

**Relatório do Primeiro Trabalho Prático**

|  |  |
| --- | --- |
| Grupo | 13 |
| 38866 | Manuel Dias |

|  |  |
| --- | --- |
| Docente | Artur Ferreira |
|  |  |

Relatório final realizado no âmbito da Unidade Curricular de Comunicações,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2019/2020

Maio de 2020

Índice

[Índice de Figuras 2](#_Toc39509368)

[1. Objetivos 4](#_Toc39509369)

[2. Grupo 1 5](#_Toc39509370)

[2.1 Alínea A 5](#_Toc39509371)

[2.2 Alínea B 5](#_Toc39509372)

[2.2.1 Alínea (i) 5](#_Toc39509373)

[2.2.2 Alínea (ii) 6](#_Toc39509374)

[2.2.3 Alínea (iii) 8](#_Toc39509375)

[2.3 Alínea C 8](#_Toc39509376)

[2.4 Alínea D 9](#_Toc39509377)

[3. Grupo 2 11](#_Toc39509378)

[3.1 Alínea A 11](#_Toc39509379)

[3.2 Alínea B 12](#_Toc39509380)

[4. Grupo 3 16](#_Toc39509381)

[4.1 Alínea A 16](#_Toc39509382)

[4.1.1 Função *parity\_check\_encode* 16](#_Toc39509383)

[4.1.2 Função *parity\_check\_decode* 17](#_Toc39509384)

[4.2 Alínea B 17](#_Toc39509385)

[4.2.1 Função *hamming\_encode* 17](#_Toc39509386)

[4.2.2 Função *hamming\_decode\_correct* 18](#_Toc39509387)

[4.3 Alínea C 18](#_Toc39509388)

[5. Grupo 4 21](#_Toc39509389)

[5.1 Alínea (i) 22](#_Toc39509390)

[5.2 Alínea (ii) 22](#_Toc39509391)

[5.3 Alínea (iii) 25](#_Toc39509392)

[5.4 Alínea (iv) 26](#_Toc39509393)

[5.5 Alínea (v) 26](#_Toc39509394)

[5.6 Alínea (vi) 27](#_Toc39509395)

# Índice de Figuras

[Figura 1 - Histograma do ficheiro "a.txt" 6](#_Toc39509396)

[Figura 2 - Histograma do ficheiro "alice29.txt" 6](#_Toc39509397)

[Figura 3 - Histograma do ficheiro "arj241.exe" 6](#_Toc39509398)

[Figura 4 - Histograma do ficheiro "bird.gif" 6](#_Toc39509399)

[Figura 5 - Histograma do ficheiro "cp.htm" 6](#_Toc39509400)

[Figura 6 - Histograma do ficheiro "lena.bmp" 6](#_Toc39509401)

[Figura 7 - Histograma do ficheiro "lena.jpg" 7](#_Toc39509402)

[Figura 8 - Histograma do ficheiro "lena.zip" 7](#_Toc39509403)

[Figura 9 - Histograma do ficheiro "Person.java" 7](#_Toc39509404)

[Figura 10 - Histograma do ficheiro "progc.c" 7](#_Toc39509405)

[Figura 11 - Histograma do ficheiro "watermark.cs" 7](#_Toc39509406)

[Figura 12 - Histograma com M=100 de uma FMP igual para todos os símbolos 9](#_Toc39509407)

[Figura 13 - Histograma com M=100 de uma FMP diferente para todos os símbolos 9](#_Toc39509408)

[Figura 14 - Histograma com M=1000 de uma FMP igual para todos os símbolos 9](#_Toc39509409)

[Figura 15 - Histograma com M=1000 de uma FMP diferente para todos os símbolos 9](#_Toc39509410)

[Figura 16 - Histograma com M=10000 de uma FMP igual para todos os símbolos 9](#_Toc39509411)

[Figura 17 - Histograma com M=10000 de uma FMP diferente para todos os símbolos 9](#_Toc39509412)

[Figura 18 - Ficheiro original "a.txt" 12](#_Toc39509413)

[Figura 19 - Criptografia de "a.txt" guardado no ficheiro "OTPC\_a.txt" 12](#_Toc39509414)

[Figura 20 - Descodificação de "OTPC\_a.txt" guardado em"OTPD\_OTPC\_a.txt" 12](#_Toc39509415)

[Figura 21 - Ficheiro original "teste.txt" 12](#_Toc39509416)

[Figura 22 - Criptografia de "teste.txt" guardado em "OTPC\_teste.txt" 12](#_Toc39509417)

[Figura 23 - Descodificação de "OTPC\_teste.txt" guardado em "OTPD\_OTPC\_teste.txt" 13](#_Toc39509418)

[Figura 24 - Ficheiro original "lena.bmp" 13](#_Toc39509419)

[Figura 25 - Criptografia de "lena.bmp" guardado em "OTPC\_lena.bmp" 13](#_Toc39509420)

[Figura 26 - Descodificação de "OTPC\_lena.bmp" guardado em "OTPD\_OTPC\_lena.bmp" 13](#_Toc39509421)

[Figura 27 - Ficheiro original "bird.gif" 14](#_Toc39509422)

[Figura 28 - Criptografia de "bird.gif" guardado em "OTPC\_bird.gif" 14](#_Toc39509423)

[Figura 29 - Descodificação de "OTPC\_bird.gif" guardado em "OTPD\_OTPC\_bird.gif" 14](#_Toc39509424)

[Figura 30 - Resultado dos testes aos quatros ficheiros 15](#_Toc39509425)

[Figura 31 - Teste às funções de parity com ausência de erros 18](#_Toc39509426)

[Figura 32 - Teste às funções de Hamming com ausência de erros 18](#_Toc39509427)

[Figura 33 - Teste às funções de parity com 1 bit em erro 19](#_Toc39509428)

[Figura 34 - Teste às funções de hamming com 1 bit em erro 19](#_Toc39509429)

[Figura 35 - Teste às funções de parity com múltiplos bits em erro 19](#_Toc39509430)

[Figura 36 - Teste às funções de hamming com múltiplos bits em erro, com apenas 1 bit em erro num bloco 19](#_Toc39509431)

[Figura 37 - Teste às funções de hamming com múltiplos bits em erro, com múltiplos bits em erro num bloco 19](#_Toc39509432)

[Figura 38 - Ficheiro original 20](#_Toc39509433)

[Figura 39 - Ficheiro descodificado 20](#_Toc39509434)

[Figura 40 - Representação do modelo de Shannon 21](#_Toc39509435)

[Figura 41 - Representação da GUI 21](#_Toc39509436)

[Figura 42 - Resultado obtido após carregar o ficheiro "lena.zip" 22](#_Toc39509437)

[Figura 43 - Estado da GUI após o uso das funções de huffman ao ficheiro "lena.bmp" 23](#_Toc39509438)

[Figura 44 - Ficheiro original "lena.bmp" 24](#_Toc39509439)

[Figura 45 - Ficheiro resultante "HuffD\_T\_HuffC\_lena.bmp" 24](#_Toc39509440)

[Figura 46 - Estado da GUI após o uso das funções de zip ao ficheiro "lena.bmp" 24](#_Toc39509441)

[Figura 47 - Ficheiro original "lena.bmp" 24](#_Toc39509442)

[Figura 48 - Ficheiro resultante " lena.bmp" 24](#_Toc39509443)

[Figura 49 - Resultados da cifra ao ficheiro "a.txt" sem codificação de fonte 25](#_Toc39509444)

[Figura 50 - Resultados da cifra ao ficheiro "a.txt" com codificação de fonte 25](#_Toc39509445)

[Figura 51 - Ficheiro original "a.txt" 26](#_Toc39509446)

[Figura 52 - Resultado do ficheiro "a.txt" após codificação de fonte e cifra 26](#_Toc39509447)

[Figura 53 - Estado da GUI após a codificação de canal de bit de paridade par (8, 7) 26](#_Toc39509448)

[Figura 54 - Estado da GUI após a codificação de canal de Hamming (7, 4) 27](#_Toc39509449)

# 1. Objetivos

Este trabalho tem como objetivos:

* Desenvolver programas e aplicações em MATLAB/OCTAVE;
* Estudo e aplicação de conceitos sobre Sistemas de Comunicação Digitais (SCD), teoria da informação, codificação de fonte, cifra e canal;

# 2. Grupo 1

No primeiro grupo do trabalho prático, é pretendido aplicar alguns conceitos de teoria de informação e codificação de fonte.

## 2.1 Alínea A

Para desenvolver a função *entropy\_histogram*, foi decidido que esta tem três parâmetros de entrada, o nome do ficheiro sem extensão, a extensão do ficheiro a ser analisado e o valor de M e um de saída, o valor da entropia.

No processo do planeamento da função, foi verificado que esta é composta por quatro partes:

1. Leitura do ficheiro e obtenção dos seus símbolos;
2. Calcular a entropia;
3. Visualização do histograma;
4. Guardar o histograma num ficheiro de imagem;

Na primeira parte, foi recorrido à função *file2vector*, dada nas aulas práticas, para obter o conteúdo do ficheiro num vetor. Depois foi usada a função *unique* do MATLAB para obter os símbolos usados no ficheiro.

Na segunda parte é usada a função *histc* que recebe um vetor de sinais e um vetor de todos os símbolos que o vetor de sinais tem e retorna o número de ocorrências de cada símbolo no vetor de sinal. A partir destes números é possível verificar a probabilidade de ocorrência de cada sinal e por consequente calcular o valor da entropia.

Na terceira parte foi necessário criar uma figura e guardá-la em memória para depois na quarta fase guardar a figura gerada num ficheiro de imagem. Esta figura tem como nome, o nome do ficheiro. É usada a função *histograma* do MATLAB para visualizar o histograma na figura.

Na quarta parte é feito uma chamada à função *saveas* do MATLAB para guardar a figura resultante num ficheiro de imagem *.png*.

## 2.2 Alínea B

### 2.2.1 Alínea (i)

Após analisar a extensão dos ficheiros da parta *COM-TestFiles*, foi identificado os seguintes ficheiros com formatos com compressão: “arj241a.exe”, “bird.gif”, “lena.bmp”, “lena.jpg” e “lena.zip”.

### 2.2.2 Alínea (ii)

|  |  |
| --- | --- |
| Figura - Histograma do ficheiro "a.txt" | Figura - Histograma do ficheiro "alice29.txt" |
| Figura - Histograma do ficheiro "arj241.exe" | Figura - Histograma do ficheiro "bird.gif" |
| Figura - Histograma do ficheiro "cp.htm" | Figura - Histograma do ficheiro "lena.bmp" |

|  |  |
| --- | --- |
| Figura 7 - Histograma do ficheiro "lena.jpg" | Figura 8 - Histograma do ficheiro "lena.zip" |
| Figura 9 - Histograma do ficheiro "Person.java" | Figura 10 - Histograma do ficheiro "progc.c" |
| Figura 11 - Histograma do ficheiro "watermark.cs" |  |

### 2.2.3 Alínea (iii)

A partir dos histogramas e do cálculo da entropia de cada ficheiro, é possível fazer uma relação entre eles.

Primeiro que tudo é preciso entender o que é a entropia. A entropia é o valor esperado da informação de cada concretização, ou seja, é uma correlação entre a probabilidade de cada símbolo com o número de ocorrências deste num acontecimento. Um acontecimento pode ser dado como um ficheiro ou um vetor de símbolos.

A partir disto é possível dizer que é espectável que a entropia tenha um valor máximo para cada concretização e assim comparar com os valores de entropia obtidos. No caso que está a ser estudado, um símbolo corresponde a um caracter e este pode ser no máximo 1 Byte, ou 8 bits. A partir disto, é possível dizer que perante a situação de máxima entropia temos que . Um ficheiro com uma entropia máxima, demonstra que todos os símbolos possam ser compactados.

Dos ficheiros indicados na alínea 1.B)(i), é possível verificar que todos estes têm um valor muito aproximado da máxima entropia. No entanto, há uns ficheiros com uma maior entropia do que outro, pois este valor não depende apenas do formato da compressão, mas da informação em si do ficheiro.

Ao relacionar estes valores com a entropia, é possível analisar que os histogramas que estão associados a uma entropia mais aproximada do valor da máxima entropia, são os histogramas cujo número de ocorrências de cada símbolo é igualmente distribuído. Isto pode ser verificado comparando o histograma do ficheiro "arj241.exe" com o "lena.bmp". Este último ficheiro tem caracteres que ou não tem ocorrências ou quase nenhumas, enquanto que o primeiro, quase todos os símbolos tem aproximadamente o mesmo número de ocorrências.

## 2.3 Alínea C

Para implementar a função *symbol\_source*, foi decidido que esta recebe quatro parâmetros de entrada, um vetor de símbolos, um vetor da probabilidade associada a cada símbolo, o valor total de símbolos dentro de um ficheiro e o nome do ficheiro a gerar.

Esta está dividida em três partes:

1. Verificar os conteúdos dos parâmetros de entrada;
2. Calcular o número de ocorrências de cada símbolo;
3. Escrever no ficheiro os símbolos consoante o número de ocorrências calculado;

Na primeira parte é verificado se todos os símbolos têm uma probabilidade associada e se a soma de todas as probabilidades é igual a 1.

Na segunda parte, a partir da probabilidade de cada símbolo e do valor total de símbolos do ficheiro, é calculado o número de ocorrências do mesmo e escrito no ficheiro.

## 2.4 Alínea D

|  |  |
| --- | --- |
| Figura - Histograma com M=100 de uma FMP igual para todos os símbolos | Figura - Histograma com M=100 de uma FMP diferente para todos os símbolos |
| Figura - Histograma com M=1000 de uma FMP igual para todos os símbolos | Figura - Histograma com M=1000 de uma FMP diferente para todos os símbolos |
| Figura - Histograma com M=10000 de uma FMP igual para todos os símbolos | Figura - Histograma com M=10000 de uma FMP diferente para todos os símbolos |

A partir dos resultados obtidos é possível concluir que o número de ocorrências total de um acontecimento não modifica o valor da entropia nem do histograma, mas sim o número médio de ocorrências de cada símbolo. Isto é verificado a partir do valor da entropia de cada FMP que se mantém constante para qualquer M.

# 3. Grupo 2

No segundo grupo do trabalho prático é pretendido explorar o conceito de cifra através da aplicação do sistema One Time Pad de Vernam.

## 3.1 Alínea A

Para aplicar qualquer tipo de sistema de cifra, tanto o algoritmo de codificação como o algoritmo de descodificação têm o mesmo número de passos, no entanto podem diferenciar na realização de algumas operações. Por isso a implementação das funções *one\_time\_pad\_cipher* e *one\_time\_pad\_decipher* é semelhante.

Para a implementação das funções, estas recebem dois parâmetros de entrada, o nome do ficheiro e a chave, e um parâmetro de saída, o nome do ficheiro resultante da codificação/descodificação.

As funções estão divididas em três partes:

1. Leitura do ficheiro e verificação de parâmetros;
2. Operação de codificação ou descodificação;
3. Escrita da codificação ou descodificação num ficheiro;

Na primeira parte é lido o ficheiro passado como parâmetro de entrada e é verificado se este e a chave contêm o mesmo número de caracteres, pois esta é uma necessidade obrigatória segundo a cifra de Vernam.

Na segunda parte, para a codificação é somado o caracter corrente da mensagem com o da chave e é feito o resto da divisão de 256, pois o valor resultante não pode ultrapassar o tamanho de um caracter, que é 255. Para a descodificação é feito quase a mesma coisa só que em vez da realização soma, realiza-se a subtração.

Na terceira parte, no caso da codificação, é feito uma concatenação do nome do ficheiro com o prefixo “OTPC\_” e é criado um ficheiro com o nome do resultado desta concatenação. No caso da descodificação o prefixo da concatenação é “OTPD\_”.

## 3.2 Alínea B

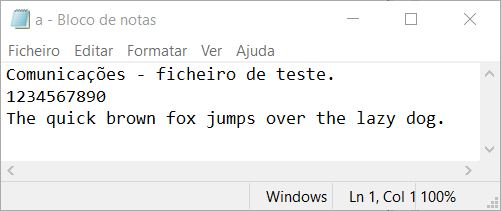


Figura 18 - Ficheiro original "a.txt"

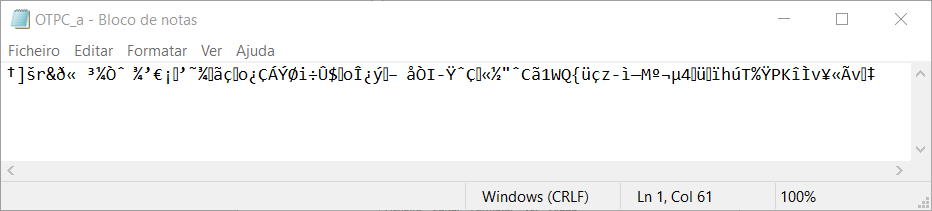


Figura 19 - Criptografia de "a.txt" guardado no ficheiro "OTPC\_a.txt"

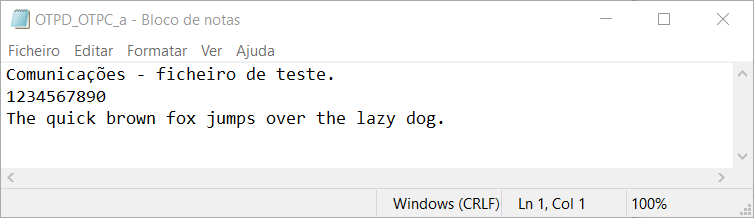


Figura 20 - Descodificação de "OTPC\_a.txt" guardado em"OTPD\_OTPC\_a.txt"

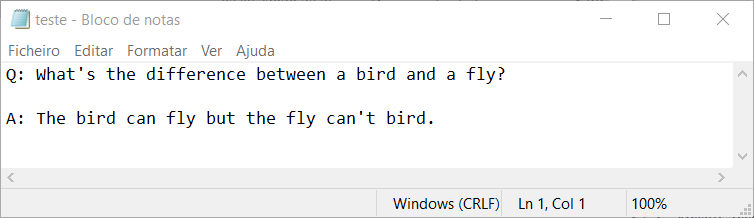


Figura 21 - Ficheiro original "teste.txt"

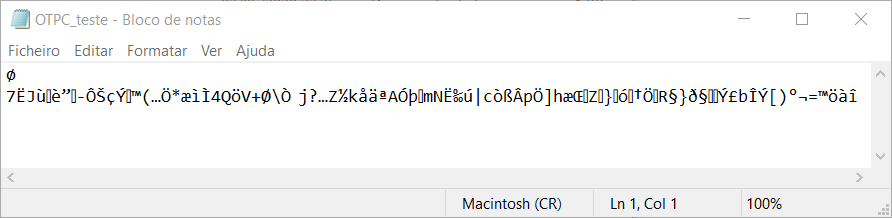


Figura 22 - Criptografia de "teste.txt" guardado em "OTPC\_teste.txt"

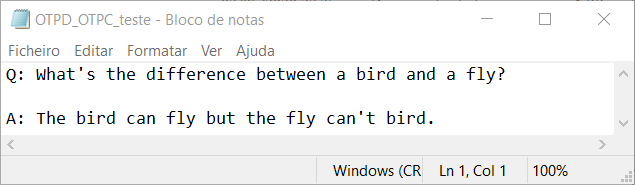


Figura 23 - Descodificação de "OTPC\_teste.txt" guardado em "OTPD\_OTPC\_teste.txt"



Figura 24 - Ficheiro original "lena.bmp"

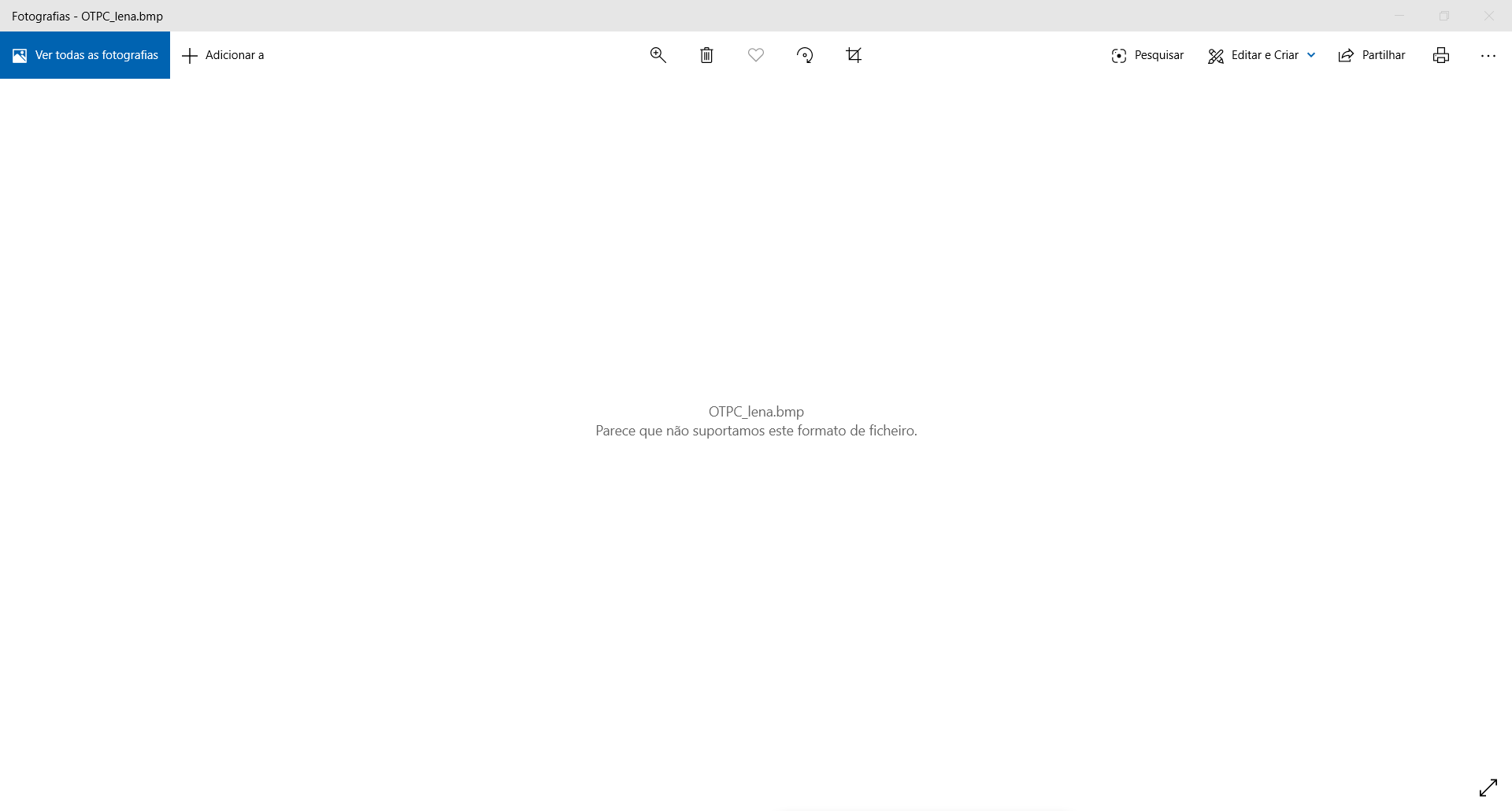


Figura - Criptografia de "lena.bmp" guardado em "OTPC\_lena.bmp"

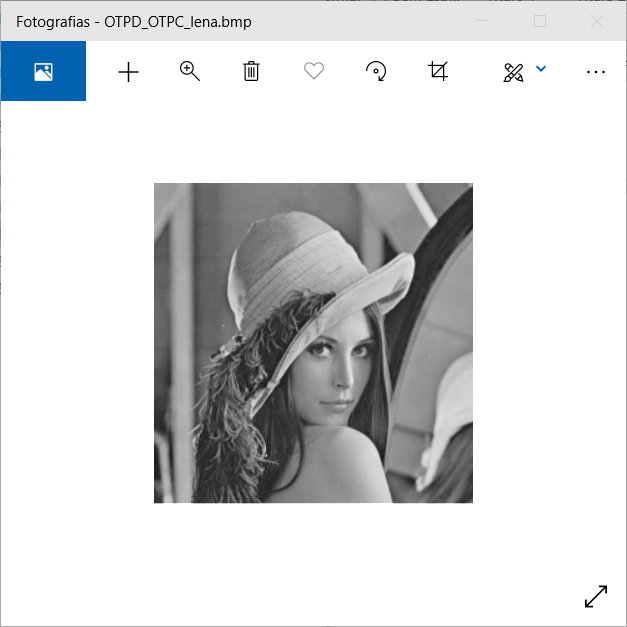


Figura - Descodificação de "OTPC\_lena.bmp" guardado em "OTPD\_OTPC\_lena.bmp"



Figura - Ficheiro original "bird.gif"

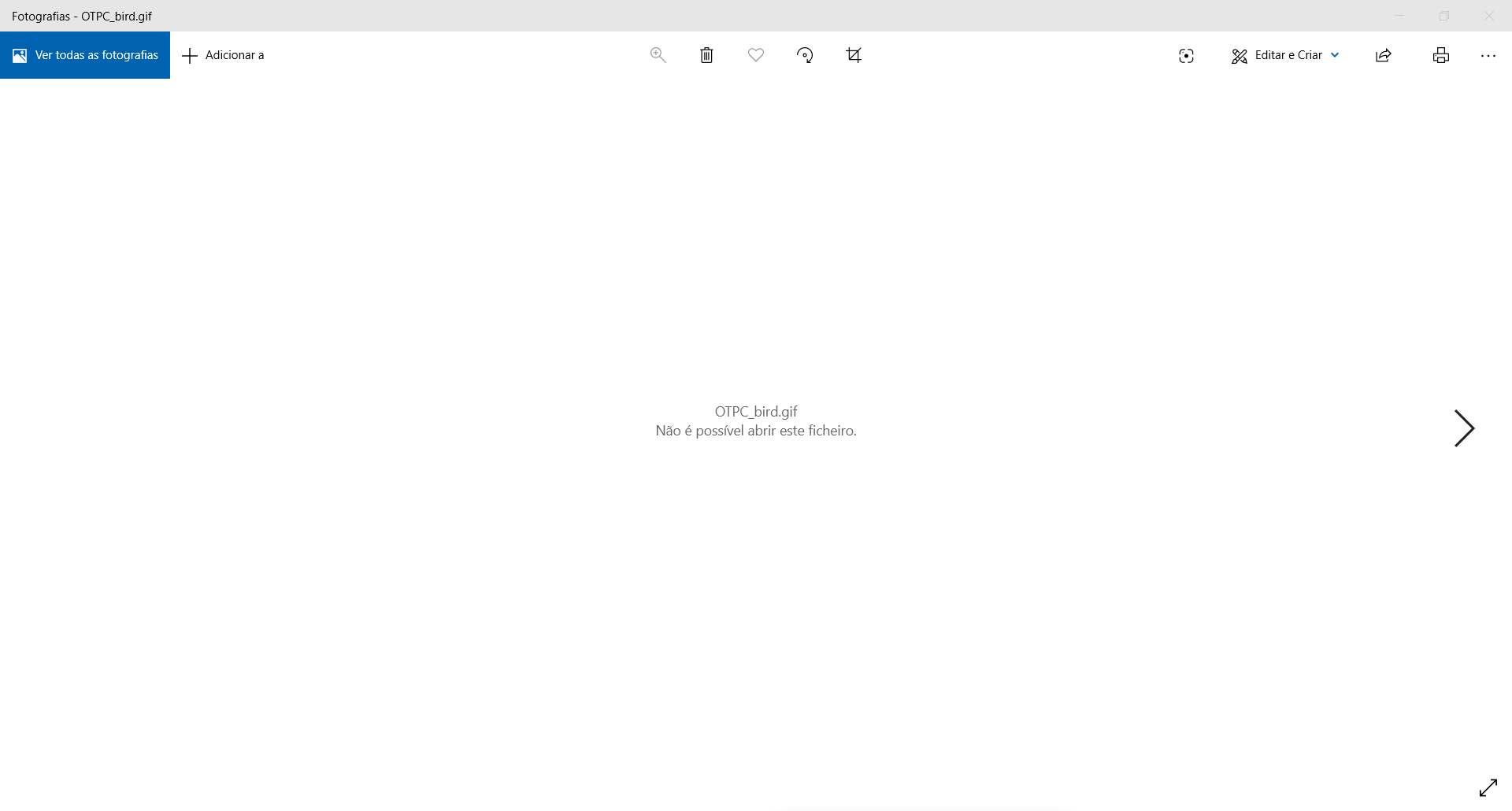


Figura - Criptografia de "bird.gif" guardado em "OTPC\_bird.gif"

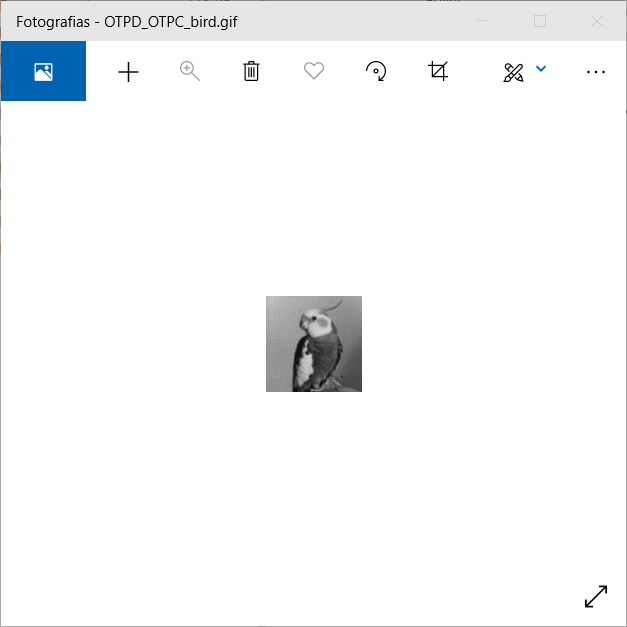


Figura - Descodificação de "OTPC\_bird.gif" guardado em "OTPD\_OTPC\_bird.gif"

Para verificar o correto funcionamento do par de ações codificação/descodificação, foi implementado uma função *testFiles* que verifica o conteúdo de dois ficheiros. Primeiro verifica se ambos os ficheiros têm o mesmo número de caracteres e depois verifica se são iguais caracter a caracter. Se alguma destas condições falhar, então a função gera um erro com uma mensagem apropriada. É possível verificar o resultado dos testes aos quatros ficheiros usados como exemplos na Figura 30.

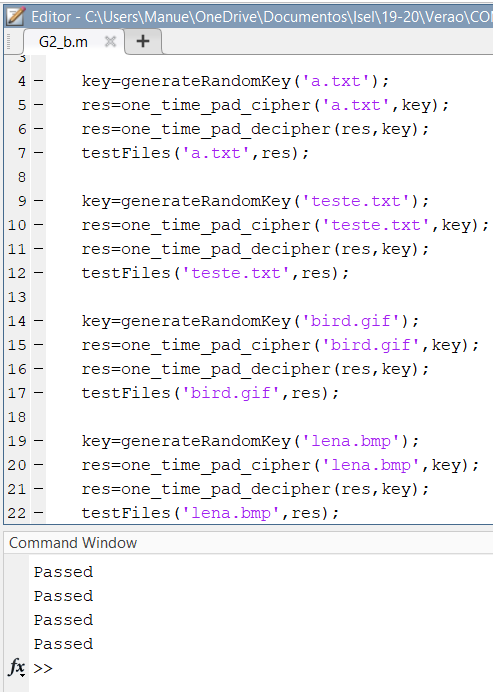


Figura - Resultado dos testes aos quatros ficheiros

# 4. Grupo 3

No terceiro grupo do trabalho prático é explorado os conceitos de codificação de canal através do controlo de erros. É pretendido realizar a deteção de erros com código de bit de paridade (8, 7) e também a correção de erros com código de Hamming (7, 4).

## 4.1 Alínea A

### 4.1.1 Função *parity\_check\_encode*

Para implementar a função *parity\_check\_encode*, foi decidido que esta função recebe como parâmetro de entrada o nome do ficheiro a codificar e que retorne o nome do ficheiro gerado com o conteúdo codificado.

A função está dividida em quatro partes:

1. Ler o conteúdo do ficheiro e transformar o vetor numa matriz binária;
2. Preparar a matriz resultante da codificação;
3. Operação da codificação;
4. Escrever o conteúdo num novo ficheiro;

Na primeira parte é lido o conteúdo do ficheiro para um vetor e transformar esse vetor numa matriz binária de Mx8, onde M equivale ao número de Bytes no ficheiro. Depois é guardado a transposta desta matriz, pois na operação de codificação, vai ser recorrido à indexação linear[[1]](#footnote-1) de matrizes do MATLAB, em que este percorre a todas as linhas por coluna em vez de todas as colunas por linha. Isto quer dizer que sem a transposição, a codificação não seria feita de uma maneira sequencial dos bits do ficheiro. Por isso ao fazer a transposição, a matriz resultante será de 8xM.

Na segunda parte é necessário fazer uma preparação, pois se em vez disto for feito uma concatenação da matriz, a função teria uma performance muito má por causa das operações internas de alocação e remoção de memória. Nesta preparação é preciso ter em conta aos bits restantes da mensagem para não haver percas de informação.

Na terceira parte é feita a codificação, recorrendo às capacidades de indexação linear de matrizes do MATLAB.

Na quarta parte é criado um ficheiro, cujo nome é o resultado da concatenação de “ParC\_” com o nome do ficheiro original, e escrito o conteúdo codificado. Este conteúdo, antes de ser escrito, é transposto outra vez e transformado para decimal, para ser escrito como caracter cada bloco de 8 bits.

### 4.1.2 Função *parity\_check\_decode*

Esta função é muito semelhante à função anterior, mas com algumas diferenças. Uma delas é no processo de descodificação é verificado se em cada bloco de mensagem existe um erro de transmissão. Isto é verificado a partir da soma de todos os bits, e verificar se o resto da divisão por 2 da soma é igual ao último bit do bloco. Se for igual quer dizer que não houve problemas de transmissão.

Para obter a mensagem do bloco, é tirado partido de algumas funcionalidades já implementadas, a transposição da matriz binária. Como se sabe que o último bit em cada bloco é o bit de paridade, então é guardado numa matriz auxiliar apenas os bits de mensagem, resultando uma matriz de 7xM.

A última diferença está no nome do ficheiro gerado com a mensagem agora descodificada, que resulta da concatenação de “ParD\_” com o nome do ficheiro codificado.

## 4.2 Alínea B

### 4.2.1 Função *hamming\_encode*

Para implementar esta função, foi decidido que esta recebe como parâmetro de entrada o nome do ficheiro a codificar e retorna como parâmetro de saída o nome do ficheiro resultante com a codificação.

A função está dividida em quatro partes:

1. Ler o conteúdo do ficheiro e transformar numa matriz binaria;
2. Preparar a matriz resultante da codificação;
3. Realizar a operação de codificação;
4. Guardar o resultado num novo ficheiro;

A primeira parte desta função é igual à primeira parte da função *parity\_check\_encode*.

Na segunda parte, a preparação é feita pelas mesmas razões dadas na função *parity\_check\_encode*.

Na terceira parte, a operação é realizada para cada quatro bits de mensagem. Depois é pegado nestes bits para calcular o valor dos bits de paridade.

Na quarta parte, esta função repete o mesmo processo que na função *parity\_check\_encode*, no entanto, a concatenação é feita com o prefixo “H74C\_”.

### 4.2.2 Função *hamming\_decode\_correct*

Esta função é semelhante à função anterior, no entanto, no processo de descodificação é realizada a correção de erros. Para realizá-la, é pegado nos bits de paridade e analisado a síndroma resultante. A síndroma revela que bit é necessário corrigir. Como só é relevante os bits de mensagem, só é verificado se a síndroma obtida equivale a um erro num bit de mensagem. A correção é feita a partir da soma desse bit com o valor 1 e é feito o resto da divisão por 2 desse valor.

Tal como na função *parity\_check\_decode*, esta função também pode tirar partido da transposição do bloco obtido, para filtrar os bits de paridade, guardando numa matriz auxiliar apenas os bits de mensagem.  
No final da função, a escrita da mensagem descodificada é realizada num ficheiro, cujo nome é o resultado da concatenação de “H74D\_” com o nome do ficheiro codificado.

## 4.3 Alínea C

|  |  |
| --- | --- |
| Figura - Teste às funções de parity com ausência de erros | Figura - Teste às funções de Hamming com ausência de erros |

|  |  |
| --- | --- |
| Figura - Teste às funções de parity com 1 bit em erro | Figura - Teste às funções de hamming com 1 bit em erro |
| Figura - Teste às funções de parity com múltiplos bits em erro | Figura - Teste às funções de hamming com múltiplos bits em erro, com apenas 1 bit em erro num bloco |
|  | Figura - Teste às funções de hamming com múltiplos bits em erro, com múltiplos bits em erro num bloco |

Sobre o resultado obtido na Figura 37, ao analisar o ficheiro descodificado, presente na Figura 39, é notável que comparando com o ficheiro original, este tem apenas 5 caracteres errados. Como foram inseridos 75 bits errados, mesmo que em cada bloco errado todos os bits tivessem errado, 6x7=42, houve blocos que continham apenas 1 ou 0 bits errados, sendo possível então fazer a correção desses blocos. De notar que para inserir erros em qualquer teste, foi recorrido à função *ber\_simulation* que simula uma transmissão com uma BER passada por parâmetro, ou seja, é muito improvável que se consiga reproduzir o mesmo efeito.

|  |  |
| --- | --- |
| Figura 38 - Ficheiro original | Figura 39 - Ficheiro descodificado |

# 5. Grupo 4

No quarto grupo do trabalho prático, é pretendido implementar uma simulação de transmissão de ficheiros ou de informação, usando o modelo de comunicação de Shannon. Este modelo está representado na Figura 40.



Figura 40 - Representação do modelo de Shannon

Para implementar a simulação, foi fornecido uma Graphic User Interface (GUI), sendo apenas necessário implementar as funcionalidades da mesma. Ao adicionar as funcionalidades, foi notado que era necessário modificar um pouco a GUI, adicionando apenas mais um campo à simulação de BER, demonstrado na Figura 41.

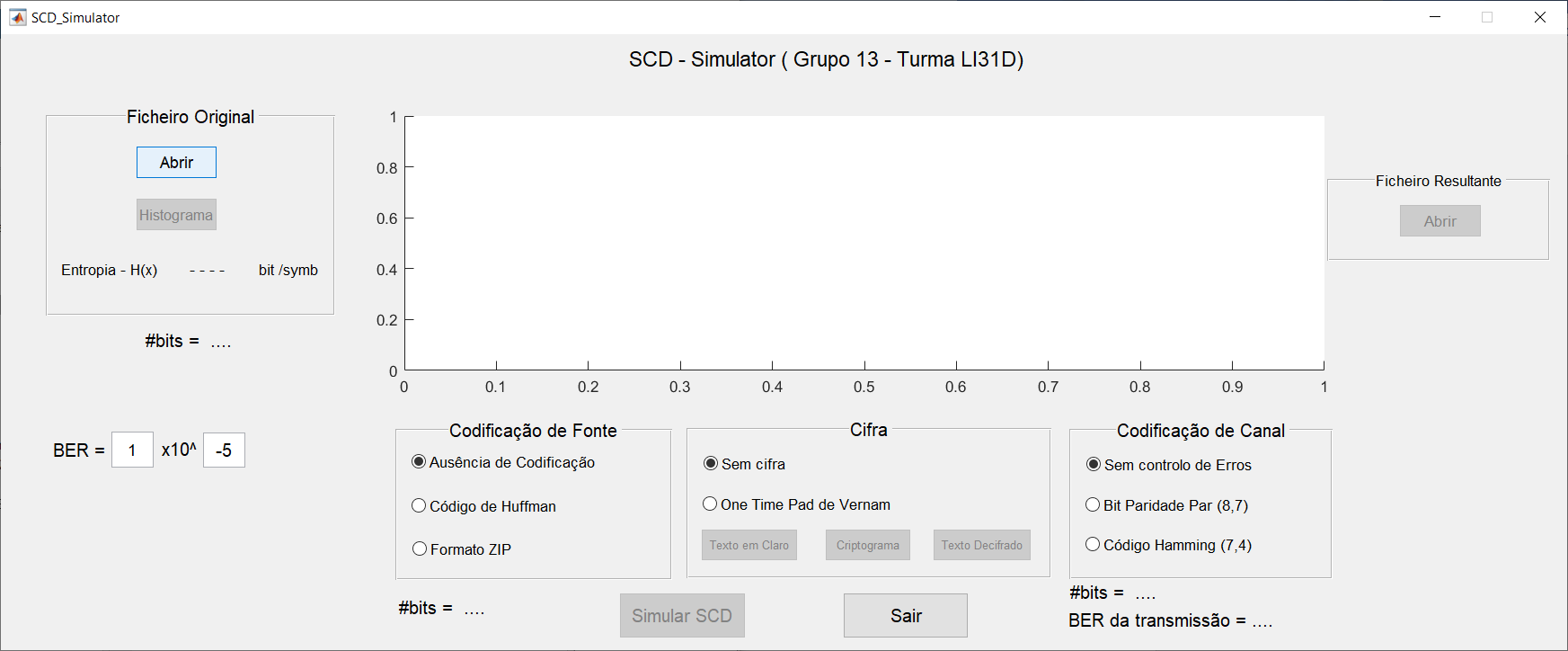


Figura 41 - Representação da GUI

Visto que as GUI’s do MATLAB funcionam através de funções de resposta após uma ação e como surgiu uma necessidade de partilha de informação entre funções, foi criado uma estrutura de dados onde é guardado toda a informação necessária para partilhar entre funções. Esta estrutura é criada quando é inicializado a GUI, tendo sido isto implementado na função *SCD\_Simulator\_OpeningFcn*.

## 5.1 Alínea (i)

Para obter o ficheiro selecionado pelo utilizador, foi necessário modificar a função de resposta do botão “Abrir” do “Ficheiro Original”, *openFileTX\_Callback*. Esta função guarda o ficheiro escolhido pelo utilizador no campo da estrutura.

Para calcular a entropia e apresentar o histograma do ficheiro selecionado, foi modificado a função de resposta do botão “Histograma”, *fileHistogram\_Callback*. Para fazer o cálculo e a apresentação do histograma, foi recorrido à função *entropy\_histogram* desenvolvido no Grupo 1 deste trabalho.

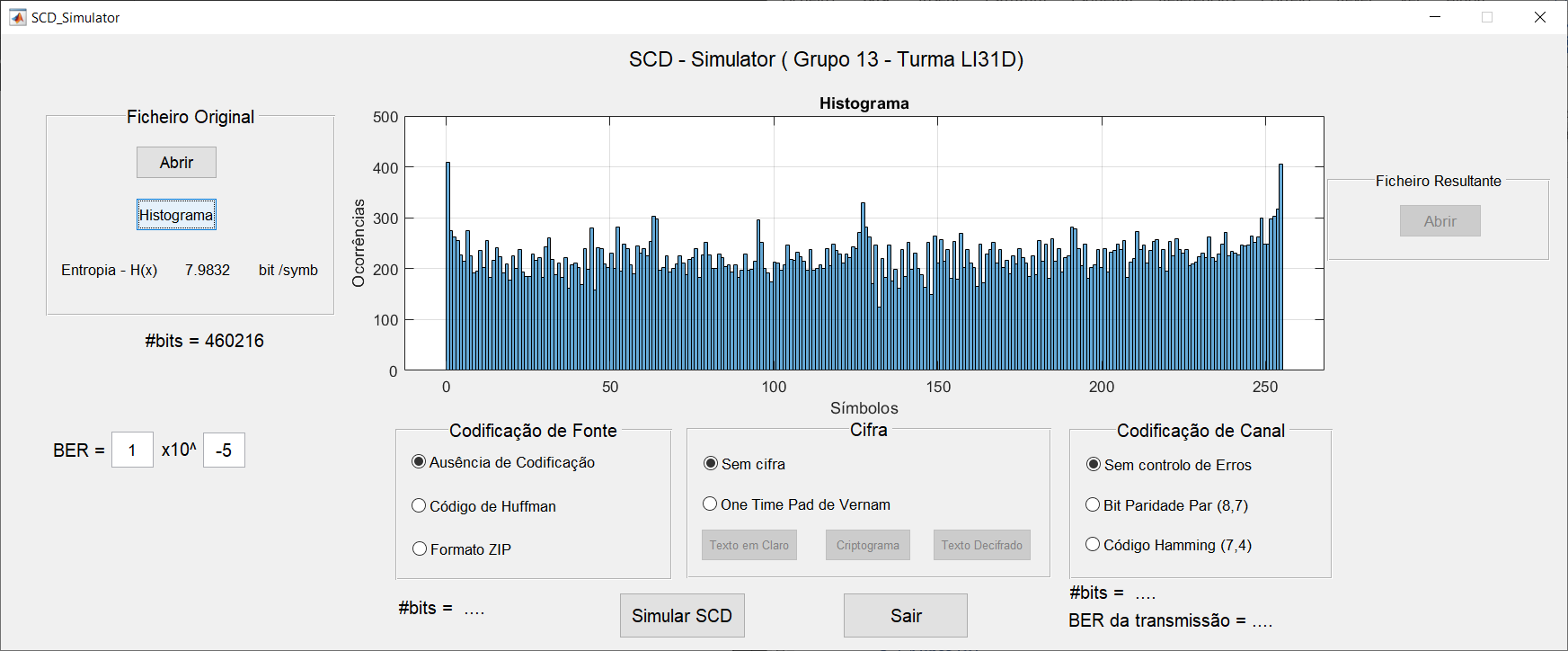


Figura 42 - Resultado obtido após carregar o ficheiro "lena.zip"

## 5.2 Alínea (ii)

Para implementar as funcionalidades de codificação e descodificação de fonte com código de Huffman, formato ZIP ou mesmo com ausência de codificação, foram adicionadas funcionalidades à função de resposta do botão “Simular SCD”, *simSCD\_Callback*. Segundo o modelo de Shannon, este tipo de codificação é a primeira operação a ser feita numa transmissão e a descodificação é a última. Para fazer a codificação, foi criada a função *codFonte* que recebe como parâmetro de entrada o nome do ficheiro e a escolha do utilizador e retorna o nome do ficheiro resultante da operação de codificação, o número de bits do ficheiro resultante e um dicionário.

Para obter a escolha do utilizador é lido os valores das *checkboxes* “Código de Huffman” e “Formato ZIP”, é dado um peso a cada valor e somado o resultado. O resultado obtido dá-nos a informação necessária para escolher que tipo de codificação é preciso fazer.

Caso o utilizador escolha “Ausência de Codificação” o nome do ficheiro retornado é igual ao de entrada.

Caso o utilizador escolha “Código de Huffman”, é chamado a função *huffman\_encode* que realiza a codificação de Huffman através da função do MATLAB *huffmanenco*. O ficheiro resultante desta codificação é o resultado da concatenação de “HuffC\_” com o nome do ficheiro de entrada.

Caso o utilizador escolha “Formato ZIP”, é chamado a função “zip\_encode” que codifica o ficheiro de entrada através da função do MATLAB *zip*. O ficheiro resultante desta codificação é o resultado da concatenação de “zip\_” com o nome do ficheiro de entrada.

Para a descodificação de canal, foi criada a função *decodFonte* que realiza o mesmo processo que *codFonte* mas descodificação.

A descodificação de Huffman é feita passando o dicionário da codificação e chamar a função do MATLAB *huffmandeco*. O ficheiro resultante da descodificação é o resultado da concatenação de “HuffD\_” com o nome do ficheiro de entrada.

A descodificação do formato zip é feita chamando a função do MATLAB *unzip*.

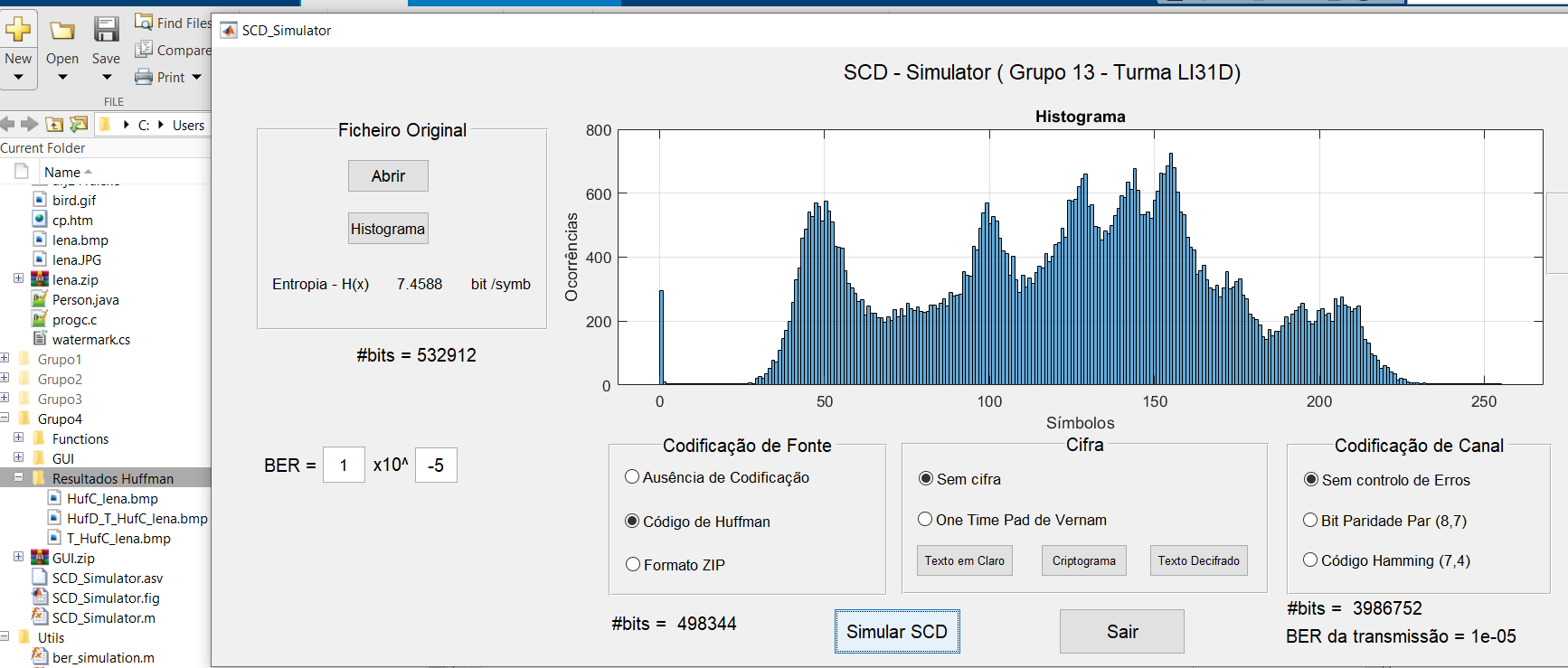


Figura 43 - Estado da GUI após o uso das funções de huffman ao ficheiro "lena.bmp"

|  |  |
| --- | --- |
| Figura 44 - Ficheiro original "lena.bmp" | Figura 45 - Ficheiro resultante "HuffD\_T\_HuffC\_lena.bmp" |

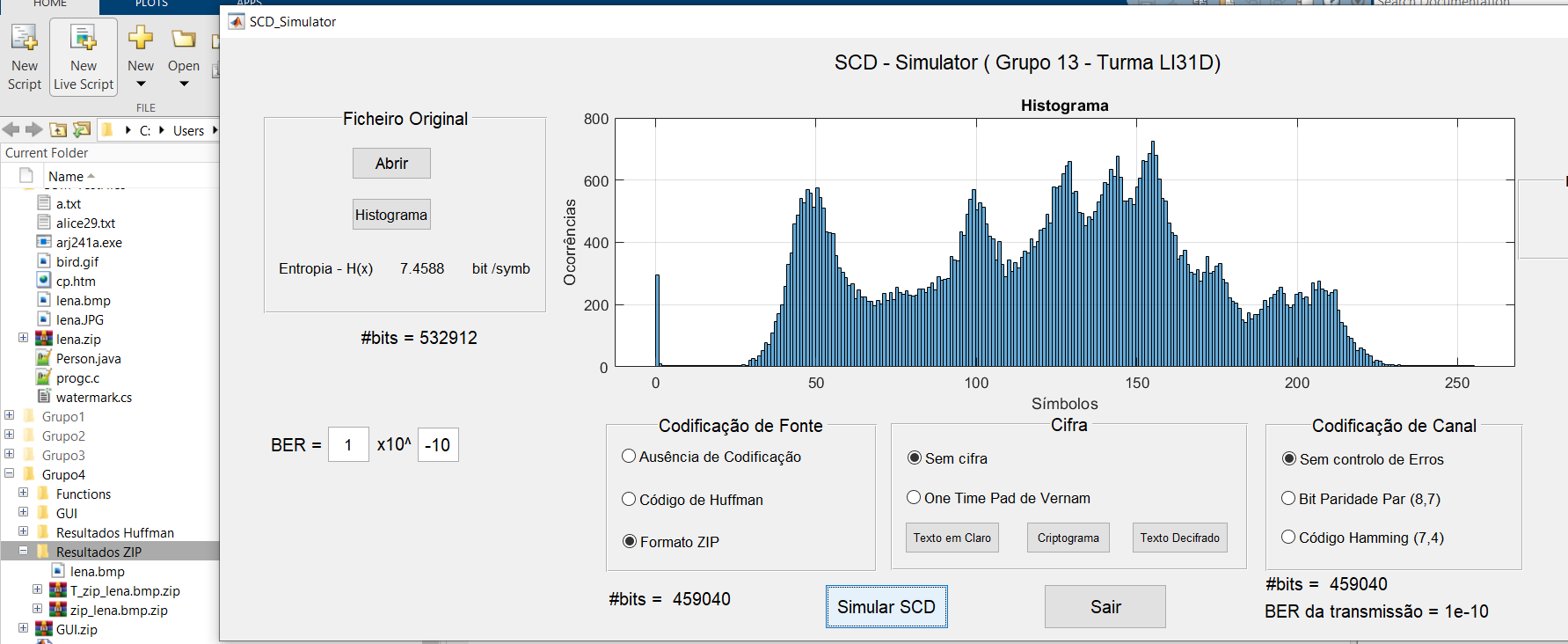


Figura 46 - Estado da GUI após o uso das funções de zip ao ficheiro "lena.bmp"

|  |  |
| --- | --- |
| Figura - Ficheiro original "lena.bmp" | Figura - Ficheiro resultante " lena.bmp" |

## 5.3 Alínea (iii)

Para adicionar a funcionalidade de cifra com sistema One Time Pad de Vernam, foram adicionadas funcionalidades à mesma função de simulação de SCD. Para implementar a criptografia e a descodificação, foram recorridos às funções realizadas no Grupo 2 deste trabalho.

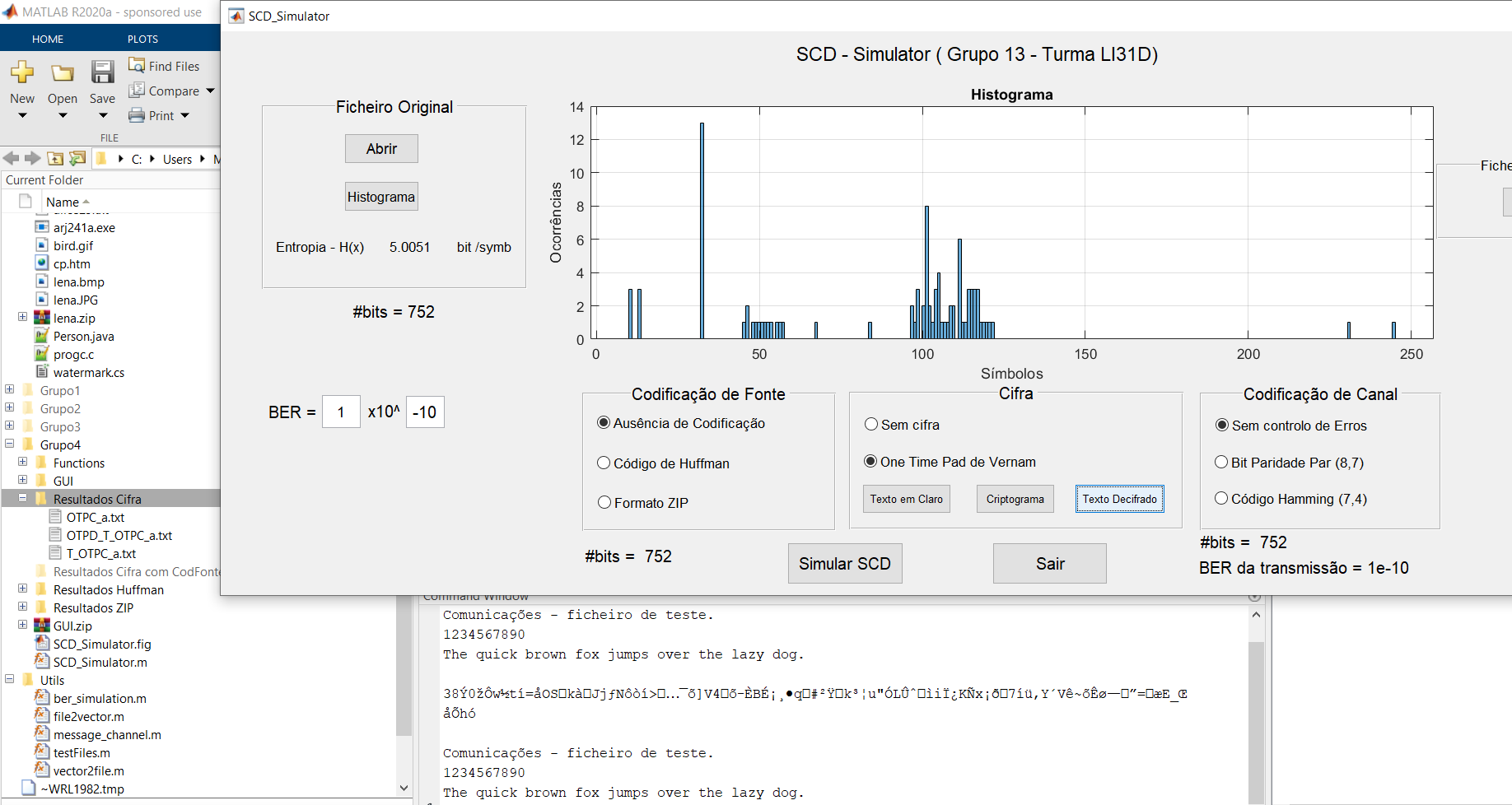


Figura 49 - Resultados da cifra ao ficheiro "a.txt" sem codificação de fonte

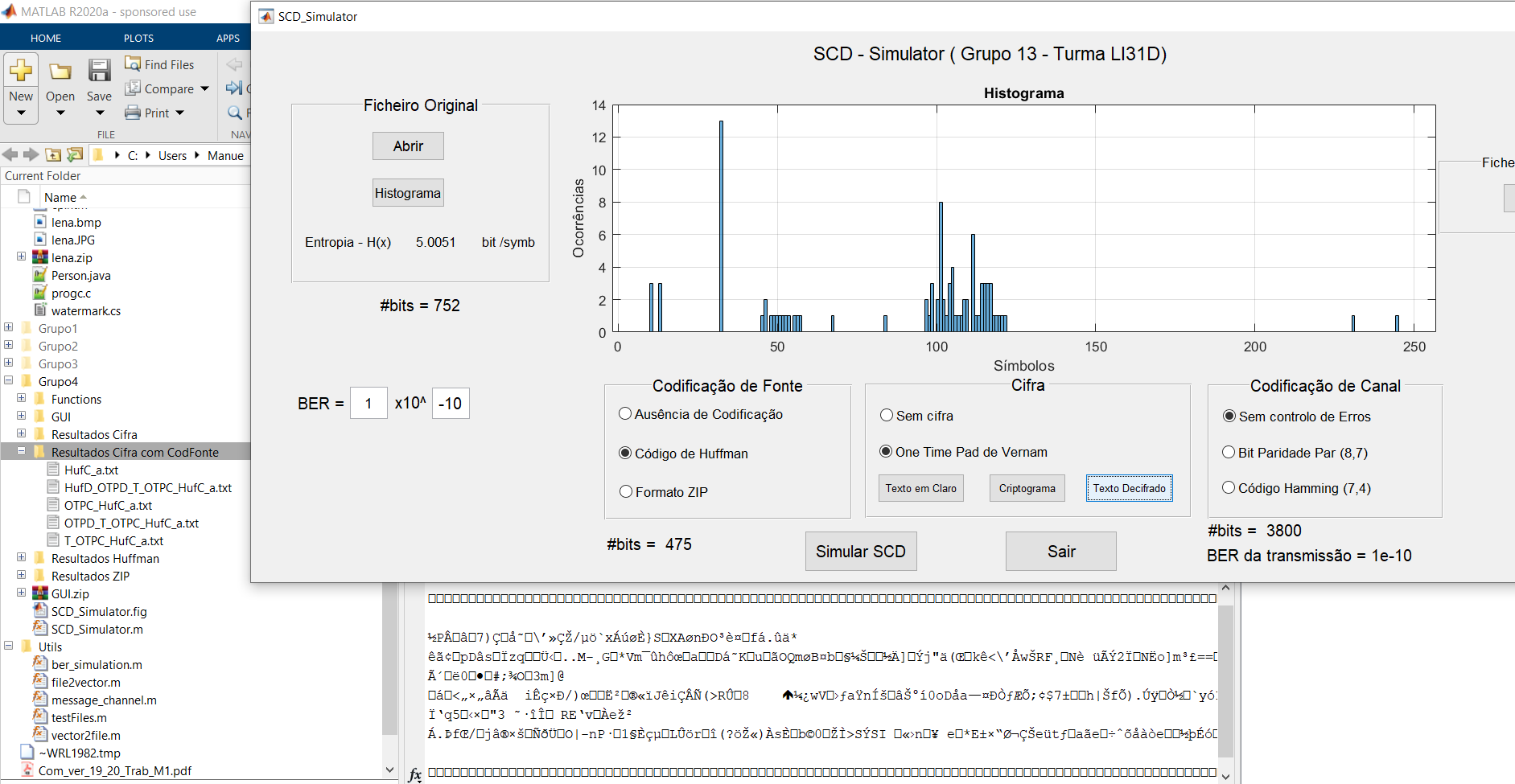


Figura 50 - Resultados da cifra ao ficheiro "a.txt" com codificação de fonte

|  |  |
| --- | --- |
| Figura 51 - Ficheiro original "a.txt" | Figura 52 - Resultado do ficheiro "a.txt" após codificação de fonte e cifra |

## 5.4 Alínea (iv)

Para simular a transmissão com BER configurável, é recorrido à função *ber\_simulation* usada no Grupo 3 como testes. A chamada desta função é feita após a codificação de canal, na função de simulação de SCD.

## 5.5 Alínea (v)

Para fazer a deteção de erros com código de bit de paridade par (8, 7), foi recorrido às funções desenvolvida no Grupo 3 deste trabalho, *parity\_check\_encode* e *parity\_check\_decode*.

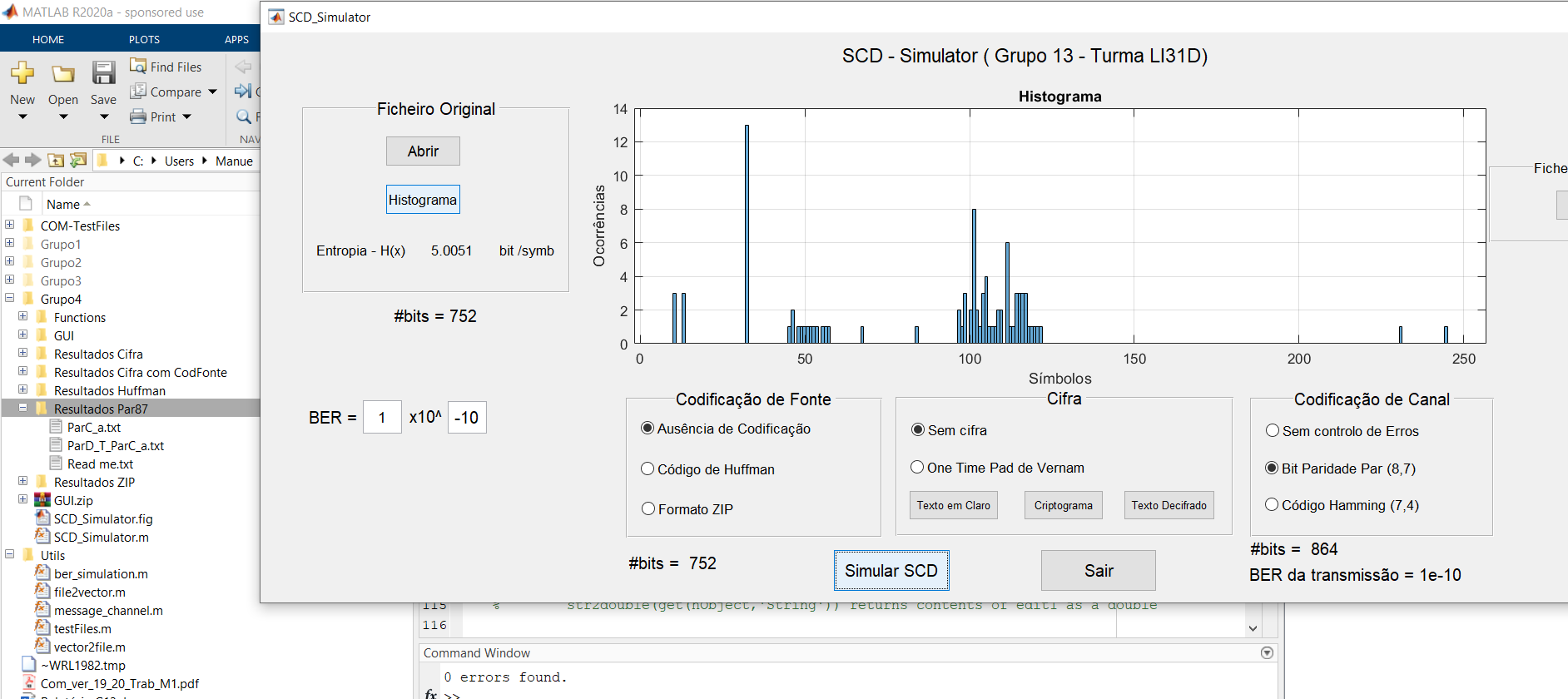


Figura 53 - Estado da GUI após a codificação de canal de bit de paridade par (8, 7)

## 5.6 Alínea (vi)

Tal como a deteção de bit de paridade par (8, 7), a codificação de Hamming (7, 4), foi recorrido às funções desenvolvidas no Grupo 3, *hamming\_encode* e *hamming\_decode\_correct*.

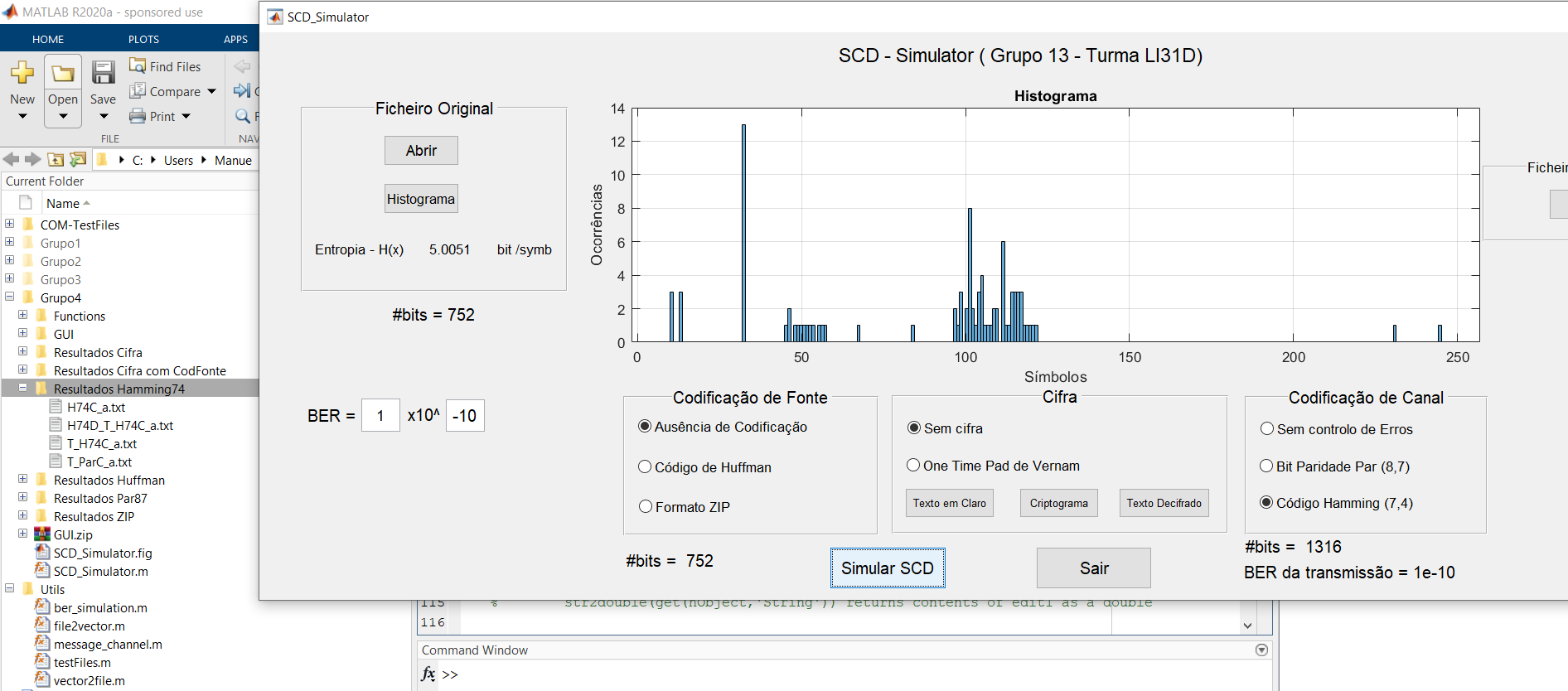


Figura 54 - Estado da GUI após a codificação de canal de Hamming (7, 4)

1. <https://www.mathworks.com/company/newsletters/articles/matrix-indexing-in-matlab.html> [↑](#footnote-ref-1)