
**Departamento de Engenharia de Eletrónica e Telecomunicações e de Computadores**

# Trabalho Prático – Módulo 2

Autores: 48089 André Filipe Pina Páscoa
48280 André Filipe do Pilar de Jesus
48287 Nyckollas Brandão

LEIC41D Grupo 15

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

Professor: Engenheiro Artur Ferreira

30 – 06 – 2022

<< Esta página foi intencionalmente deixada em branco >>

# Resumo

O presente trabalho incide sobre a segunda parte do programa da unidade curricular de Comunicação Digital: sistemas de comunicação digital (SCD), cifra, codificação de canal e outras técnicas.

Foram realizados quatro exercícios distintos, com recurso à linguagem de programação Python. O primeiro exercício incidiu sobre a matéria da cifra, onde foram implementados três sistemas criptográficos distintos. No segundo exercício, aplicou-se conhecimentos sobre códigos de controlo de erros, recorrendo à técnica *Cyclic Redundancy Check* (CRC). O terceiro exercício incidiu sobre a camada física de um SCD e à transmissão em banda base. O quarto e último exercício consistiu no desenvolvimento de um programa que estabelece uma ligação via USB em modo *simplex*, entre um PC e um Arduino.

Após a realização do trabalho, concluiu-se que os objetivos de aprendizagem foram alcançados, produzindo os resultados pretendidos e adquirindo conhecimentos da matéria em estudo.

**Palavras-chave**: sistema de comunicação digital, cifra, codificação de canal, CRC, banda base, sinais

# Abstract

This project focus on the second phase of the Digital Communications curricular unit (SCD), cipher, channel coding and other techniques.

Four exercises were implemented, with the help of the Python programming language. The first exercise consisted in the cipher subject, where three distinct cryptographic systems were implemented.

In the second exercise, knowledge on error control codes was applied, using *Cyclic Redundancy Check* (CRC). The third exercise focused on the physical layer of an SCD and baseband transmission. The fourth and last exercise consisted in the development of a program that establishes a connection via USB in simplex mode, between a PC and an Arduino.

After the project was completed, all the objectives were achieved, producing the intended results while acquiring the knowledge that was pretended.

**Keywords**: digital communication systems, cipher, channel coding, CRC, baseband, signals

# Índice

# Lista de Figuras

# Listagens

# 1 Introdução

Este trabalho teve como principal objetivo o estudo e aplicação de conceitos fundamentais sobre SCD (Sistemas de Comunicação Digital), cifra, codificação de canal e transmissão em banda base.

Inicialmente foi realizada uma revisão destes conceitos, com recurso às folhas de apoio e aos slides disponibilizados pelo professor.

É de ressaltar que todos os resultados produzidos, incluindo comentários no código fonte, estão em língua inglesa, à exceção do presente documento escrito em português.

Após a revisão dos conceitos em estudo, iniciou-se o desenvolvimento do exercício 1. Este exercício teve como principal objetivo o desenvolvimento de três pares cifrador/decifrador, aplicando diferentes técnicas de cifra.

Com o primeiro exercício resolvido, passou-se à implementação do exercício 2, que teve como principal objetivo o estudo de códigos de controlo de erros, com a técnica *Cyclic Redundancy Check* (CRC). Nesse exercício implementaram-se duas funções, uma que calcula o CRC de um ficheiro e outra que verifica a integridade do mesmo através do CRC calculado.

O terceiro exercício incidiu sobre os elementos da camada física de um SCD, o emissor, o canal de comunicação e o recetor.

O quarto e último exercício, teve como objetivo o desenvolvimento de um programa, que estabelece uma ligação via USB em modo *simplex*, entre um PC e um Arduino.

Todos os exercícios foram implementados com recurso à linguagem Python, visto que é uma linguagem muito versátil e tem uma vasta biblioteca de módulos uteis para os propósitos deste trabalho.

No seguinte capítulo estão apresentados os processos de resolução dos quatro exercícios.

# 2 Resolução dos Exercícios

Este capítulo contém a explicação das decisões tomadas na resolução dos exercícios.

Todos os exercícios foram devidamente comentados e testados.

Para poder executar os programas em Python, é necessário ter instalado uma versão da linguagem igual ou superior à 3.9, e as bibliotecas utilizadas no seu desenvolvimento como, por exemplo, matplotlib.

## 2.1 Exercício 1

O exercício 1 consistiu na realização de três pares cifrador/decifrador, onde foram aplicados os conhecimentos adquiridos sobre sistemas criptográficos.

A diretoria do exercício 1 está dividida em três subdiretorias:

- ceaser – contém o par cifrador/decifrador utilizando a Cifra de César;
- monochromatic – contém o par cifrador/decifrador de imagens monocromáticas;
- vernam – contém o par cifrador/decifrador utilizando a cifra de Vernam (*One Time Pad*).

### 2.1.1 Cifra de César

Começou-se por desenvolver as funções *ceasar_cipher* e *ceasar_decipher*, que realizam a cifra e a decifra, respetivamente, de um ficheiro de entrada, com recurso à Cifra de César.

A Cifra de César é uma das mais simples e conhecidas técnicas de criptografia, na qual cada letra do texto é substituída por outra, que se apresenta no alfabeto abaixo dela um número fixo de vezes (*shift*).

Ambas as funções recebem os seguintes parâmetros:

- o ficheiro de entrada, com o texto em claro no caso do cifrador, e com o texto cifrado no caso do decifrador;
- o ficheiro de saída, onde o resultado da cifra/decifra é escrito;
- o valor do *shift* da cifra.

Na Figura 1 está ilustrado um exemplo da aplicação da cifra de César sobre o alfabeto, com um *shift* de 3 caracteres.

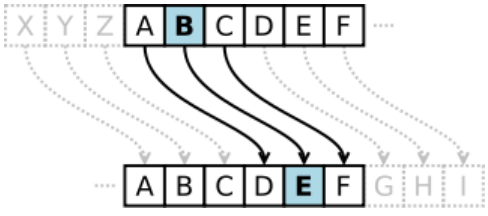


Figura 1 – Exemplo da cifra de César com *shift* de 3

O sistema criptográfico implementado funciona com textos em claro com todos os caracteres imprimíveis/visíveis da tabela ASCII. Para isso, itera-se por cada caracter do ficheiro de entrada, mantendo o caracter se este for um *escape caracter*, ou somando o valor do *shift* ao código ASCII deste, obtendo um novo caracter. Esta lógica aplicada na função *ceaser_cipher*, está apresentada na Listagem 1.

```
# Transverse the plain text
for i in range(len(plain_text)):
    char = plain_text[i]

    if char == '\n' or char == '\t' or char == '\r':
        cipher_text += char
        continue

    char_code = ord(char) + s
    if char_code > ASCII_LAST_PRINTABLE_CHAR:
        char_code -= ASCII_PRINTABLE_CHARS_LENGTH

    cipher_text += chr(char_code)
```

Listagem 1 – Cálculo do texto cifrado na função *ceaser_cipher*

A mesma lógica aplica-se, inversamente, na função *ceaser_decipher*, na Listagem 2.

```
# Transverse the cipher text
for i in range(len(cipher_text)):
    char = cipher_text[i]

    if char == '\n' or char == '\t' or char == '\r':
        decipher_text += char
        continue

    char_code = ord(char) - s
    if char_code < ASCII_FIRST_PRINTABLE_CHAR:
        char_code += ASCII_PRINTABLE_CHARS_LENGTH

    decipher_text += chr(char_code)
```

Listagem 2 - Cálculo do texto decifrado na função *ceaser_decipher*

Aplicou-se esta cifra a dois ficheiros do conjunto *CD_TestFiles.zip*: *a.txt* e *alice29.txt*.

**Resultados do ficheiro *a.txt* com *shift = 4*:**

- Entropia do texto em claro: H(X) = 4.8783 bit/símbolo;
- Entropia do texto cifrado: H(Y) = 4.8783 bit/símbolo;
- Entropia do texto decifrado: 4.8783 bit/símbolo.

Como é possível constatar, os valores da entropia do texto em claro, do texto cifrado e do texto decifrado são iguais, visto que todos os caracteres iguais no texto em claro terão como cifra o mesmo caracter no texto cifrado.

Na Figura 2, na Figura 3 e na Figura 4 estão apresentados os histogramas do texto em claro, do texto cifrado e do texto decifrado, respetivamente.

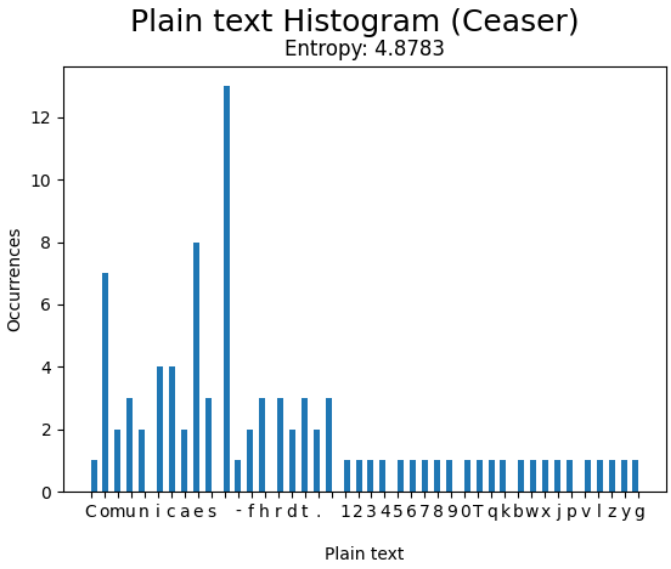


Figura 2 – Histograma do ficheiro *a.txt* utilizando a cifra de César

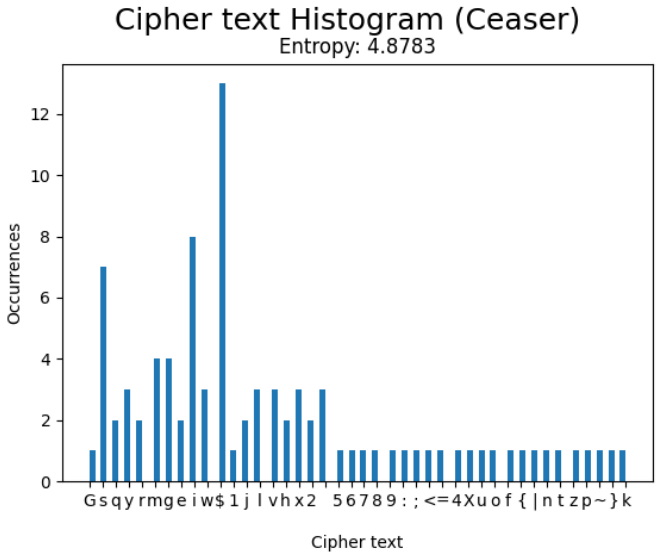


Figura 3 – Histograma do texto cifrado do ficheiro *a.txt* utilizando a cifra de César

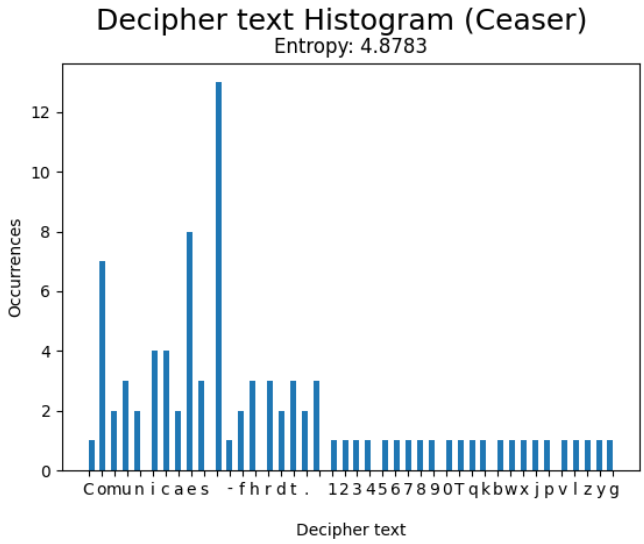


Figura 4 - Histograma do texto decifrado do ficheiro *a.txt* utilizando a cifra de César

Como é observado, os histogramas têm um formato idêntico, mas com caracteres diferentes. Por exemplo, o caractere 'C' no texto em claro tem o mesmo número de ocorrências que o caractere 'G' no texto cifrado, porque 'G' é o caractere resultante da cifra de 'C'. O histograma do texto decifrado é igual ao do texto em claro, como era de esperar.

Após a análise dos ficheiros produzidos pelas duas funções, conclui-se que o cifrador e o decifrador estão a funcionar corretamente.

**Resultados do ficheiro *alice29.txt* com *shift = 10*:**

- Entropia do texto em claro: H(X) = 4.5129 bit/símbolo;
- Entropia do texto cifrado: H(Y) = 4.5128 bit/símbolo;
- Entropia do texto decifrado: 4.5128 bit/símbolo;

Como é possível constatar, os valores da entropia também são muito próximos, havendo apenas a diferença de uma décima milésima, possivelmente devido às dimensões do ficheiro. Os histogramas obtidos para este ficheiro não são aqui apresentados por não terem qualidade, visto que o ficheiro original é muito extenso.

Após a análise dos ficheiros produzidos pelas duas funções, conclui-se que o cifrador e o decifrador estão a funcionar corretamente.

### 2.1.2 Cifra de Vernam (*One Time Pad*)

Começou-se por desenvolver as funções *vernam_cipher* e *vernam_decipher*, que realizam a cifra e a decifra, respetivamente, de um ficheiro de entrada, com recurso à Cifra de Vernam (*One Time Pad*).

A Cifra de Vernam é uma técnica de criptografia que não pode ser quebrada se utilizada corretamente. Consiste num algoritmo em que o texto em claro é combinado, caracter por caracter, a uma chave secreta aleatória que deve ter, no mínimo, o mesmo número de caracteres do texto em claro.

Ambas as funções recebem os seguintes parâmetros:

- o ficheiro de entrada, com o texto em claro no caso do cifrador, e com o texto cifrado no caso do decifrador;
- o ficheiro de saída, onde o resultado da cifra/decifra é escrito;
- uma chave, de preferência da mesma dimensão que o texto em claro.

Nesta cifra, é realizado uma operação de XOR entre o código de cada caracter do texto em claro e o código do caracter da chave associado ao mesmo, obtendo um novo código ASCII. Esta lógica, aplicada na função *vernam_cipher*, está apresentada na Listagem 3.

```
# Transverse the plain text
for i in range(len(plain_text)):
    char = plain_text[i]
    key_char = key[i % len(key)]

    char_code = ord(char) ^ ord(key_char)
    cipher_text += chr(char_code)
```

Listagem 3 – Cálculo do texto cifrado na função *vernam_cipher*

A mesma lógica também se aplica na função *vernam_decipher*, na Listagem 4.

```
# Transverse the cipher text
for i in range(len(cipher_text)):
    char = cipher_text[i]
    key_char = key[i % len(key)]

    char_code = ord(char) ^ ord(key_char)
    decipher_text += chr(char_code)
```

Listagem 4 - Cálculo do texto decifrado na função *vernam_decipher*

Aplicou-se esta cifra a dois ficheiros do conjunto *CD_TestFiles.zip*: *a.txt* e *alice29.txt*.

**Resultados do ficheiro *a.txt*:**

- Entropia do texto em claro: H(X) = 4.8783 bit/símbolo;
- Entropia do texto cifrado: H(Y) = 5.2834 bit/símbolo;
- Entropia do texto decifrado: 4.8783 bit/símbolo.

Como é possível constatar, os valores da entropia do texto em claro e do texto decifrado são iguais, comprovando que o sistema funciona corretamente. Observa-se também que a entropia do texto cifrado é maior, logo o algoritmo de cifra é eficiente.

Na Figura 5, na Figura 6 e na Figura 7 estão apresentados os histogramas do texto em claro, do texto cifrado e do texto decifrado, respetivamente.

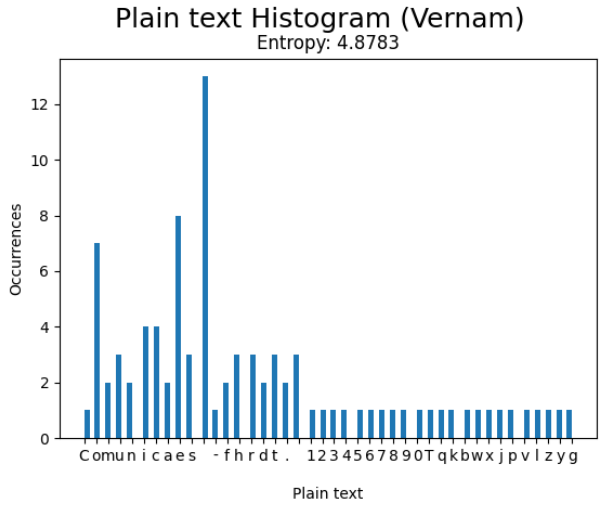


Figura 5 – Histograma do ficheiro *a.txt* utilizando a cifra de Vernam

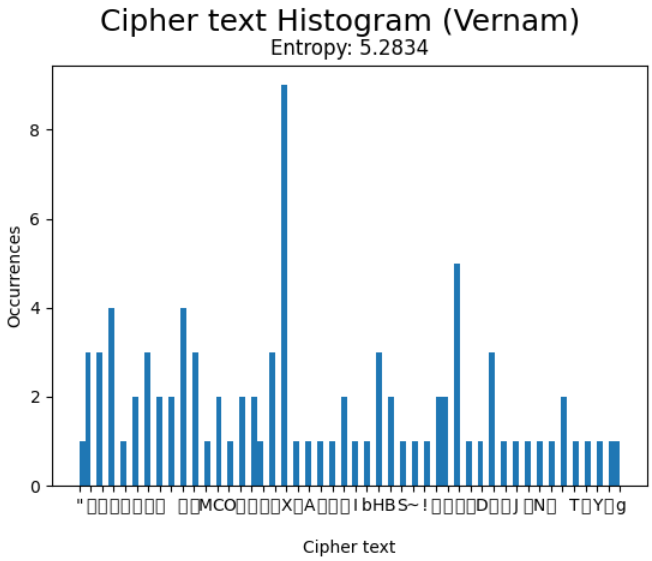


Figura 6 – Histograma do texto cifrado do ficheiro *a.txt* utilizando a cifra de Vernam

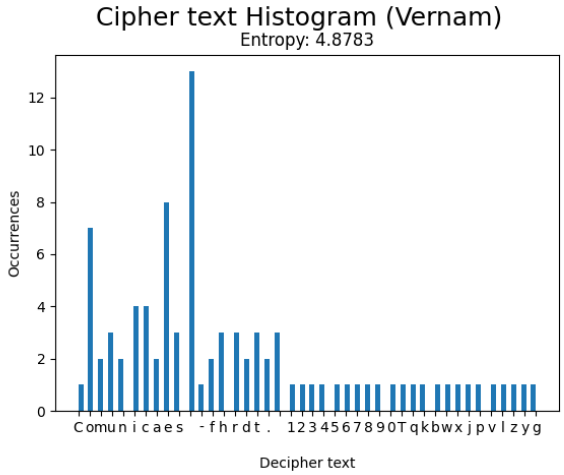


Figura 7 - Histograma do texto decifrado do ficheiro *a.txt* utilizando a cifra de Vernam

Como é observado, o histograma do texto cifrado contém caracteres não imprimíveis, visto que o código resultante da operação de XOR pode ser qualquer caractere da tabela ASCII. Esse histograma também tem uma maior distribuição das ocorrências pelos diferentes caracteres, tendo assim uma maior entropia. O histograma do texto decifrado é igual ao do texto em claro, como era de esperar.

Após a análise dos ficheiros produzidos pelas duas funções, conclui-se que o cifrador e o decifrador estão a funcionar corretamente.

**Resultados do ficheiro *alice29.txt*:**

- Entropia do texto em claro: H(X) = 4.5129 bit/símbolo;
- Entropia do texto cifrado: H(Y) = 6.0893 bit/símbolo;
- Entropia do texto decifrado: 4.5129 bit/símbolo;

Observa-se também que a entropia do texto cifrado é maior, logo o algoritmo de cifra é eficiente. Os histogramas obtidos para este ficheiro não são aqui apresentados por não terem qualidade, visto que o ficheiro original é muito extenso.

Após a análise dos ficheiros produzidos pelas duas funções, conclui-se que o cifrador e o decifrador estão a funcionar corretamente.

### 2.1.3 Cifra de imagens monocromáticas

Na alínea c, é pedido para implementar um cifrador/decifrador de imagens monocromáticas (em níveis de cinzento). Com esse objetivo, as funções *monochromatic_cipher* e *monochromatic_decipher* foram implementadas.

Ambas as funções recebem os seguintes parâmetros:

- o ficheiro de entrada, com o texto em claro (imagem original) no caso do cifrador, e com o texto cifrado (imagem cifrada) no caso do decifrador;
- o ficheiro de saída, onde o resultado da cifra/decifra é escrito;
- uma área retangular, que define que porção da imagem é cifrada/decifrada.

A função *monochromatic_cipher* calcula e retorna uma chave aleatória utilizada na cifra do ficheiro. Esta chave é passada como parâmetro à função *monochromatic_decipher*.

Neste algoritmo, itera-se por todos os pixels da imagem recebida, e se este estiver dentro da região retangular, é realizado uma operação de XOR entre esse pixel e a chave associado ao mesmo. Esta lógica está apresentada na Listagem 5.

```
for y in range(img_height):
    for x in range(img_width):
        if y in rect_height and x in rect_width:
            pixel = img[y][x]
            pixel_key = key[key_c := key_c + 1]

            ciphered_pixel = pixel ^ pixel_key
            img[y][x] = ciphered_pixel
```

Listagem 5 – Cálculo do texto cifrado na função *monochromatic_cipher*

A mesma lógica também se aplica na função *monochromatic_decipher*, na Listagem 6.

```
for y in range(img_height):
    for x in range(img_width):
        if y in rect_height and x in rect_width:
            pixel = img[y][x]
            pixel_key = key[key_c := key_c + 1]

            deciphered_pixel = pixel ^ pixel_key
            img[y][x] = deciphered_pixel
```

Listagem 6 - Cálculo do texto decifrado na função *monochromatic_decipher*

Aplicou-se esta cifra ao ficheiro *lena.bmp* do conjunto *CD_TestFiles.zip*.

**Resultados do ficheiro *lena.bmp*:**

Na Figura 8, na Figura 9 e na Figura 10 estão apresentados a imagem original (texto em claro), a imagem cifrada (texto cifrado) e a imagem decifrada (texto decifrado), respetivamente.

Figura 8 – Imagem original (texto em claro) do ficheiro *lena.bmp*

Figura 9 – Imagem cifrada (texto cifrado) após a aplicação do cifrador monocromático

Figura 10 – Imagem decifrada (texto decifrado) após a aplicação do decifrador monocromático

Ao cifrador foram passados como parâmetro dados de uma área retangular no centro da imagem original. Observando a imagem cifrada, comprova-se que o cifrador cifrou a imagem apenas na área selecionada. Como a imagem decifrada é igual à original, comprova-se que o sistema criptográfico funciona corretamente.

## 2.2 Exercício 2

O exercício 2 teve como principal objetivo o desenvolvimento de um programa que permite calcular o CRC de um ficheiro e verificá-lo posteriormente. É usada a classe CrcCalculator do módulo *crc* para este propósito.

A diretoria do exercício 2 está dividida em três subdiretorias:

- crc – contém as funções *crc_file_compute* e *crc_file_check*, da alínea 2a;
- test_crc – contém os testes e resultados experimentais da alínea 2b;
- commacode_crc – contém os ficheiros relacionados com a alínea 2c.

O método *crc_file_compute* escreve num ficheiro de output o CRC do ficheiro de input nos primeiros 4 bytes, com uma cópia do ficheiro original.

```
with open(input_file, "rb") as f:
    crc_calculator = CrcCalculator(config)
    data = f.read()
    checksum = crc_calculator.calculate_checksum(data)

with open(output_file, "wb") as f:
    f.write(checksum.to_bytes(length=CHECKSUM_LENGTH,
byteorder="little"))
    f.write(data)
```

Listagem 7 - Cálculo do CRC do ficheiro na função *crc_file_compute*

O método *crc_file_check* verifica que o CRC dado condiz com os dados presentes no ficheiro, retornando True se estiver correto.

```
with open(input_file, 'rb') as f:
    checksum = int.from_bytes(f.read(CHECKSUM_LENGTH),
byteorder="little")
    data = f.read()

crc_calculator = CrcCalculator(config)

return crc_calculator.verify_checksum(data, checksum)
```

Listagem 8 – Verificação do CRC na função *crc_file_check*

### 2.2.1 Resultados obtidos

Para testar se o CRC funciona corretamente quando os ficheiros transmitidos por um canal com erros, foram introduzidos *bit flips* nos ficheiros de teste. As probabilidades de *bit flip* são as seguintes: 0.01%, 0.1%, 0.5%, 1% e 5%.

Como é possível observar no ficheiro *test_crc_with_errors.log*, os únicos ficheiros onde o CRC não detetou erros foram nos ficheiros *alphabet.txt* e *a.txt* com probabilidades de *bit flip* baixas. Neste caso, devido às baixas probabilidades de erro, não foram produzidos erros nos ficheiros, ou seja, o CRC validou corretamente os ficheiros.

### 2.2.2 Integração com o código unário

A integração do CRC com o par codificador/descodificador *comma code* desenvolvido na fase 2 foi efetuada com a geração de uma *shared library* (.dll) do programa em C.

O CRC é calculado a partir do ficheiro original, e inserido no início do ficheiro já codificado por *comma code*. Após a descodificação, é feita a validação do ficheiro descodificado com o checksum CRC.

Para validar o funcionamento da integração, foram adicionados os mesmos erros da alínea anterior nos ficheiros de teste. Os resultados encontram-se no ficheiro *test_commacode_crc_with_errors.log.* Como é possível observar nos *logs*, o CRC consegue encontrar todos os ficheiros com erro, tal como na alínea anterior, contudo, caso haja demasiados erros, não é sequer possível descodificar o ficheiro.

## 2.3 Exercício 3

O objetivo do exercício 3 é desenvolver dois sistemas de comunicação digital, onde a principal diferença é a implementação do emissor, sendo este um emissor NRZU ou PSK.

A diretoria do exercício 3 está dividida em duas subdiretorias:

- NRZU – contém a implementação com NRZU;
- PSK – contém a implementação com PSK.

### 2.3.1 NRZ-Unipolar

A implementação do emissor NRZU (Non-Return-To-Zero Unipolar) consiste em repetir os bits de entrada N vezes, onde N é o número de amostras. O recetor deste sinal calcula a média das N amostras, e, se a média for inferior a metade da amplitude predefinida, o bit é assumido como 0, caso contrário é assumido como 1.

Na Figura 11 está apresentado o sinal NRZU com a codificação da sequência binária 10110001. Este sinal corresponde aos sinais x(t) e y(t), que são iguais, visto que o canal é perfeito.

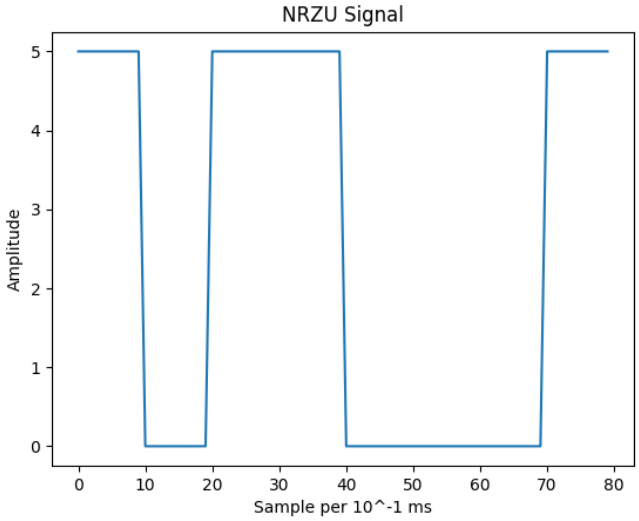


Figura 11 – Sinal da sequência binária 10110001 codificado com NRZU (sinais x(t) e y(t))

### 2.3.2 Phase-shift Keying

No modulador de PSK (Phase-Shift Keying) foi gerado para cada bit 1, um sinal $-2\,cos(2\pi 2000t)$ e para cada bit 0, um sinal $2\,cos(2\pi 2000t)$. Para cada bit foram retiradas 10 amostras destes sinais.

Na Figura 12 está apresentado o sinal NRZU com a codificação da sequência binária 10110001. Este sinal corresponde aos sinais x(t) e y(t), que são iguais, visto que o canal é perfeito.

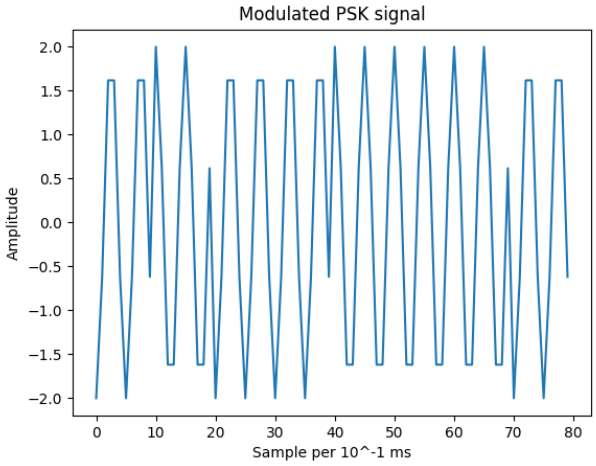


Figura 12 – Sinal da sequência binária 10110001 modulado com PSK (sinais x(t) e y(t))

No recetor, para cada tempo de bit, multiplica-se o sinal por um sinal referência de bit 1. Se a média do sinal produzido for maior que 0, então o bit é 1, caso contrário assume-se que é 0.

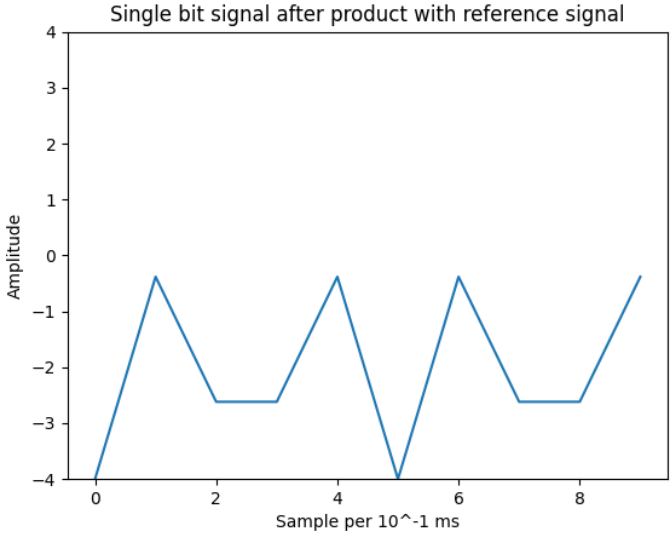


Figura 13 – Representação do sinal correspondente ao produto do sinal origem (de um bit 0) com o sinal de referência de bit 1

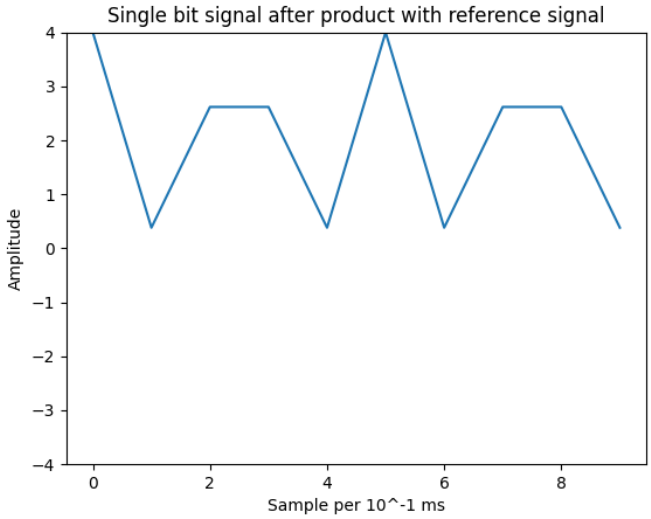


Figura 14 – Representação do sinal correspondente ao produto do sinal origem (de um bit 1) com o sinal de referência de bit 1

### 2.3.3 Alínea 2 b)

A alínea b) tem como objetivo testar os sistemas de comunicação digital na presença de ruído no canal de comunicação.

Para o emissor NRZU, quando a = 1 e n(t) é uma distribuição normal com desvio padrão 2, BER já não se encontra a 0. Isto acontece devido ao NRZU considerar um bit a 1, quando a média das amostras desse tempo de bit for maior que metade da amplitude. Quando o desvio padrão é maior que 2, a probabilidade de um bit 0 sofrer ruído de forma à média das amplitudes das suas amostras ser maior que 2.5 (amplitude / 2) é alta.

Para o emissor PSK, quando a = 1 e o desvio padrão de n(t) é maior ou igual a 2, o BER já não se encontra a 0. Isto deve-se à baixa quantidade de amostras por cada bit.

Para o emissor NRZU, quando a < 0.5 e n(t) constante, os bits 1 não serão detetados pelo recetor, porque a amplitude dos bits a 1 do sinal y(t) é < 2.5 (metade da amplitude original).

Para o emissor PSK, quando a < 1 e n(t) constante, o BER é 0, ou seja, PSK é resistente à atenuação do sinal.

Como é possível observar, a nossa implementação de PSK é mais suscetível a ruído no seu sinal (desvio padrão alto), enquanto a NRZU é mais suscetível à atenuação do seu sinal (a < 1).

Os valores de BER encontram-se no ficheiro *file_processing.log*.

## 2.4 Exercício 4

O exercício 4 teve como objetivo a demonstração de uma conexão serial em modo *simplex* entre dois intervenientes, PC e Arduino.

O Arduino envia dados que serão lidos pelo PC, através de um programa em Python, com recurso ao módulo *pyserial*. O código de Arduino foi desenvolvido no IDE oficial da plataforma, Arduino IDE (https://www.arduino.cc/en/software).

Na Figura 15 está apresentado o diagrama de blocos da conexão realizada.

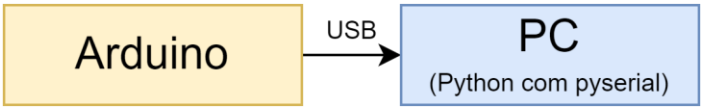


Figura 15 – Diagrama de blocos da conexão entre o Arduino e o PC

A diretoria do exercício 4 contém o ficheiro *serial_arduino.py*, que contem a função *receive_from_arduino*, que, como o nome indica, recebe informação de uma porta serial, através do módulo *pyserial*. Na Listagem 7 está apresentado o código correspondente a esta função.

```
def receive_from_arduino(port):
    """
    Receives and prints data from the Arduino, using the serial
port.

    :param port: The serial port to use.
    """

    ser = serial.Serial(port, baudrate=9600)
    while True:
        read_value = ser.readline()

        print(read_value)
```

Listagem 7 – Código da função *receive_from_arduino*

A mesma diretoria contém a subdiretoria *arduino*, com o ficheiro *serial_hello_world.ino*, contendo o código Arduino implementado. Este ficheiro tem duas funções, uma que realiza o setup da porta serial e outro que realiza um ciclo, escrevendo a mensagem "Hello World" na serial port. Na diretoria deste exercício também se o ficheiro *arduino_experimental_result.mp4*, que contém uma demonstração do SCD em funcionamento.

### 2.4.1 Características do SCD

O SCD implementado tem topologia ponto-a-ponto, tendo apenas dois nós, o dispositivo Arduino e o PC. A direção da transmissão é *simplex*, via conexão USB, no sentido Arduino -> PC. O tipo de sinal transmitido é digital, ou seja, existiu transferência de bits entre os equipamentos, que codificam as mensagens enviadas.

# 3 Conclusão

Em suma, podemos concluir que os objetivos definidos para a realização deste trabalho foram atingidos, e consideramos que as resoluções dos exercícios apresentadas, são adequadas e cumprem com os requisitos do enunciado

Durante a implementação, adquirimos conhecimentos relativos aos sistemas de comunicação digital (SCD), a cifra e às técnicas de codificação de canal. Este trabalho também nos possibilitou o desenvolvimento de competências na utilização das linguagens “C” e “Python” na construção de programas de baixa/média complexidade.

Esta série de problemas proporcionou-nos a oportunidade de utilizar conhecimentos que viemos a adquirir nas aulas da unidade curricular, e durante o nosso estudo autónomo.

# 4 Software utilizado

Segue-se uma lista do software utilizado na realização deste trabalho, juntamente com uma breve descrição sobre a sua utilização:

- **Python 3.10**: linguagem utilizada no desenvolvimento dos exercícios;
- **PyCharm 2021.3.3 (Professional Edition)**: ambiente de desenvolvimento de programas em Python;
- **Git/GitHub**: controlo de versões e armazenamento do projeto num repositório;
- **Microsoft Word:** escrita do presente documento.

# 5 Referências


[1] Slides 8.Sistemas Criptograficos (2 de maio de 2022) Retrieved May 25, 2022, from https://2122moodle.isel.pt/pluginfile.php/1158192/mod_resource/content/3/8.Sistemas%20Criptograficos.pdf

[2] Wikimedia Foundation. (2022, April 28). Caesar cipher. Wikipedia. Retrieved May 25, 2022, from https://en.wikipedia.org/wiki/Caesar_cipher

[3] Wikimedia Foundation. (2022, May 25). One-time pad. Wikipedia. Retrieved May 25, 2022, from https://en.wikipedia.org/wiki/One-time_pad

[4] Slides 9.Codificação de Canal (26 de abril de 2022). Retrieved April 28, 2022, from https://2122moodle.isel.pt/pluginfile.php/1158193/mod_resource/content/1/9.Codifica%C3%A7%C3%A3o%20de%20Canal.pdf

[5] Slides 10.Banda Base - Códigos de Linha (19 de maio de 2022). Retrieved May 10, 2022, from https://2122moodle.isel.pt/pluginfile.php/1160160/mod_resource/content/1/10.Banda%20Base%20-%20C%C3%B3digos%20de%20Linha.pdf

[6] Slides 11. Sinais - Classificacao e Propriedades (23 de maio de 2022). Retrieved May 16, 2022, from https://2122moodle.isel.pt/pluginfile.php/1160464/mod_resource/content/1/11.Sinais%20-%20Classificacao%20e%20Propriedades.pdf

[7] Slides 12.Sistemas - Operacoes (24 de maio de 2022). Retrieved May 16, 2022, from https://2122moodle.isel.pt/pluginfile.php/1160466/mod_resource/content/1/12.Sistemas%20-%20Operacoes.pdf