Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Trabalho Prático Módulo 2

47206 : Tiago Alexandre Figueiredo Pardal (a47206@alunos.isel.pt)

48253 : Carlos Guilherme Cordeiro Pereira (a48253@alunos.isel.pt)

Relatório para a Unidade Curricular de Course
da Licenciatura em Engenharia Informática e de Computadores

Professor : Doutor Artur Ferreira

22 – 06 – 2022

Resumo

No âmbito do segundo módulo prático da cadeira, este relatório tem como propósito a elaboração dos algoritmos pretendidos, e análise dos resultados experimentais.

Através do trabalho realizado pretendemos demonstrar conhecimento sobre a primeira parte da matéria contendo tópicos tais como entropia e codificação e descodificação com técnicas de estatística e de dicionário.

Este relatório parte do pressuposto do acesso por parte do leitor ao código desenvolvido no âmbito do mesmo, não sendo assim necessário enunciá-lo em extensão, bastando apenas mencionar trechos do mesmo.

Índice

Lista de Figuras

Lista de Listagens

1

Criptografia

Os casos em estudo são um ficheiro de texto e uma imagem

Os histogramas aqui presentes encontram se em maior disponíveis juntamente com o código.

1.1 Cifra de César

A cifra de césar consiste em somar 3 a cada letra.

Assim sendo já sabemos que esta cifra nunca terá segurança perfeita.

Além de que o histograma do ficheiro cifrado será apenas um *shift* de 3 sobre o histograma do ficheiro original.

Ou seja, se o ficheiro original tem 3 ocorrências do carater *a* o ficheiro cifrado tem 3 ocorrências do carater *d* que corresponde a *a+3*.

É esperado que todos os ficheiros mantenham o mesmo valor de entropia.

1.1.1 Exemplo Experimental Ficheiro Texto

```
1 Comunicacoes – ficheiro de teste.
2 1234567890
3 The quick brown fox jumps over the lazy dog.
```

Listagem 1.1: Texto Plain Caesar

```
1 Frpxqlfdfrhv#0#ilfkhlur#gh#whvwh1
2 456789:;<3#
3 Wkh#txlfn#eurzq#ir{#mxpsv#ryhu#wkh#od}|#grj1
```

Listagem 1.2: Texto Cifrado Caesar

```
1 Comunicacoes – ficheiro de teste.
2 1234567890
3 The quick brown fox jumps over the lazy dog.
```

Listagem 1.3: Texto Decifrado Caesar

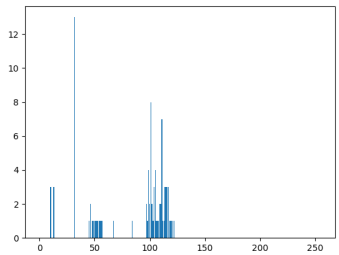

Figura 1.1: Histograma Texto Plano Caesar

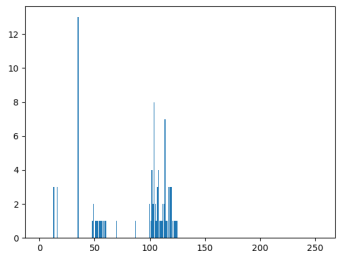

Figura 1.2: Histograma Texto Cifrado Caesar

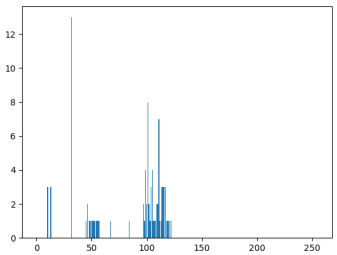

Figura 1.3: Histograma Texto Decifrado Caesar

O texto em plano tem a seguinte entropia: 4.926

O texto cifrado usando caesar tem a seguinte entropia: 4.926

O texto decifrado usando caesar tem a seguinte entropia: 4.926

Como é visível a entropia dos 3 ficheiros é igual, tal é de esperar visto a alteração efetuada sobre o texto cifrado ser constante.

Os resultados do histograma confirmam a teoria já explicada.

1.1.2 Exemplo Experimental Imagem

Neste caso, as 3 imagens aparentam ser iguais, tal deve se a estarmos a realizar apenas uma soma de 3 bits a cada pixel, o que leva a uma alteração muito reduzida.

Figura 1.4: Imagem Plana Caesar

Figura 1.5: Imagem Cifrada Caesar

Figura 1.6: Imagem Decifrada Caesar

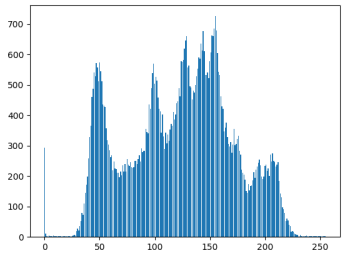

Figura 1.7: Histograma Imagem Plana Caesar

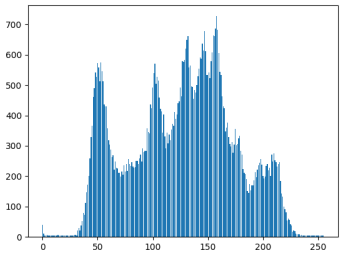

Figura 1.8: Histograma Imagem Cifrada Caesar

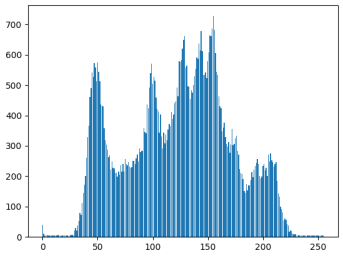

Figura 1.9: Histograma Imagem Decifrada Caesar

A imagem em plano tem a seguinte entropia: 7.459

A imagem cifrada usando caesar tem a seguinte entropia: 7.461

A imagem decifrada usando caesar tem a seguinte entropia: 7.461

A entropia ser igual para os 3 ficheiros, comparando com o original, no entanto diverge do ficheiro original ligeiramente.

Esta anomalia deve se à forma como efetuamos a escrita da imagem num ficheiro, pois como é visível pelo histograma, apenas um elemento difere entre o original e decifrado, e este é o elemento 0000, que corresponde a **NULL**, ou seja a nossa codificação da imagem é mais eficiente que a original, e não escreve no ficheiro valores nulos desnecessários.

Esta anomalia apenas se verifica quando ciframos e deciframos imagens.

1.2 Cifra de Vernam

A cifra de vernam consiste em gerar uma chave única com dimensão igual ou superior à mensagem, e soma la à mensagem a enviar.

Na nossa implementação a chave tem quase sempre dimensão superior ao ficheiro cifrado.

Para o exemplo da imagem não iremos a presentar aqui a chave, mas a mesma encontra se disponível, com o código enviado.

Esta cifra em teoria alcança segurança perfeita, se a chave for enviada através de uma fonte segura.

O histograma do ficheiro cifrado tende a demonstrar o uso de bits de forma semelhante ao longo de todos, ou seja, uma alta entropia.

1.2.1 Exemplo Experimental Ficheiro Texto

```
1 Comunicacoes – ficheiro de teste.
2 1234567890
3 The quick brown fox jumps over the lazy dog.
```

Listagem 1.4: Texto Plain Vernam

```
1 5Y£ûél2w
2 Ü=fÇ»7s\P~ßûG2x;KæÁ$ïéc
3 ñkhRr<èGîÑâ]Ë:
4 cé7,¿fkzH4r
```

Listagem 1.5: Texto Cifrado Vernam

```
1 tXzi:ÀL)d¿UÉx0äCâß9~sÊ7tM$2rÖáòL{ÏÍ
2 nKóa<RZmgo.ëóLh%çR@ëëÆF/xUoÏ&éEâaÆ&3í&ç(#ãòûÅêÜ£2<ód]BÑoa%;YG<pEçj
      ß
3 nÇï ÚÒ+àYÓZVp)QqTC`4[Häc
4 9ê
5 ]éîêD5DoÉ"Î:Ñ<[W>Æ8CMeâquï&ùGgfÔÉZÅ/éLë.30Á0=]ò{~"fa%Ç

6 î[9XéE'Ñxcåp>
```

Listagem 1.6: Chave Vernam

```
1 Comunicacoes – ficheiro de teste.
2 1234567890
3 The quick brown fox jumps over the lazy dog.
```

Listagem 1.7: Texto Decifrado Vernam

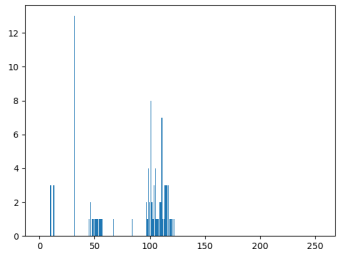

Figura 1.10: Histograma Texto Plano Vernam

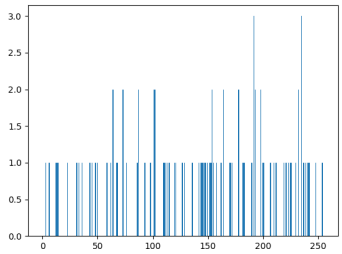

Figura 1.11: Histograma Texto Cifrado Vernam

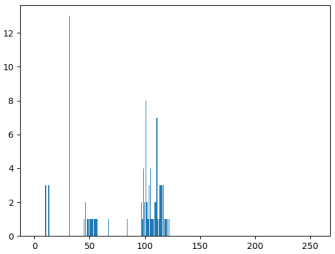

Figura 1.12: Histograma Texto Decifrado Vernam

O texto em plano tem a seguinte entropia: 4.926

O texto cifrado usando vernam tem a seguinte entropia: 6.249

O texto decifrado usando vernam tem a seguinte entropia: 4.926

A entropia do ficheiro original e decifrado é igual, o que aponta, em conjunto com o histograma e os resultados experimentais, que os ficheiros são idênticos.

A entropia do ficheiro cifrado em vernam à partida é sempre um valor superior ao ficheiro de origem, visto devido à aleatoriedade da chave levar a uma maior distribuição dos bytes usados.

1.2.2 Exemplo Experimental Imagem

Figura 1.13: Imagem Plana Vernam

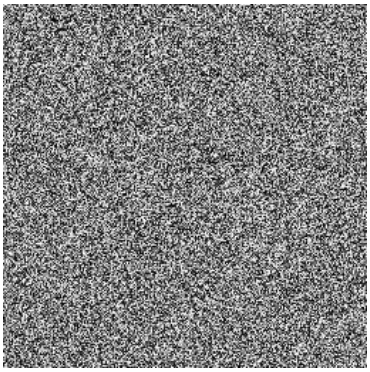

Figura 1.14: Imagem Cifrada Vernam

Figura 1.15: Imagem Decifrada Vernam

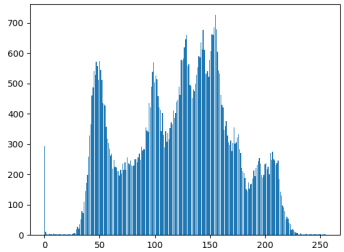

Figura 1.16: Histograma Imagem Plana Vernam

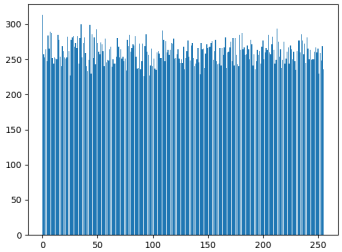

Figura 1.17: Histograma Imagem Cifrada Vernam

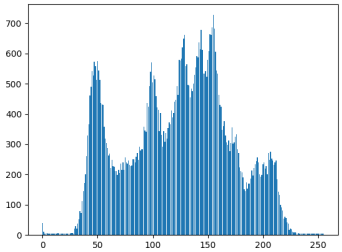

Figura 1.18: Histograma Imagem Decifrada Vernam

A imagem em plano tem a seguinte entropia: 7.459

A imagem cifrada usando vernam tem a seguinte entropia: 7.997

A imagem decifrada usando vernam tem a seguinte entropia: 7.461

A diferença da entropia entre a imagem original e a decifrada, assim como as diferenças visíveis no histograma, já se encontram devidamente explicadas no exemplo de imagem de césar.

De resto os valores de entropia e os histogramas são à volta do que esperávamos.

1.3 Cifrador/ Decifrador de Imagens Monocromáticas

Para esta implementação usamos a cifra de vernam já implementada anteriormente.

Para além de criarmos um ficheiro para guardar a chave, criamos também um ficheiro para guardar a informação da janela.

Este último ficheiro poderia ser acrescentado ao ficheiro da chave.

```
1 lft_x=120
2 lft_y=125
3 rgt_x=180
4 rgt_y=180
```

Listagem 1.8: Informação Imagem

1.3.1 Resultados Experimentais

Figura 1.19: Imagem Monocromática Plana

Figura 1.20: Imagem Monocromática Cifrada

Figura 1.21: Imagem Monocromática Decifrada

2

Cyclic Redundancy Check

É de resalvar que carateres não reconhecidos por vezes não são apresentados em relatório mas estão presentes nos ficheiros. Tal deve se à formatação do documento.

2.1 Introdução

A verificação cíclica de redundância ou CRC é baseado na teoria de códigos de correção de erros cíclicos. O uso de códigos cíclicos sistemáticos, que alteram a mensagem adicionando um valor de verificação de tamanho fixo, com o propósito de deteção de erros. Códigos cíclicos são simples de implementar e possuem o benefício apresentar ótimas respostas na deteção de erros causados pela "Rajada" de bits: sequencias continuas de símbolos de dados errados (*Burst Errors*).

2.2 Análise de resultados

Na implementação deste algoritmo foram criados os métodos:

- **crc_file_compute**, este recebendo um ficheiro de entrada calcula e escreve para um ficheiro de saída o conteúdo do ficheiro de entrada mais o seu valor de CRC;
- **crc_file_check**, este verifica a integridade dos dados, ou seja, calcula novamente o CRC do ficheiro original e compara com o valor de CRC do ficheiro gerado anteriormente;
- **make_errors**, este método gera propositadamente um ficheiro de saída com o conteúdo do ficheiro de entrada, mas com erros para a verificar se mesmo assim o CRC se mantém.

Para os ficheiros de teste disponibilizados fomos capazes de calcular e verificar o valor de CRC, como também verificamos que mesmo apos forçar erros nos ficheiros os mesmos preservam o valor de CRC original.

2.3 Exemplos

Tomando como exemplo o ficheiro de testes "*a.txt*"e o ficheiro "*Person.java*"temos:

```
1 Comunicacoes – ficheiro de teste.
2 1234567890
3 The quick brown fox jumps over the lazy dog.
```

Listagem 2.1: a.txt Original

```
1 Comunicacoes – ficheiro de teste.
2 1234567890
3 The quick brown fox jumps over the lazy dog.
4
5 0x13
```

Listagem 2.2: a.txt CRC

```
>qicacoes – ficheiro de teste.
1234567890
The quick brown fox jumps over the lazy dog.

0x13
```

Listagem 2.3: a.txt Com erros

```
import java.util.StringTokenizer;
import java.util.GregorianCalendar;
import java.util.Calendar;

import oursource.comparacoes.*;

/**
  Classe Pessoa.
  Esta classe implementa a interface Serializable de forma a
  permitir escrever instancias suas em ficheiro.
 */
public class Person implements java.io.Serializable, Comparable {

  /* Atributos que no podem variar numa pessoa. */
...

} /* Fim da classe Person */
```

Listagem 2.4: Excerto Person.java Original

```
import java.util.StringTokenizer;
import java.util.GregorianCalendar;
import java.util.Calendar;

import oursource.comparacoes.*;

/**
  Classe Pessoa.
  Esta classe implementa a interface Serializable de forma a
```

```
11   permitir escrever instancias suas em ficheiro.
12  */
13 public class Person implements java.io.Serializable, Comparable {
14
15   /* Atributos que no podem variar numa pessoa. */
16 ...
17
18 } /* Fim da classe Person */
19
20
21 0x52
```

Listagem 2.5: Excerto Person.java CRC

```
1  R&Pm,o@WU+~+b05KC
2  G@Odm−3Bi4T&KWvEthQqr\^0$R5BDT8XoL~wV#WhhBUp;wbFboTO '3,!d_]wRNE
       [ /M*rxG~zvFx}cN9

3
4  6.s P8s o|!@f
5  −Q
6  fIE~w^> !|
7  OYsMsA"xM

8  ghv*awci%%U+Ualizable, Comparable {
9
10   /* Atributos que no podem variar numa pessoa. */
11
12 ...
13
14 } /* Fim da classe Person */
15
16
17 0x52
```

Listagem 2.6: Excerto Person.java Com erros

```
++++ CRC check error ++++
a.txt   !SUCCESS!
Person.java   !SUCCESS!

++++++ Make errors ++++++
File :  a.txt
Error:  0.0   !SUCCESS!
Error:  0.01    !SUCCESS!
Error:  0.1   !SUCCESS!
Error:  0.5   !SUCCESS!
Error:  1.0   !SUCCESS!
Error:  5.0   !SUCCESS!

File :  Person.java
Error:  0.0   !SUCCESS!
Error:  0.01    !SUCCESS!
Error:  0.1   !SUCCESS!
Error:  0.5   !SUCCESS!
Error:  1.0   !SUCCESS!
Error:  5.0   !SUCCESS!
```

Listagem 2.7: Excerto mensagens de consola na funcao de testes

Podemos comprovar tanto pelo conteúdo dos ficheiros tal como pelas mensagens de *output* da função de testes o correto funcionamento do algoritmo implementado.

Uma vez que os testes abrangeram um largo leque de ficheiros não achamos necessidade de integrar o par codificador/descodificador de código unário (*comma code*) desenvolvido no exercício 3 do primeiro módulo do trabalho prático, mas a maneira de como implementaríamos era de codificar um ficheiro usando a técnica de *comma code* apos a codificação do ficheiro liamos o mesmo em binário para o calculo do CRC, apos isso convertíamos o CRC para *comma code* com recurso ao dicionário presente no ficheiro para proceder a escrita do mesmo.

3

Sistema de Comunicação Digital (SCD)

3.1 Introdução

Foi proposto que simulássemos um Sistema de Comunicação Digital (SCD) com $T_b = 1ms$ e as seguintes características:

Figura 3.1: SCD

- **Emisor** - O sinal $x(t)$ presente na saída do emissor pode ser:

  NRZ-Unipolar - $x_{NRZ}(t) = 5\Pi(\frac{t}{Tb})$.

  PSK - $x_{PSK0}(t) = 2cos(2\pi 2000t)$ e $x_{PSK1}(t) = 2cos(2\pi 2000t)$.

  Na geração dos sinais, cada tempo de bit é representado por 10 amostras.

- **Canal de Comunicação** - O sinal de saída é $y(t) = x(t) + n(t)$, em que α é um valor no intervalo $]0, 1]$ e $n(t)$ é o sinal de ruído. A relação sinal-ruído deverá ser parametrizável.

- **Recetor** - Este bloco é baseado no conceito da deteção coerente.

3.2 Implementação

Foi nos pedido que implementasse mos dois algoritmos de emissor/recetor de um SCD, o *Non-Return to Zero* ou NRZU e o *Phase Shift Keying* ou PSK.

Na implementação do algoritmo do NRZU, uma vez que e uma onda quadrada, muito sucintamente multiplicamos o dados que recebemos pelo tempo de bit e cada bit de dados ficou codificado com a sua amplitude.

Na implementação do algoritmo do PSK uma vez que são dois sinais para codificar bits tivemos de verificar as suas transições.

3.3 Resultados experimentais

Inicialmente para comprovarmos a eficácia do nosso algoritmo usamos e para facilitar a correção de algum possível erro, usamos, para condições ideais a sequencia binaria de 10110001.

Para o NRZU o gráfico desenhado foi:

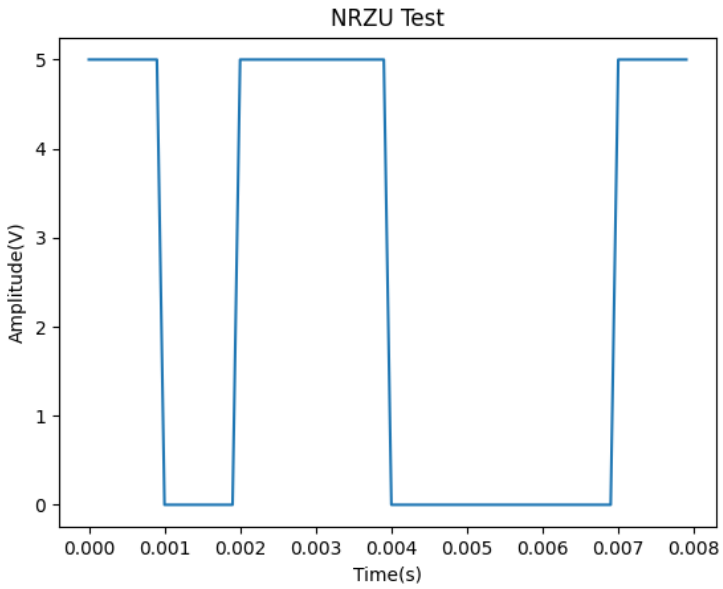

Figura 3.2: NRZU test

Para o PSK o gráfico desenhado foi:

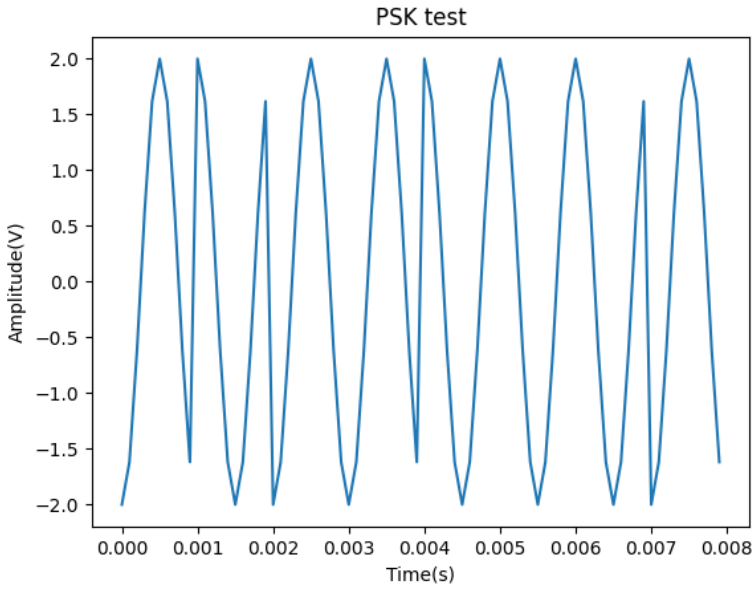

Figura 3.3: PSK test

Uma vez que em condições normais os dados enviados são a identidade dos dados recebidos não houve a necessidade de duplicar os gráficos.

Apos o comprovar da eficácia do nosso algoritmo executamo-lo nos ficheiros de teste disponibilizados e tivemos os seguintes resultados:

Para $\alpha = 1$ e ruido variável:

NRZU:

```
++++ NRZU coder/decoder ++++
Alpha: 1.0
With noise
File: a.txt BER: 0.45611702127659576
File: alice29.txt BER: 0.4309980932342692
File: cp.htm BER: 0.48322639416983526
File: Person.java BER: 0.3976177730192719
File: progc.c BER: 0.4234985610421085
```

Listagem 3.1: Excerto mensagens de consola na função de testes NRZU

PSK:

```
++++ PSK Modulator/Demodulator ++++
Alpha: 1.0
With noise
File: a.txt BER: 0.45611702127659576
File: alice29.txt BER: 0.4309980932342692
File: cp.htm BER: 0.48322639416983526
File: Person.java BER: 0.3976177730192719
File: progc.c BER: 0.4234985610421085
```

Listagem 3.2: Excerto mensagens de consola na função de testes PSK

Para dois qualquer α e ruido constante:

NRZU:

```
Without noise
...
Alpha: 0.2
File: a.txt BER: 0.0
File: alice29.txt BER: 0.4309980932342692
File: cp.htm BER: 0.48322639416983526
File: Person.java BER: 0.0
File: progc.c BER: 0.4234985610421085
...
Alpha: 0.8
File: a.txt BER: 0.45611702127659576
File: alice29.txt BER: 0.4309980932342692
File: cp.htm BER: 0.48322639416983526
File: Person.java BER: 0.3976177730192719
File: progc.c BER: 0.4234985610421085
```

Listagem 3.3: Excerto mensagens de consola na funcao de testes NRZU

PSK:

```
Channel without noise
...
Alpha: 0.1
File: a.txt BER: 0.45611702127659576
File: alice29.txt BER: 0.4309980932342692
File: cp.htm BER: 0.48322639416983526
File: Person.java BER: 0.3976177730192719
File: progc.c BER: 0.4234985610421085
...
Alpha: 0.4
File: a.txt BER: 0.45611702127659576
File: alice29.txt BER: 0.4309980932342692
File: cp.htm BER: 0.48322639416983526
File: Person.java BER: 0.3976177730192719
File: progc.c BER: 0.4234985610421085
```

Listagem 3.4: Excerto mensagens de consola na funcao de testes PSK

Como podemos comprovar quando α é constante e com ruido variável temos um alta taxa de erro ou *BER*, no entanto quando o ruido e constante mas α variável podemos ainda ter um *BER* relativamente alto tal como nulo, ou seja, um ficheiro idêntico ao inicial.

4

SCD com uso a Arduíno

O **SCD** realizado encontra se dividido em três partes:

- O Emissor, no nosso caso o Arduíno, envia em ciclo a *String* "Sending Info"a cada 2 segundos.
- O Recetor, em Python, tenta receber uma mensagem, num período de 2 segundos, estipulado por um timeout de igual duração, e escreve num ficheiro de texto.
- O Canal de Comunicação é um cabo USB

4.1 Canal de Comunicação

Sendo o canal de comunicação um cabo USB, a partir dos seus protocolos podemos depreender como funciona este **SCD** especifico.

O protocolo usb faz uso de uma onda quadrada do tipo *No Return to Zero - Space* NRZ-S

Este tipo de onda quadrada carateriza se pela alternação entre +v e 0 com cada valor de bit '0'.

O protocolo usb faz uso do NRZ-S com a técnica de *bit-stuffing*, em que a cada 6 bit '1' consecutivos é inserido um bit '0' extra, de forma a implementar um mecanismo de sincronismo entre o emissor e recetor.

O cabo usb na sua implementação mais rudimentar é composto por apenas 4 cabos:

- 2 cabos de fornecimento de energia, **POWER 5V** e **GND**
- 2 cabos de data **+DATA** e **-DATA**

Versões mais recentes variam no tipo de cabos, acrescentando cabos de data, como é o caso do usb 3.0.

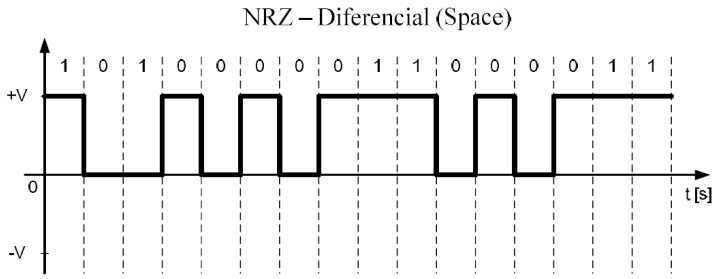

Figura 4.1: Exemplo onda quadrada NRZ-S

4.2 Descrição SCD Arduíno

O presente SCD pode ser descrito da seguinte forma:

- Tipo de Ligação: Ponto a Ponto
- Quanto à Direção da Transmissão: *Simplex*
- Tipo de Informação Enviada: *String*

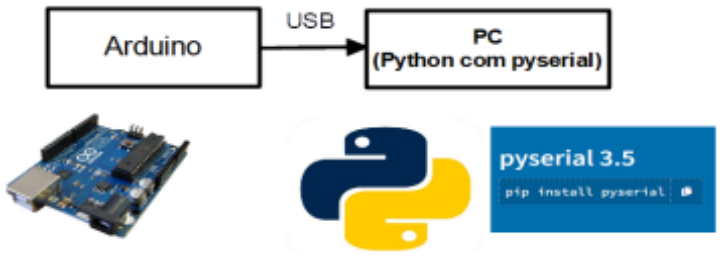

Figura 4.2: SCD Arduíno

Neste capítulo não apresentamos resultados experimentais, por não acharmos que estes sejam relevante para o caso, visto que seria apenas o nosso ficheiro do tipo *.txt* com a *String* "Sending Info".