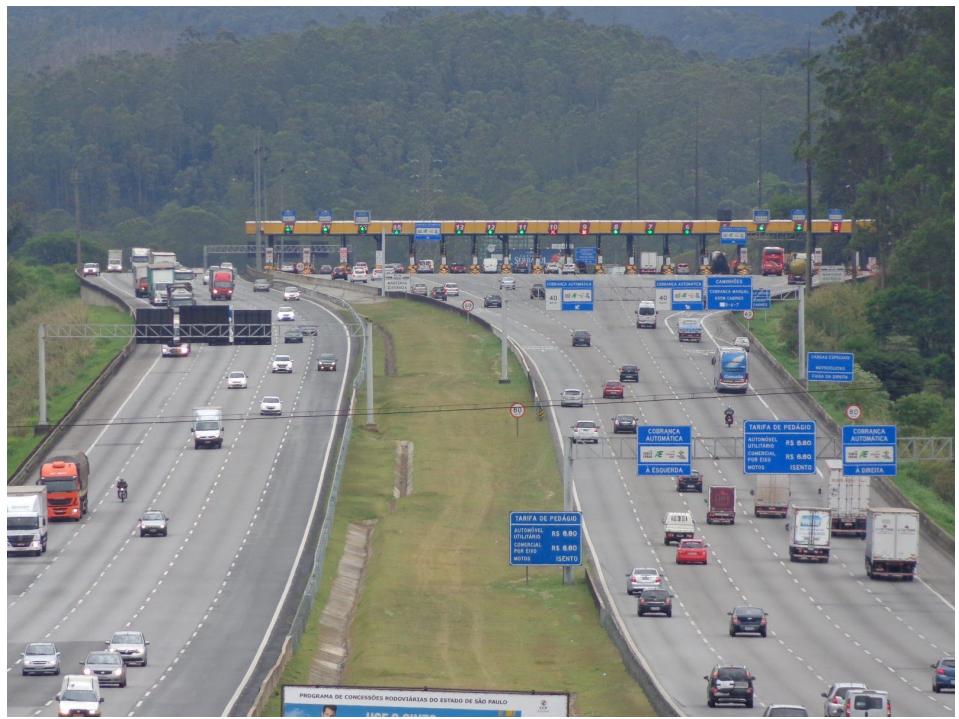


Figura 2 - Imagem “DSC05354.png”



Figura 3 - Imagem “monalisa.png”



Figura 4 - Imagem “peppers.png”



Figura 5 - Imagem “watch.png”

Não houve maiores problemas nos testes utilizando as imagens fornecidas previamente (*Figuras 1, 3, 4 e 5*). Dessa forma, este relatório seguirá descrevendo apenas os resultados das *Figuras 2* (inserida posteriormente) e *5* (a de maiores dimensões dentre as fornecidas previamente).

4.1. Resultados utilizando imagem “watch.png” (Figura 5)



Figura 6 - Imagem “watch.png” após inserção de parte do texto completo no plano de bits 1 (devido ao fato do texto completo convertido em binário ser maior que a imagem, parte do conteúdo foi ignorado)



Figura 7 - Imagem “watch.png” após inserção do primeiro capítulo do texto no plano de bits 1

Pelas Figuras 6 e 7 é possível perceber que a inserção da mensagem no plano de bits 1 (2^{o} bit menos significativo), pelo menos a olho nu tende a passar de forma despercebida por alguém que veja a figura.

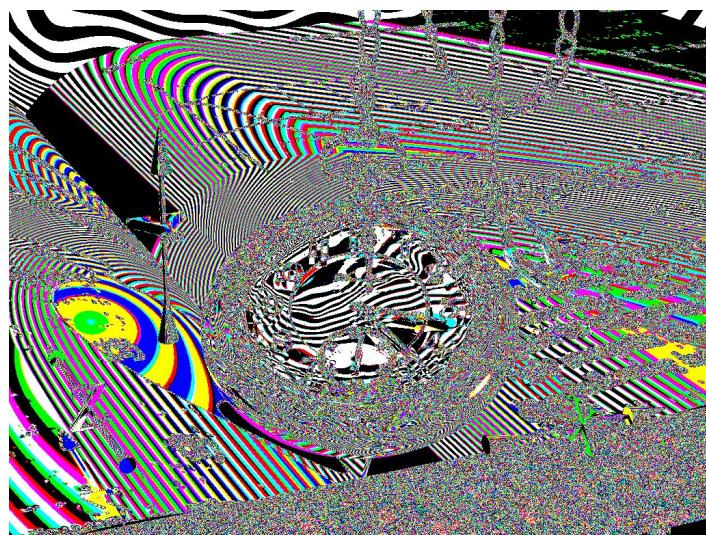


Figura 8 - Plano de bits 0 da Figura 6

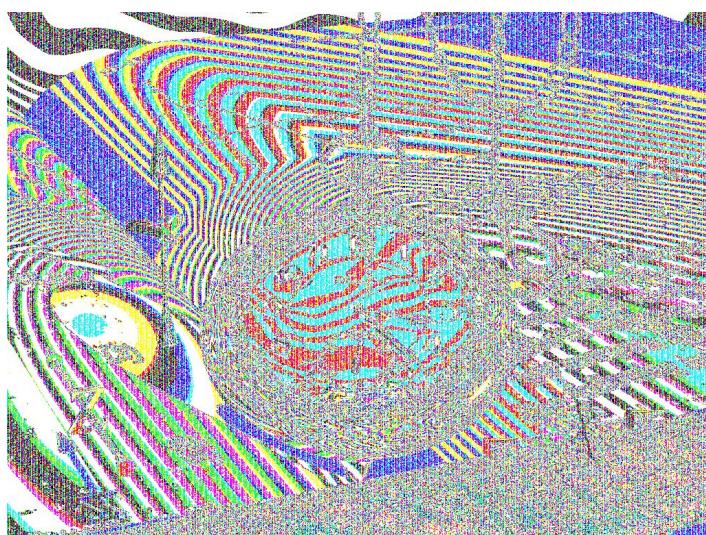


Figura 9 - Plano de bits 1 da Figura 6

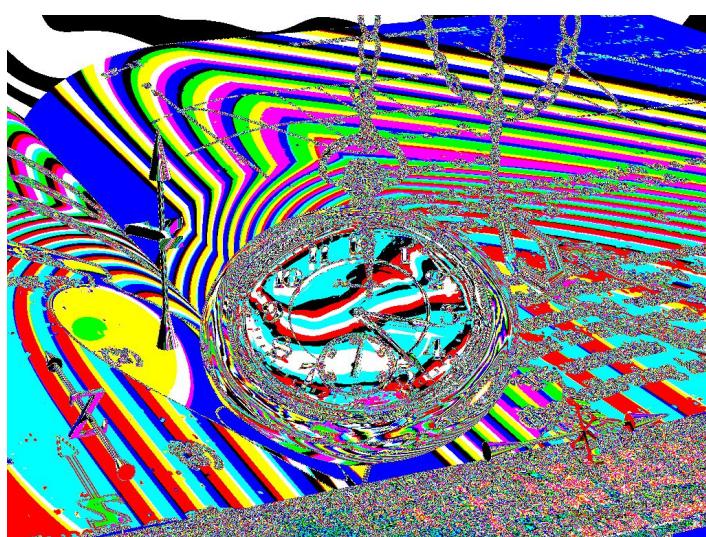


Figura 10 - Plano de bits 2 da Figura 6



Figura 11 - Plano de bits 7 da Figura 6

Pelas *Figuras 8 a 11*, ao extrair os planos de bits é possível notar:

- que o plano de bits 1, onde foi inserida a mensagem, em comparação com os “vizinhos” 0 e 2, aparenta ter um ruído;
- e que o plano de bits 7 é uma maneira de representar a imagem original em RGB-binário, pois leva em conta apenas o bit mais significativo.

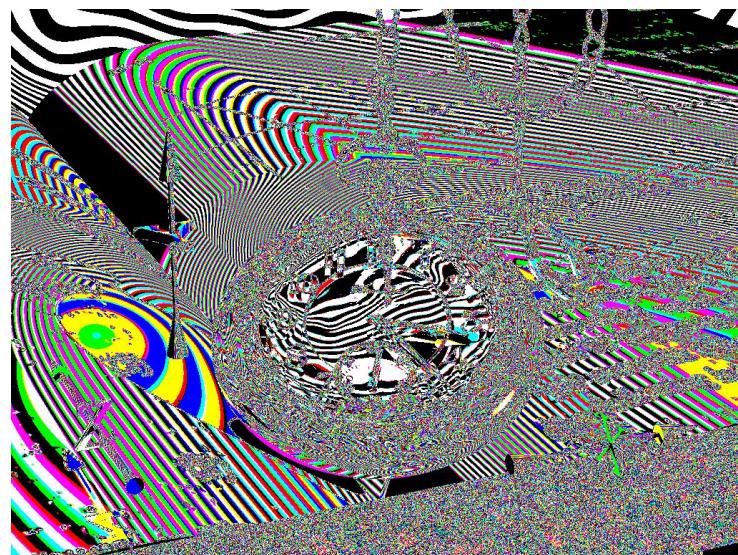


Figura 12 - Plano de bits 0 da Figura 7

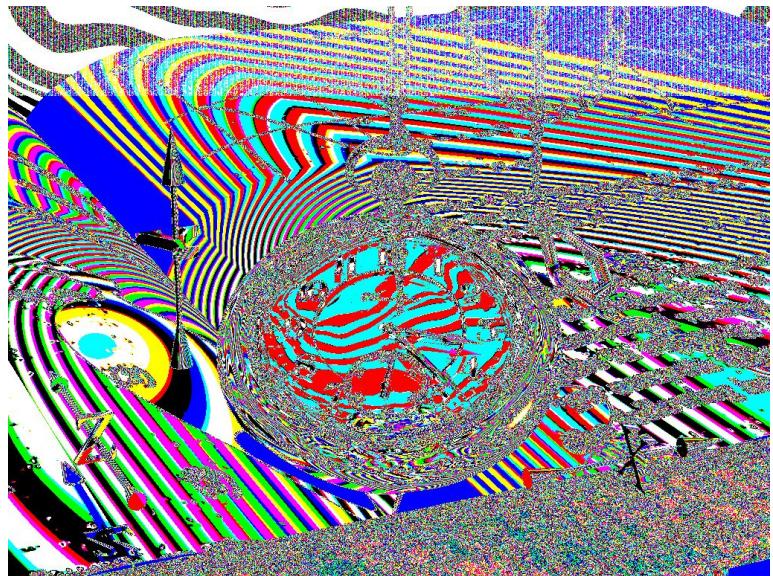


Figura 13 - Plano de bits 1 da Figura 7

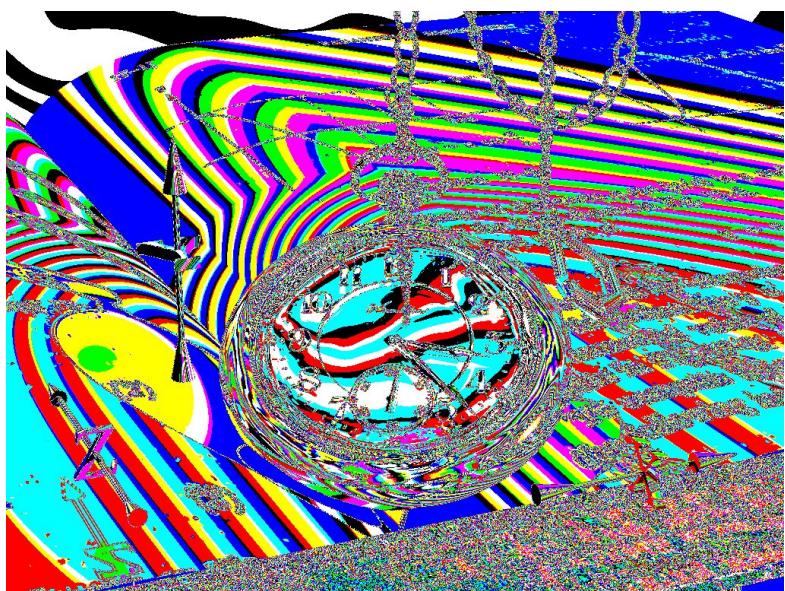


Figura 14 - Plano de bits 2 da Figura 7



Figura 15 - Plano de bits 7 da Figura 7

Pode-se perceber pelas *Figuras 12 a 15* que:

- em comparação com a *Figura 9*, a *Figura 13* possui o ruído detectado apenas na parte superior, levando à suposição de que o aparente ruído no plano de bits 1 é consequência da mensagem inserida neste plano; lembrando que a mensagem inserida na *Figura 7* deve ocupar apenas parte da imagem enquanto a mensagem inserida na *Figura 6* deve ocupá-la inteiramente;
- como esperado, os demais planos da *Figura 7* não possuem diferença com os respectivos planos da *Figura 6*, uma vez que não devem ser afetados pela esteganografia.

4.2. Resultados utilizando imagem “DSC05354.png” (Figura 2)

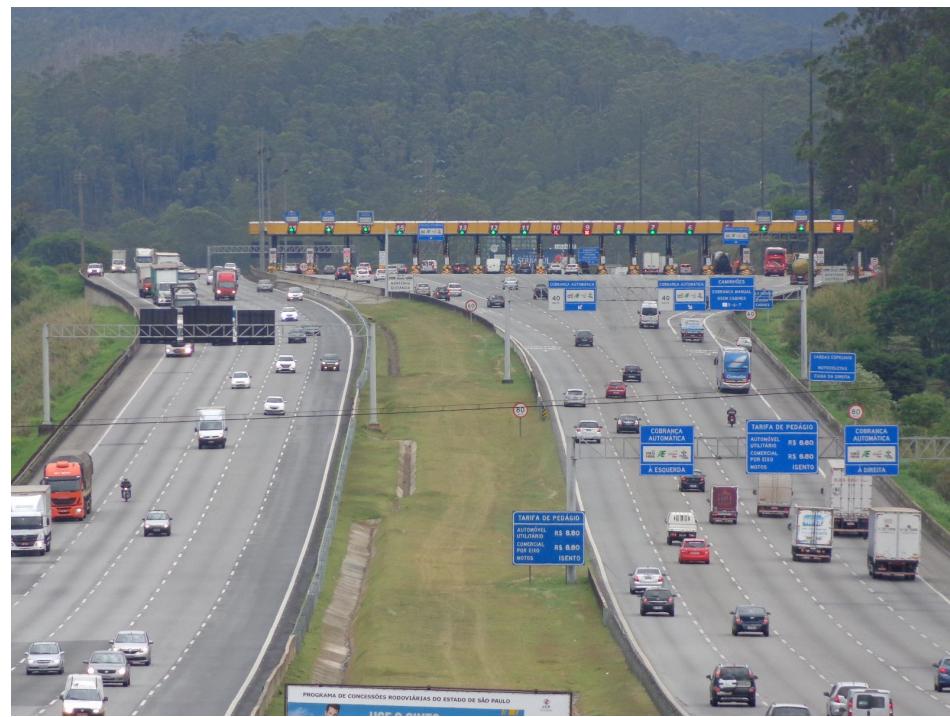


Figura 16 - Imagem “DSC05354.png” após inserção de parte do texto completo no plano de bits 1 (devido ao fato do texto completo convertido em binário ser maior que a imagem, parte do conteúdo foi ignorado)



Figura 17 - Imagem “DSC05354.png” após inserção do primeiro capítulo do texto no plano de bits 1

Pelas *Figuras 16 e 17* é possível perceber novamente que a inserção da mensagem no plano de bits 1 (2^{o} bit menos significativo), pelo menos a olho nu tende a passar de forma despercebida por alguém que veja a figura.



Figura 18 - Plano de bits 0 da Figura 16

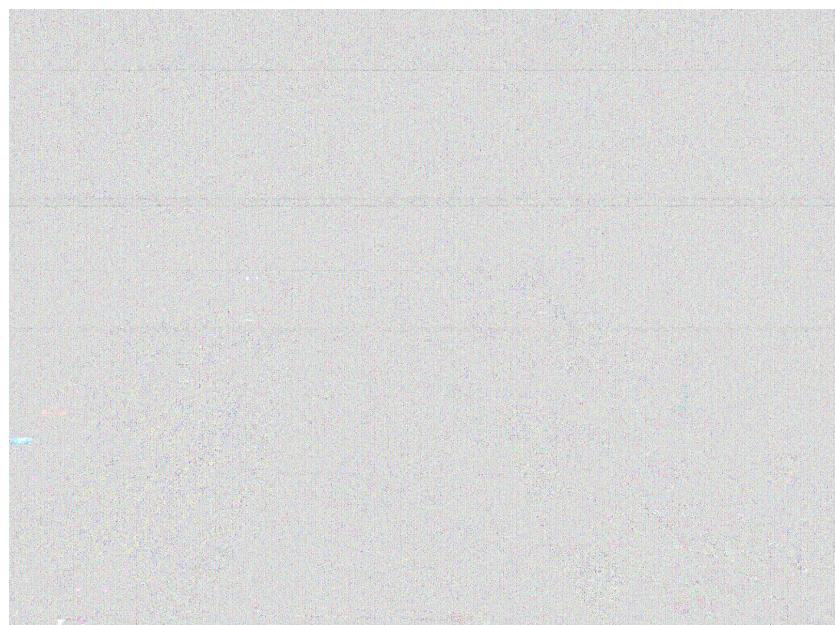


Figura 19 - Plano de bits 1 da Figura 16

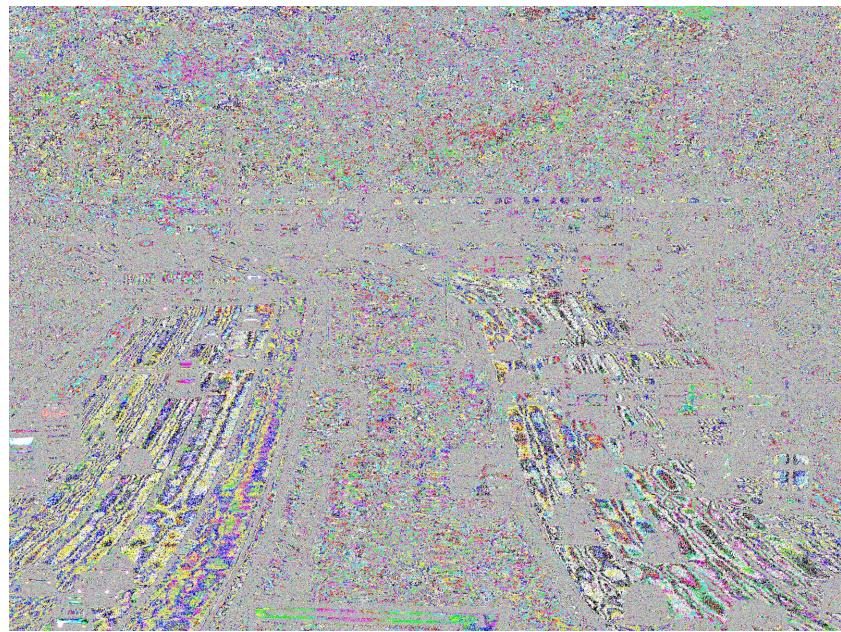


Figura 20 - Plano de bits 2 da Figura 16



Figura 21 - Plano de bits 7 da Figura 16

Pelas *Figuras 18 a 21*, ao extrair os planos de bits é possível notar que:

- o plano de bits 1, onde foi inserida a mensagem, em comparação com os “vizinhos” 0 e 2, aparenta ter um ruído que a deixa esbranquiçada;
- o plano de bits 7 realmente é uma maneira de representar a imagem original em RGB-binário, pois leva em conta apenas o bit mais significativo.



Figura 22 - Plano de bits 0 da Figura 17

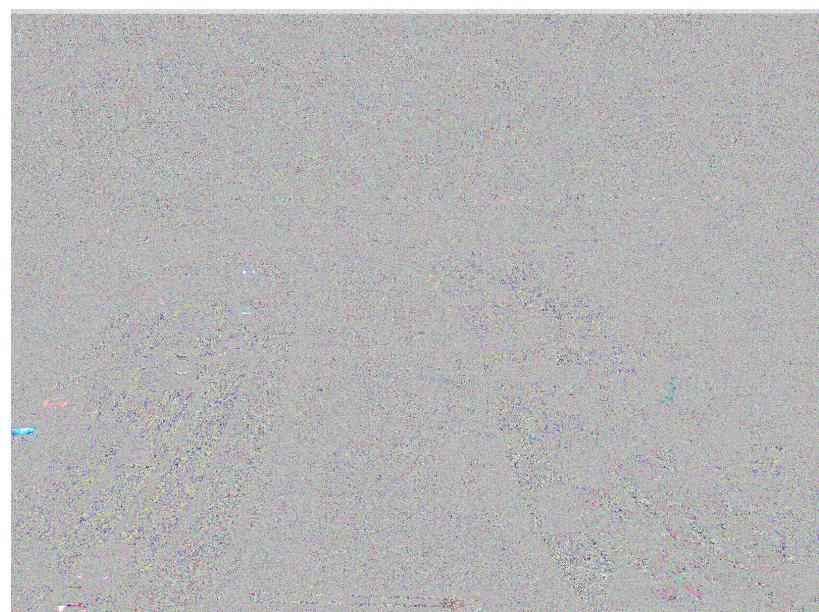


Figura 23 - Plano de bits 1 da Figura 17

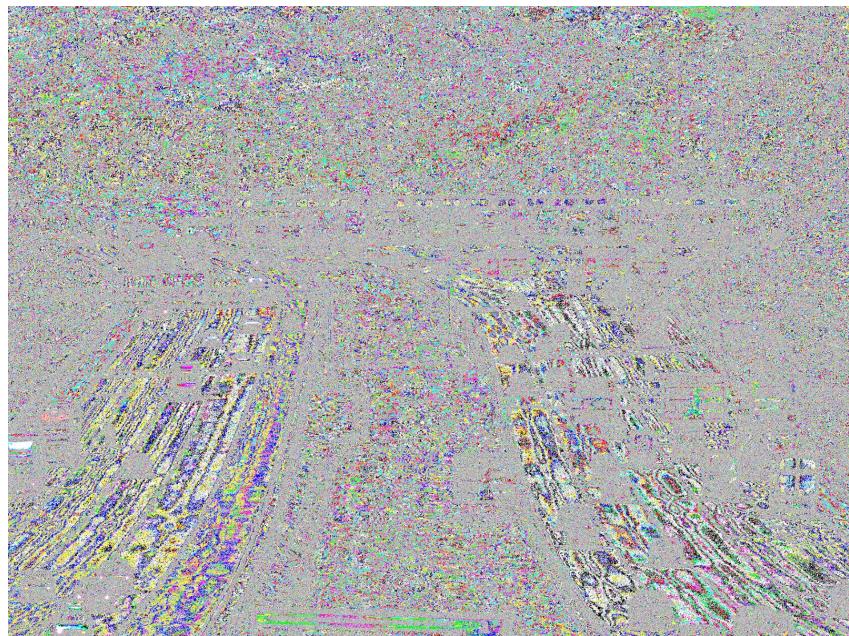


Figura 24 - Plano de bits 2 da Figura 17



Figura 25 - Plano de bits 7 da Figura 17

Pode-se perceber pelas *Figuras 22 a 25* que:

- em comparação com a *Figura 19*, a *Figura 23* possui o ruído detectado apenas na parte superior, levando à suposição de que o aparente ruído no plano de bits 1 é consequência da mensagem inserida neste plano; lembrando que a mensagem inserida na *Figura 7* deve ocupar apenas parte da imagem enquanto a mensagem inserida na *Figura 6* deve ocupá-la inteiramente;
- como esperado, os demais planos da *Figura 17* não possuem diferença com os respectivos planos da *Figura 16*, uma vez que não devem ser afetados pela esteganografia.

Devido ao fato do texto em binário correspondente ao primeiro capítulo do livro sobre *Sherlock Holmes* ser muito menor do que as dimensões da imagem, a primeira observação do parágrafo acima pode não ser perceptível a olho nu dependendo das condições de visualização deste relatório. O ruído esbranquiçado aparece, na *Figura 23*, apenas como uma faixa superior de altura pequena.

Na primeira tentativa de execução de testes com esta imagem digital e o texto do livro sobre *Sherlock Holmes*, o processo de esteganografia (codificação) mostrou-se significativamente lento, levando a analisar todo o algoritmo em busca de melhorias.

Foi identificado que, ao converter os caracteres do texto lido em codificação ASCII, ocorria uma iteração explícita pelos caracteres do texto. Ao invés de realizar manualmente a conversão, procurou-se uma forma mais otimizada de conversão, até que foi encontrada a biblioteca *BitArray* que já possuía inclusive a conversão completa da cadeia de caracteres em uma lista de bits. Com o uso dessa biblioteca, houve melhora significativa no desempenho do algoritmo.

5. Conclusão

O processo de esteganografia em planos de bits menos significativos de imagens digitais mostrou-se bastante interessante, pois as imagens resultantes após a codificação dificilmente seriam notadas como modificadas por um usuário qualquer a olho nu. Além disso, os processos de codificação e de decodificação apresentaram desempenho bastante razoável mesmo em uma imagem de dimensões maiores (vide imagem utilizada na subseção 4.2).

A realização de testes incluindo uma imagem de dimensões bem maiores do que as inicialmente fornecidas e de um texto consideravelmente longo foi importante para notar um gargalo no desempenho do algoritmo implementado inicialmente. Isso levou a encontrar e utilizar uma biblioteca específica para a finalidade desejada (conversão de cadeia de caracteres em lista de bits), melhorando o desempenho da solução.