COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE CARRERA

Forward Error Correction (FEC) scheme for a fishing radio-buoy system

Autor: Jaime Masjuan Ginel

Director: Javier de Salas

Madrid

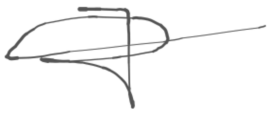
Julio 2022

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Forward Error Correction (FEC) scheme for a fishing radio-buoy system

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2021/2022 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Jaime Masjuan Ginel Fecha: 07/ 07/ 2022



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Javier de Salas Lasagabáster Fecha: 07/ 07/ 2022

COMILLAS

UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE CARRERA

Forward Error Correction (FEC) scheme for a fishing radio-buoy system

Autor: Jaime Masjuan Ginel

Director: Javier de Salas

Madrid

Julio 2022

SISTEMA DE CORRECCIÓN DE ERRORES (FEC) PARA UNA RADIO-BOYA DE PESCA

Autor: Masjuan Ginel, Jaime

Director: de Salas, Javier

Entidad colaboradora: Equipos Navales Industriales SA, Enisa

Resumen

Introducción. Es innegable el efecto que, de un tiempo a esta parte, está teniendo la pesca masiva para nuestros océanos. Esto se debe a que una gran parte de todos los peces capturados que inicialmente van destinados al consumo humano se ve desperdiciada por dos principales motivos: no hay espacio suficiente para almacenar semejantes cantidades, o simplemente no termina por consumirse. Todo ello ha llevado a que en las últimas décadas hayan aparecido técnicas cuyo fin es realizar la pesca de una manera eficiente que, al mismo tiempo, cumpla con las demandas de la población y respete el medioambiente. Una de estas técnicas es la pesca mediante palangres. Esta técnica consiste en el uso de un sedal kilométrico, plagado de anzuelos, que va a la deriva por el mar. Para poder llevar a cabo la recogida de la pesca, es necesario tener un sistema radio-boya que permita obtener la posición de dicho palangre desde un barco.

El Proyecto. El objetivo principal que persigue este Proyecto de Fin de Grado no es otro que el de aumentar el rendimiento de un sistema radio-boya, como el previamente descrito, diseñado y fabricado por la compañía Enisa. Para ello, se estudiarán diferentes códigos basados en técnicas de corrección de errores (FEC). Al tratar de aumentar dicho rendimiento, los resultados que se esperan conseguir son variados. Uno de ellos es lograr una mejora en la respuesta frente a diferentes entornos

donde la posibilidad de que se produzcan errores en la transmisión de los datos sea diferente. Del mismo modo, se pretende también conseguir un aumento en la cantidad de información enviada por transmisión y en el alcance que esta pueda llegar a tener.

Descripción del algoritmo. Debido a las limitaciones que presenta el microchip PIC16F18326 empleado en el sistema de radio-boya, se van a diseñar y estudiar unos códigos de corrección de errores para la transimisión y recepción de una cadena similares a los códigos Hamming. Como el microchip utiliza bytes de ocho bits, los tres casos a estudiar utilizarán todos los bits disponibles para cada uno de los bytes transmitidos. Hay dos tipos de bits posibles: los que contienen el mensaje que se pretende transmitir y aquellos conocidos como bits de paridad [1], cuya misión es la de ayudar a poder recuperar la información de manera fiable y con buen resultado. La tasa de información enviada, R, se basa en el número de bits que contienen el mensaje, k, y el número de bits total en un byte, n. Así:

$$R = \frac{k}{n} \tag{1}$$

Actualmente este valor se encuentra en un 20%, esto es, hay un 80% de la transmisión destinada a la recuperación del mensaje exclusivamente. Para poder aumentar estas cifras, se ha decidido estudiar tres códigos diferentes. El primero, caso (2, 8), utiliza dos bits por cada byte como mensaje ($R = 25\%$), el segundo, caso (3, 8), utiliza tres bits por cada byte como mensaje ($R = 37.5\%$) y, finalmente, el último caso (4, 8), ha utilizado cuatro bits por cada byte como mensaje ($R = 50\%$). Para llevar a cabo la codificación en cada caso, se ha seleccionado un conjunto de bytes tales que maximicen la mínima distancia Hamming entre ellos. La distancia Hamming [2] es el número de bits en los que difieren dos bytes distintos. Estos grupos de bytes serán asignados a cada uno de los posibles mensajes que se pueden enviar, es decir, en el caso (2, 8) habrá cuatro posibles mensajes a enviar, en el (3, 8) habrá ocho y en el (4, 8) habrá dieciséis. En el transmisor se dividirá cada byte de la cadena según el caso empleado, y a cada una de las partes resultantes se le asignará su correspondiente código antes de transmitir la información. Ya en el receptor, el algoritmo empleado se ayu-

dará de una matriz H cuya finalidad es la de poder detectar y ubicar los errores que puedan aparecer durante el proceso de transmisión. Este proceso se lleva a cabo mediante la multiplicación de dicha matrix con el byte. Si el resultado de dicha operación es cero no se habría producido ningún error, pero si por el contrario sí que hubiese uno o más errores, el resultado identificaría inequívocamente cuales son los bits erróneos hasta el número de bits que cada algoritmo puede corregir. Así, solo habría que cambiar el valor de dichos bits y la información estaría recuperada.

Simulación y resultados. Para poder llevar a cabo una simulación y una comparación fiable de los nuevos algoritmos con el antiguo, se ha creado un código en Python que simula con fidelidad la transmisión y recepción de la información enviada por la boya. El entorno consta de tres funciones, una que simula la transmisión, otra la recepción y una tercera que modela una inclusión de bits de manera aleatoria en base a una probabilidad previamente establecida conocida como BER (Bit Error Rate). Los resultados obtenidos mostraron como se obtenía una gran mejora del rendimiento del algoritmo en comparación a las simulaciones del código previo, y la implementación cumplía con las especificaciones de tiempo requeridas para que pudiera aplicarse en un entorno real en los microcontroladores que se utilizan en el hardware actual.

Conclusiones. Tras todo el estudio llevado a cabo, se concluye que el algoritmo está listo para ser empleado en pruebas marítimas y comprobar su rendimiento en una situación real. Como trabajo futuro se abre la puerta a la implementación y estudio de técnicas más sofisticadas en el caso de que se empleara otro microprocesador con una capacidad mayor para operaciones complejas.

Referencias.

[1] Mary K. Wootters, Any errors in this dissertation are probably fixable: topics in probability and error correcting codes. PhD Thesis, Department of Mathematics, University of Michigan 2014.

[2] Richard W. Hamming, Error Detecting and Error Correcting Codes. The Bell System Technical Journal, Vol. 29, April 1950.

FORWARD ERROR CORRECTION (FEC) SCHEME FOR A FISHING RADIO-BUOY SYSTEM

Author: Masjuan Ginel, Jaime

Supervisor: de Salas, Javier

Collaborating Entity: Equipos Navales Industriales SA, Enisa

Abstract

Introduction. It is undeniable the effect that, for some time now, massive fishing is having on our oceans. This is due to the fact that a big part of all the fishes captured which are initially destined for human consumption is wasted for two main reasons: there is not enough space to store these huge amounts, or just because nobody finally consumes it. All of this has encouraged the development of new techniques whose primary goal is to create a more efficient way of fishing such that it can meet the demands of our society while being environmentally friendly. One of these techniques is called longline fishing. This technique consists in the use of a long line several kilometers in length, riddled with fishhooks and drifting through the sea. In order to being able to locate the longline, it is necessary to have a radio-buoy system that allows to obtain its position.

The Project. The main goal pursued by this Final Degree Project is no other than to increase the performance of a radio-buoy system, similar to the one previously described, designed and manufactured by Enisa. To this effect, a study will be carried out about different forward error correction (FEC) techniques and codes. Whilst trying to increase the performance, the results expected are diverse. One of them is to obtain an improvement in the response against different environments where the probability of errors during the transmission varies. Simi-

larly, this project also aims to obtain a increase in the information sent per transmission and in the distance which it may travel.

Description of the algorithm. Due to the limitations of the microchip PIC16F18326 used for the radio-buoy system, the FEC codes which are going to be studied and designed are very similar to the Hamming codes. As the microchip uses eight bit bytes, the three cases that will be studied will use all the bits available for each of the bytes transmitted. There are two kinds of bits: the ones that compose the message which has to be transmitted and the ones known as parity bits [1], which have the purpose of helping the receiver to recover the information in a reliable way. The code rate, R, is based on the number of bits that has the message, k, and the total number of bits per byte, n. Thus:

$$R = \frac{k}{n} \tag{2}$$

Currently, this value is 20%, this means that 80% of the transmission is destined exclusively to recovering the message. In order to increase this value, the project will be focused on the study of three different error correction cases. First, case (2, 8), it uses two bits per byte as the message ($R = 25\%$), second, case(3, 8), uses three bits per byte as the message ($R = 37.5\%$), and, finally, case (4, 8), which uses four bits per byte as the message ($R = 50\%$). To carry out the encoding in every case, a set of bytes, known as codewords, was selected with the criterion of maximizing the minimum Hamming distance between them. The Hamming distance [2] is the number of bits that differ from one byte to another. This sets will be assigned to each of the possible message that can be sent, that is to say, in case (2, 8) there will be four possible messages to transmit, in case (3, 8) there will be eight and in case (4, 8) there will be sixteen. In the transmitter each byte of the string will be divided depending on the case used and each of the resultant parts will be assigned to its corresponding codeword before transmitting the information. At the receiver, the algorithm will use the so-called parity matrix H which has the primary mission of detecting and finding errors that may appear during the transmission process. This process will happen by multiplying the parity matrix and the codeword re-

ceived. If the result of the operation is zero there would be no error, but if one or more errors happened, the resulting parity value would uniquely identify which bits are in error up to the number of bits that each algorithm can correct. Flipping precisely those identified bits would recover the correct information.

Simulation and results. To ensure that the simulation and comparison between the current algorithm and the ones developed in this project is carried out with reliability, a Python simulation environment has been developed in order to represent with fidelity the transmission and reception of the information sent by the buoy. This environment has three main functions, one that simulates the transmission, one that simulates how the receiver works and another one that randomly models and includes error bits in the information using different BERs (Bit Error Rate). The results obtained showed a significant improvement of the performance in comparison with the simulations of the previous algorithm. Also the implementation met with the specifications of time needed to be able to use this algorithm in a real-life environment using the actual microcontrollers used in the hardware.

Conclusions After all the study carried out, the conclusion is that the algorithm is ready to be used in sea trials to ensure its performance in a real-life situation. As future work, there is room for the implementation and study of more sophisticated techniques in case that another microprocessor with capacity for achieving more complex operations will be used.

References.

[1] Mary K. Wootters, Any errors in this dissertation are probably fixable: topics in probability and error correcting codes. PhD Thesis, Department of Mathematics, University of Michigan 2014.

[2] Richard W. Hamming, Error Detecting and Error Correcting Codes. The Bell System Technical Journal, Vol. 29, April 1950.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

Radio buoys are a fishing aid often used to locate and track longlines, drift nets and any other fishing gear that is set to drift in the ocean. Radio buoys are equipped with a GNSS (Global Navigation Satellite System) receiver and a radio transmitter that allows them to broadcast their location as well as other sensor data such as depth or water temperature. Fishing gear is set adrift and recovered after a few days. Hopefully, full of catch. Depending on winds and currents, drifting fishing gear may travel long distances and hence may be difficult to locate.

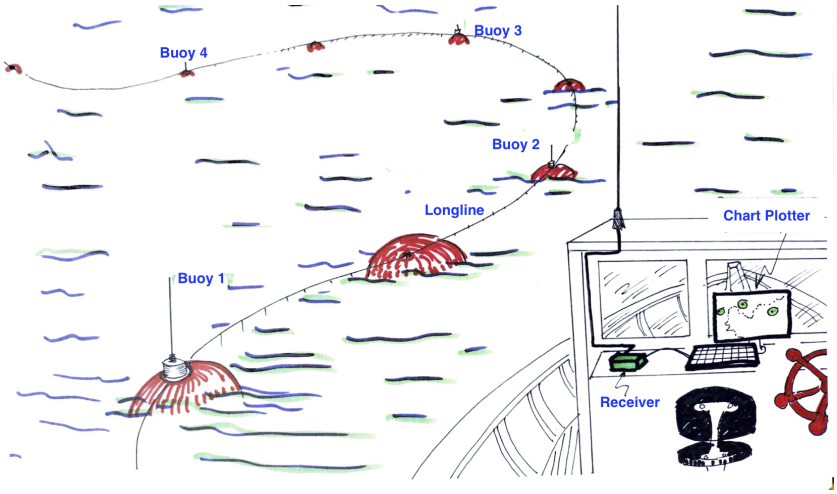


Figure 1: Radio buoys used by a longliner vessel

On board the fishing boat, there is a radio receiver with a modem that decodes the signals coming from the buoys and presents them on the screen of a chart plotter so that the skipper can easily find them and navigate towards them. Figure 1 shows a drawing of a fishing vessel and a few radio buoys set on a long line.

It is of interest for the skipper to be able to locate the buoys from far away, that way, he or she can decide which buoys to navigate to and thus

save time and fuel. Distances of 20 nautical miles or more are desirable but radio regulations limit the buoy's transmitter power to 4 Watts [3] and this presents an interesting trade off because the received signals onboard the vessel are very weak. Weak signals mean bit errors occur making it impossible to render the buoys at their location. It is for this reason that it is desirable to come up with a Forward Error Correction (FEC) scheme that allows the recovery of the message in spite of the bit errors. Note the communication is unidirectional from the buoys to the modem thus making it impossible for the receiver to let the transmitter know that a packet has been received with errors.

1.2 Motivation

During the second half of 2021 I had the privilege of taking an internship in the company Equipos Navales Industriales S.A., Enisa. I found myself working on a thrilling project that could make an important impact in the fishing industry. As the need of a more a sustainable way of fishing has lately appeared, new techniques are gaining strength. One of the reasons I chose to study an engineering degree was to be able to make a positive impact on the environment while developing new technologies and as I have always been a person close to the sea and its particular issues, helping to develop a longliner fishing system project in order to reduce the problems this industry is creating was everything I could ask for.

After six months working in Enisa and while I was approaching the end of the internship, the posibility of helping the company the next months with a long-term project appeared. Thus, I was presented with the opportunity of trying to improve the current communication system while adapting complex communication methodologies to the limited resources available in order to increase the efficiency of the present model.

1.3 Resources Used

This project will use several tools in order to develop correctly the three phases mentioned in the Working Metholodogy section. This tools will consist in the use of four main programs: Visual Studio Code, MP Lab, Github as a software repository and Github Issues as a bug-tracking system.

1.3.1 Visual Studio Code

Firstly, Visual Studio Code is a source-code editor developed by Micrososft that is able to work with several programming languages such as Python, LaTex or C/C++ among others [4]. The utilisation of VS Code will allow the proper simulations in Python. The main Python libraries which will be used are numpy and matplotlib. As mentioned before, these modules are going to facilitate the use of arrays and 2D or 3D graphics. Thus, comparison

between different sets of codewords will be performed. Figure 2 shows the VS Code workspace.

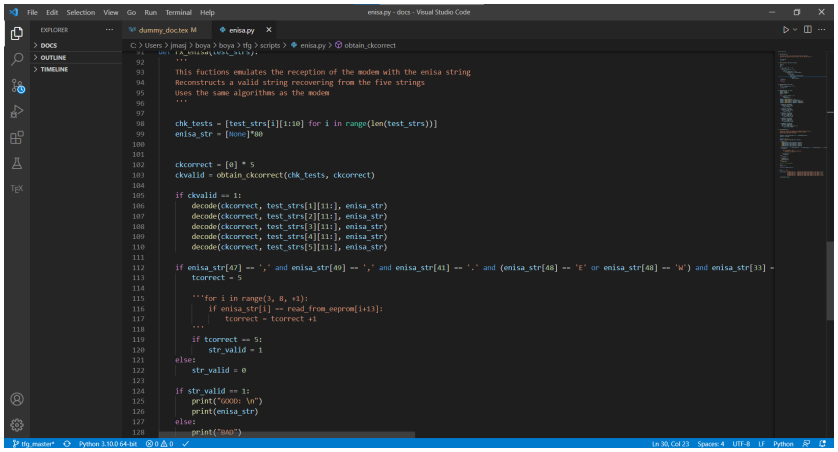


Figure 2: Visual Studio Code workspace

1.3.2 MPLAB X IDE

Secondly, MPLAB is an integrated development environment (IDE) developed by Microchip for the implementation and development of applications on the PIC microcontollers [5]. The purpose of this software in this project will be the debugging and programming of the 8-bit PIC microcontroller. Once the best set of codewords is selected MPLAB will be used for the implementation in the product with Microchip's PicKit 4. MPLAB will be used to develop C programs. The version utilised for this project is MPLAB X. Figure 3 shows a picture of the MP Lab X workspace.

1.3.3 Github

The last tool which will be used is the GitHub repository. Git is a very powerful version control software used by GitHub, the most popular code repository in the world with more than 73 million developers. It will facilitate a coordinated programming and a more efficient working methodology.

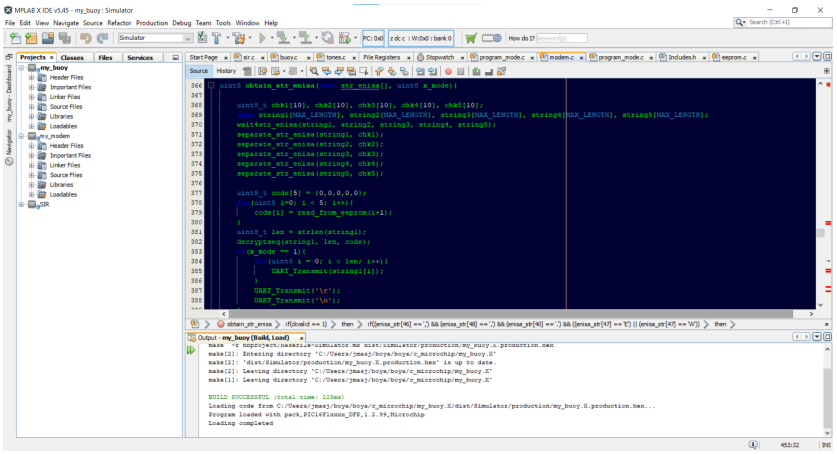


Figure 3: MPLab workspace

GitHub Issues will be used as our bug-tracking system to increase the quality of the developed software and to minimize the bugs. Github is going to be used in all the three different phases described in the Working Methodology section 3. A screenshot of the web interface for Github's bug tracking system ("Issues") is shown in Figure 4.

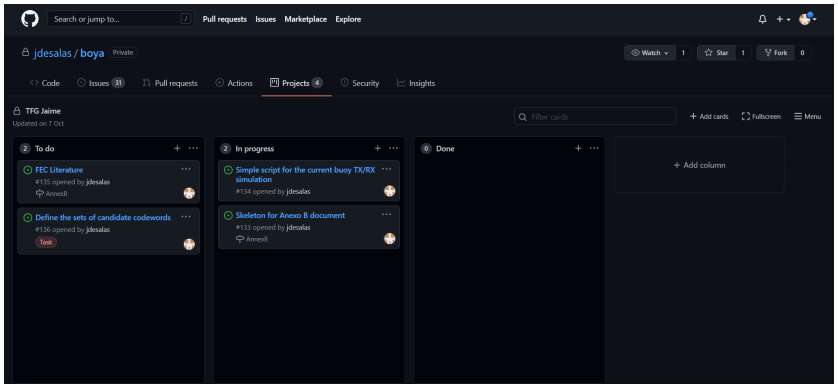


Figure 4: GitHub workspace

1.4 Project Objectives

The main objective of this project is to study different Forward Error Correction codes that improve the performance of the existing scheme. The key performance metric is the probability of recovering the message for a

given Bit Error Rate. The end goal is of course to implement this scheme in the buoy's and the modem's 8-bit microcontroller. There are two specific improvements that we want to achieve with the improved FEC method.

1.4.1 Bandwidth Efficiency

One of the metrics for the new FEC scheme is the code rate defined above. Code rate R is the ratio of the message size over the total size including the error correction bytes. If the code rate can be increased, it means that fewer bytes will be sent over the channel. Fewer bytes over the same bandwidth means each buoy will report its location and sensor information faster. The buoys broadcast in the 27 MHz band, so called Citizen's Band or CB. CB is a license free band with a limited power of 4 Watts [3] so it is generally quite crowded. The radio buoys use a TDMA (Time Division Multiple Access) taking advantage that each buoy has a GNSS receiver and therefore good knowledge of a common time base. The common time base is used to split the time interval into slots that are assigned to each buoy so that they do not interfere with each other. If the transmissions are faster, the number of slots can be increased and thus the number of buoys per channel.

1.4.2 Longer Communication Range

The second aspect of an improved FEC scheme is the improved bit error rate (BER). Better BER means better reception when the received signal is weak due to path loss propagation over longer distances. Being able to see the buoys on the chart plotter screen at a greater distance leads to time and fuel savings. This is of course highly appreciated by the fishermen and remains one of the most important goals.

1.5 Project Timeline

Firstly, a study about the existing method was undertaken, in order to understand how the current radio-buoy system works. Secondly, the project's goals were defined. Thus, the next step was to create a simulation environment that represented a real-life situation. After creating the simulation

environment, the new possible methods were defined and simulated in order to obtain results that could define which was the best candidate. The step that followed after selecting the winner, was to implement the new software into the radio-buoy system.

This is the Final's Degree Project timeline:

Sep. 30st	Anexo A
Nov. 30th	Literature review
Dec. 31st	Study of existing FEC
Jan. 31st	Goals and Methodology
Jan. 31st	Milestone: Anexo B
Feb. 11th	Python migration of existing FEC
March 4th	Simulation environment
March 18th	Selection of new set of codewords sets
April 15th	Coding of new FEC
April 22nd	Run simulations of candidate FECs
April 22nd	Milestone: FEC selection
May 6th	Coding of TX
May 27th	Coding of RX
May 27th	Milestone: C implementation
June 4th	C and Python cross check
June 11th	Result compilation and conclusions
June 31st	Write TFG
June 31st	Milestone: TFG and presentation ready

2 State of the Art

2.1 Basic concepts of Forward Error Correction

Forward error correction (FEC) is a technique for controlling the number of errors transmited in a message without forcing the message to be re-transmitted. Essentially, the sender needs to transmit a message to the receiver over a medium that is likely to introduce errors. The goal of FEC is for the receiver to recover the same exact message transmitted in spite of the errors that the noisy medium (or channel) introduces. This is achieved by adding some redundancy bits. Error Correction codes is a very complex topic with extensive research since the 1950's. We will only touch the surface of it, since in our application, we need to find a trade-off between the ability to detect and correct errrors and the complexity of the algorithm. Both the radio buoy and the modem use 8-bit microcontollers running at 4 MHz and 20 MHz respectively so the computational resources are very limited.

We will start adding here some common notation [1] that is widely used in error correction literature and that we will use through the full paper. The sender A needs to transmit a message x of length k to the receiver B. To that end, A uses a codeword $c = C(x)$ of length n where C is a function that maps the field of binary words of length k or with a subset C of the field of binary words of length n. More formally,

$$x \in F^k$$
$$c \in C \text{ and } C \subset F^n$$

Where F^k and F^n are finite fields of order 2^k and 2^n. The number of possible messages x and codewords is then 2^k because there is a one to one correspondence between each message and its codeword. We can represent the basic setup [1] in Figure 5.

The number n is greater than k. This brings us to the definition of the code rate R

$$R := k/n \tag{3}$$

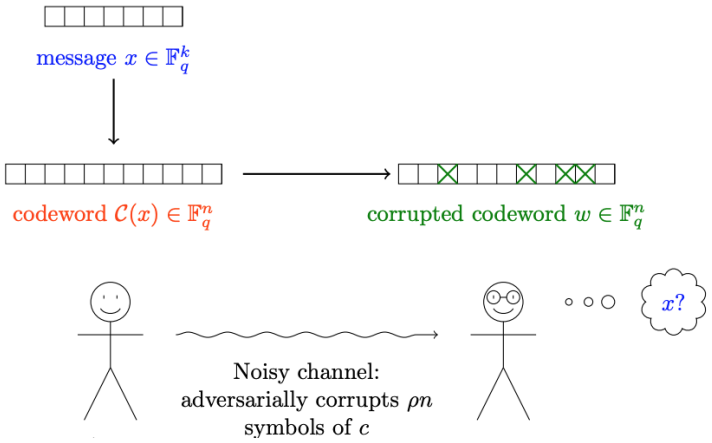


Figure 5: Setup for Error Correction Codes

The code rate R is a number between zero and one and measures the amount of redundancy bits that are sent to the receiver.The greater the value of R the better but there is an obvious trade-off with the ability to recover errors.

Another useful definition is the Hamming distance between two code words, which is the number of bits that differ between two codewords c and c'.

$$\delta(c, c') := \sum_{i=1}^{n} 1_{c_i \neq c'_i} \tag{4}$$

The distance of a set of codewords C is then

$$\delta(C) := min\, \delta(c, c') \; \forall c, c' \in C \tag{5}$$

We will use the Hamming distance to help us pick the set of Codewords to be used.

2.2 Common FEC Methods

The most common error correction codes are Hamming Codes, Binary Convolution Codes, Reed-Solomon Codes and the Low-Density Parity Check Codes.

The Hamming Code is a technique developed by R. W. Hamming [2] that uses a set of parity binary bits to guarantee that the number of 1's in the data is odd or even. There are two types of parity bits: even parity bit or odd parity bit. The value of the parity bit depends on the sum of the amount of 1's for a given set of bits. If that count is even, then the parity bit is a 1. On the other hand, if the count of the bits happens to be odd its value will be a 0 and it shall be called an odd parity bit. The parity bits are located in positions that are the power of 2. In order to find the position of the wrong bit, the sum of the wrong parity bits needs to take place, allowing the code to find out where the mistake has been made.

The codewords for this project will have a length of 8 bits, since the microchip used is an 8 bit microcontroller. In order to increase the code rate, the cases studied will include messages with 2, 3, or 4 bits. For each case the Hamming distance will be maximized with a view to minimize the error rate.

The Binary Convolution Code is another type of Forward Error Correction introduced in 1955 by P.Elias [6]. It is based on the use of boolean functions to data streams. The application of boolean functions allows the FEC to create parity simbols. They are distinguished by the code rate and the memory, which is an output based on the previous inputs that the system has received. This codes are mainly used for radio, digital video, and satellite communications.

Yet another FEC method is the Reed-Solomon Code, which was developed by Irving Reed and Gustave Solomon in 1960 [7]. It is a non-binary cycled code that uses a polynomial function which is associated with a word. This word will need to have a minimun of N correct symbols that will allow the code to reconstruct the information.

The Low-Density Parity Check Code is a type of linear error correcting

code that was developed by Robert G. Gallager in 1962 [8]. This type of FEC permits to take the noise threshold up to the maximum allowed. This noise threshold sets the limit for the probablity of lost information to be as reduced as needed. Despite being developed in 1962, this technique could not be used until 1996 due to technology limitations.

This project will be mainly focused on the develop of a FEC similar to the Hamming codes due to the limitations that the 8 bit microchip has, restricting any type of polinomial division or matrix multiplication. Using the concept of the Hamming distance, the focus will be put on the selection of the best set of codewords given k and n, in order to maximize the distance between one word to another. A more detailed description can be found in section 3.

2.3 Existing FEC Mechanism

In this section the existing FEC method that is used today will be described since this is the system that we need to improve.

The error correction system currently implemented consists of sending the same string five consecutive times from the buoy to the modem. These strings are further divided into five sub-strings and a checksum is calculated by the transmitter for each of the five sub-strings.The checksum is simply one byte with the binary XOR of all the bytes in the sub-string. The transmitted checksums form a 5x5 matrix that is used to guarantee that the each of the received substring is error free. Naturally, there could also be an error in the transmitted checksums so the modem compares the rows of the 5x5 matrix and decides whether a checksum is correct if the same checksum value is repeated, say three times. If valid checksums are found for the five substrings, the modem calculates the checksums of the received substrings (SS1 to SS5 in Figure 6) and compares those with the checksums that the transmitter calculated. If a match is obtained, the substring is considered correct. When the five substrings obtain a match the complete message can be reconstructed. Note different sub-strings can come from different rows. Figure 6 depicts the existing system using color coding to identify which

checksum corresponds to which sub-string.

Figure 6: Example of the Checksum system

3 Working Methodology

There will be three distinct phases in this project. Each phase has its own methodology that we describe below.

3.1 Selection of Codeword Sets and Decoding Algorithm

During this phase, we will work on the selection of the sets of codewords. A set of Codewords is comprised by a finite set of elements in the field of binary words of length n. The word width of the embedded microcontroller used in the radio buoys is eight bits so it makes sense that we select $n = 8$. Multi-byte operations are very costly in terms of code size and performance.

For the message size k, we will use several candidates that will provide us with a trade-off between the code rate R and the achievable Bit Error Rate. The code rate of the current FEC scheme is $R = 0.2$ and because we aim to improve this scheme in terms of both R and BER, it makes sense to use values of k such that

$$k \in \{2, 3, 4\}$$

, which will provide with $R = [0.25, 0.375, 0.5]$.

The decoding algoritms for FEC codes can be sophisticated. Often times involving matrix multiplications and inversions. Matrix operations are, naturally, not supported natively by our 8-bit PIC microcontroller so we would have to implement or port a matrix library. This seems out of the question given how computationally expensive these operations can be. The alternative is to use a Look Up Table (LUT) that relates the 2^k message words with each of the codewords. Each potential 8-bit received codeword that is potentially corrupted would be checked agaisnt all 2^k possibilities and the shortest Hamming distance would be selected. For a worse case of $k = 4$ a maximum of 16 posibilities would be checked so the task seems attainable even for our 8-bit microcontroller.

3.2 BER Simulations

Once the codewords sets have been identified, we will need to somehow quantify their performance compared to the existing FEC scheme and decide which one is best for our purposes. We will do this by writing proper simulations in the Python language. There are two very convenient modules in Python that we will rely upon for our work. The first one is numpy, which is widely used for matrix and vector operations. The second one is matplotlib that offers a very rich set of features for generating 2-D and 3-D plots. For the development of the Python scripts we will use Micrososft's VS Code as the editior of choice. More details about this tool in section 1.3.

The basic idea of the simulation is to generate a set of bits that will be perturbed with certain random bit errors according to a given Bit Error Rate. The receiver will implement in Python the decoding algorithms that aim to recover the initial message free of bit errors. Each plot will show the BER on the x axis and the probability of recovering the error free message on the y axis. Naturally, many attempts would be simulated for every BER such that meaningful statistical significance can be extracted.

3.3 Implementation in the Commercial Product

Once the winning FEC codeword set is selected, we will proceed to the implementation in the commercial radio buoy. It will be done in C language using the MP Lab X toolchain provided by Microchip Techonology, the manufacturer of our microcontroller. MP Lab X includes a fully featured editor, compiler, linker and step-by-step debugger. During development, flash programming is done using Microchip's PicKit 4. Again, more details about these resources can be found in section 1.3.

The actual implementation in C and release to the market involves extensive sea trials with a fishing vessel having to sail tens of miles away from the coast in order to validate the additional range that the new FEC algorithms can achieve. Sea trials are out of the scope of this work and hence be reserved for future work. Enisa will have to fund the trials and validate the results also in an empirical fashion.

4 Development of the algorithm

In this section, a detailed description of the software developed for this project will be accomplished.

It is important to define the message that will be sent by A to B in order to have a better understanding of how the system works. The information will be the similar to the one that is presented in section 2.3, with the difference that for the next different cases, the string will not be sent five times.

4.1 FEC used for the algorithm

Given the hardware and software limitations that the microchip has, it cannot be adapted a very sophisticated FEC mechanism. Usual tools used in FEC algorithms such as matrix multiplication or polynomial division on Galoise Fields are not possible in our microcontroller. Neither there is any dedicated hardware, so all the encoding and decoding needs to be done by software.

Since the PIC16F18326 is an 8 bit microcontroller, it seems natural that our codewords are 8 bits wide. Any multi-byte operation incurs in significant overhead that may deem the solution difficult to implement. The decision that needs to be made is how many bits does the message take and how many parity bits. For example, if we were to use a simple message of 2 bits, we would have 6 bits for the parity. Actually, rather than decide up front, a simulation will be developed in order to find the performance and implementation difficulty. Thus, the best options will be decided.

The current FEC scheme uses a rate of $1/5 = 0.2$, as shown in 2.3, that is, it is used a five times redundancy. The code rate can also be defined as the ratio of the number of message bits divided by the number of code bits. The goal for this project is to improve the performance in the presence of bit errors (caused by a noisy channel) as well as reducing the number of redundant bits (increase code rate).

Table 1 shows the key parameters of each of the candidate code schemes.

The selected code sets will be such that they maximize the minimum Hamming distance between each two code words.

Message Bits k	**Alphabet Size** 2^k	**Codeword Bits** n	**Code Rate** R
2	4	8	0.25
3	8	8	0.375
4	16	8	0.5

Table 1: Parameters of Candidate Codes

For example, in the (2, 8) case, the alphabet is just four words $(00, 01, 11, 10)$ so the need is to pick just four 8-bit codewords, the other 252 possible words will not be used. Similarly, for the (3, 8) case, the alphabet is 8 words so the pick will be eigth 8-bit codewords at the same time that the other 248 are discarded. It is easy to extend the scheme for other cases.

Coming back to the simplest case (2, 8). When there is a byte to transmit, two bits will be picked at a time (by shifting and bitwise &) and selected one of the four entries of our LUT. For example, one LUT for this case could be Table 2.

Message Bits	**Codeword**
00	00000000
01	00000001
11	00000011
10	00000010

Table 2: Possible LUT for the case (2, 8)

Another possible example could be the one depicted in Table 3

Message Bits	**Codeword**
00	01010101
01	11001100
11	00011100
10	11110000

Table 3: Another possible LUT for the case (2, 8)

Neither LUT is necessarily an optimal set of codewords, they are just

examples. So if some bits were flipped due to the noisy channel, it seems the second table would be more "robust" because the codewords are more "different".

The codewords are sent over the air to the receiver so, for example, for the case (2, 8) four bytes will be needed to transmit every byte of the message (rate 0.25). For the cases (3, 8) and (4, 8), three and two bytes respectively would be needed (rates 0.375 and 0.5). Now the question is how to pick the best set of codewords that maximize the minimum Hamming distance from one codeword to another.

4.2 Main concepts used for the development of the algorithm

The selection of candidate codewords has had the following way of achieving the goal of having a set of codewords which maximized the minimum Hamming distance between them. The idea of maximizing this parameter has to do with the concept of B having the capacity of detecting and correcting a determined quantity of errors per byte sent.

4.2.1 Algebraic coding theory

Before explaining how the codewords were selected, it is important to define some aspects of the Algebraic Coding Theory. In coding theory, finite fields are used extensively and, in particular, Galois Fields of two elements or GF(2) in short [9]. GF(2) defines addition and multiplication as in the truth tables below: product is like a normal product and addition is just modulo 2 addition.

+	**0**	**1**
0	0	1
1	1	0

x	**0**	**1**
0	0	0
1	0	1

Table 4: Addition and multiplication over GF(2)

As explained before in section 2.1, the main purpose of the FEC is to

correct the bits that may get corrupted in a message sent from A to B. For that to take place, the system will need to add some redundancy information that can be used to recover the message if an error appears. In order to see how the system may work, here is an example:

Considering the encoding map $ENC : [0,1]^3$ to $[0,1]^4$ that is given by

$$ENC(x_1, x_2, x_3) = (x_1, x_2, x_3, x_1 + x_2 + x_3) \tag{6}$$

For example, if the message is $(0,1,0)$, the codeword would be: $C = ENC(0,1,0) = (0,1,0,1)$

This example can correct up to one erasure. An erasure takes place when one bit got erased and its value cannot be read. If this happens, the algorithm should be able to recover the bit based on the value of the other three bits that form the message. Continuing with the same example, if the information recieved was: $C_r = (0,1,x,1)$, the value of the third position would be obtained thanks to an system of equations obtained with the conditions imposed by the algorithm developed at A.

$$\begin{cases} x_1 = 0 \\ x_2 = 1 \\ x_3 = x \\ x_1 + x_2 + x_3 = 1 \end{cases} \tag{7}$$

At the same time, this same example is also able to detect one error, but it does not have the capacity of correcting it. Going back to the example, if the information received was: $C_r = (0,0,0,1)$, the receiver will note that there is something bad in the information because the fourth equation cannot be solved.

This idea can be generalized for any encoding map considering the concept of the Hamming distance defined previously in section 2.1. Thus, being d the minimum Hamming distance between two words in an encoding map, it is defined that a code can:

- Correct up to $d-1$ Erasures

- Detect up to $d-1$ Errors
- Correct up to $\frac{d-1}{2}$ Errors

The radio receiver in the buoy does not generate erased bits and even if we detect that an error has been detected, we cannot do much with that information. It is really the correction of errors that we are interested in for this application.

4.2.2 The Hamming bound

The Hamming bound is the answer to the question that tries to find the best relation between the code rate and the distance from one codeword to another. In order to understand this concept, we first define some terms that will contribute to an easier understanding of what the Hamming bound is.

Firstly, if the finite field, defined in section 2.1, F^n is thought of as a space that contains all the codewords inside, the Hamming Ball can be understood as the sphere of radius $\frac{d-1}{2}$ that surrounds the different codewords that compose the encoding map of a code, a more, this idea can be represented as shown in Figure 7.

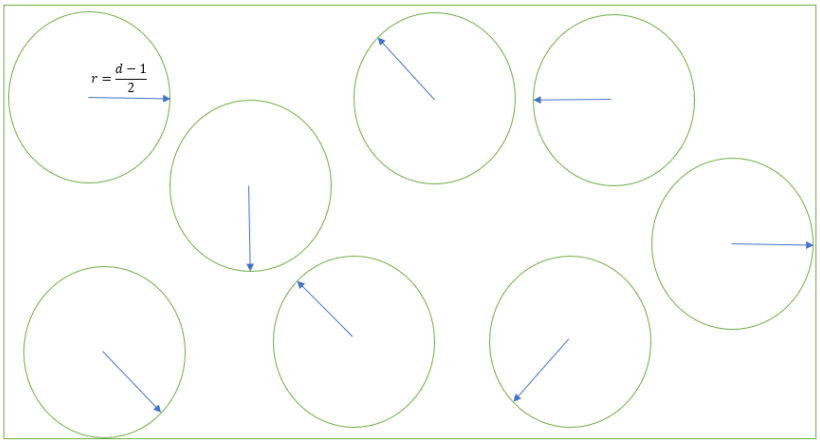


Figure 7: 2D representation of a finite field and its Hamming balls

More formally, the definition of a Hamming ball is:
Given $e = \frac{d-1}{2}$, the Hamming ball B is defined as:

$$B_{F^n}(x, e) = [y \in F^n : \Delta(x, y) \leq e] \tag{8}$$

Now, the volume of the Hamming ball is going to be the next term defined in order to understand the idea of the Hamming bound. Thus, the Volume can be defined as:

$$Vol_{|F|}(e, n) = |B_{F^n}(0, e)| \tag{9}$$

Going back to the idea of F^n as a space region, the total volume of this region has to be greater or equal than the volume of the Hamming ball times the number of codewords that compose a code with a distance d and a message length k. More formally, if $|F^n|$ is defined as q,

$$|C| \cdot Vol_q(\frac{d-1}{2}, n) \leq q^n \tag{10}$$

After applying logarithms and simplifying this equation, the Hamming bound is finally defined as,

$$HB = \frac{log_q|C|}{n} \leq 1 - \frac{log_q(Vol_q(\frac{d-1}{2}, n))}{n} \tag{11}$$

4.2.3 Linear codes

The algorithm used with the idea of improving the current communication between the buoy and the radio is based on linear codes. In order to understand how it works, first an introduction to the subject will be carried out.

A linear code is a k-dimensional subspace C determined by a basis $[c_1, c_2, ..., c_k]$ that is composed by any codeword which can be expressed as a combination of these vectors. More formally, a linear code of dimension k and length n over a finite field F^n is a k-dimensional subspace of F^n. The vectors of the matrix can be written as a matrix called the generator matrix G which is used to obtain any codeword from C. Using the definitions of both the linear

code and the generator matrix we can define C:

$$C = [G \cdot x : x \in F^k] \tag{12}$$

Note that in the same code C many generator matrices can be obtained. Precisely, there is always a generator matrix that contains an identity matrix of dimension k, with k being the number of rows and columns of the matrix. Continuing with the example provided in section 4.2.1: The encoding map ENC can be defined as a linear code:

$$C = [(1, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1)] \tag{13}$$

And one possible generator matrix with the identity matrix on it could be:

$$G = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{14}$$

We have seen that the matrix G can be used at the transmitter (radio buoy) to generate the right codewords given the message bits. Note how the matrix G relates vectors of k components (message words) into codewords of n components. The matrix G is thus an nxk matrix. As discussed before, given our encoding maps consist of very few words, they will be best implemented using a LUT rather than the matrix multiplication.

At the modem receiver, we will use the so called parity check matrix H, and it is such that the code C is the kernel of H. Formally defined as:

$$C = Ker(H) = [x \in F^n : H \cdot x = 0] \tag{15}$$

A good observation is that the distance of the code C is the number d such that H has d linearly dependent columns.

In simple terms, the matrix H results in zero when multiplied by a correct codeword. If the codeword has been damaged during transmission, the multiplication will not give zero. Moreover, the result of this multiplication

can help us determine which are the bit(s) that have been received in error [1].

4.3 Selection of candidate codewords

Now that the main concepts of linear codes are defined, the next step is to see their application for the algorithm developed. As defined in section 2.1 the three different codes studied are:

- Case (2, 8): a code of message length 2 and codeword length equal to 8 bits, which gives a rate of $R = 0.25$.
- Case (3, 8): a code of message length 3 and codeword length equal to 8 bits, which gives a rate of $R = 0.375$.
- Case (4, 8): a code of message length 3 and codeword length equal to 8 bits, which gives a rate of $R = 0.5$.

Here, in Figure 8, is a graphic that presents all the possible Hamming distances between the 256 candidates that are being evaluated.

4.3.1 Case (2, 8)

For the selection of codewords for this case, a simulation that calculated all the Hamming distances between the possible codewords took place. The algorithm's objective was to determine the set of codewords that could have the maximum minimum distance between codewords.

Thus, the set of codewords obtained was:

$$C = \begin{cases} (0,0,0,0,0,0,0,0) \\ (0,0,0,0,1,1,1,1) \\ (1,1,1,0,0,0,1,1) \\ (1,1,1,1,1,1,0,0) \end{cases} \tag{16}$$

Figure 9 shows how the Hamming distances between the codewords are selected.

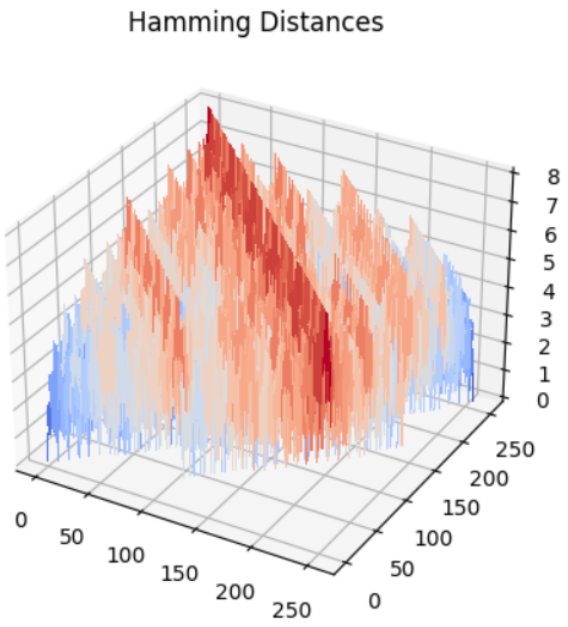


Figure 8: Hamming distances between all possible codewords

In order to obtain a generator matrix and a parity check matrix that contained the identity matrix as defined in section 4.2.3 and to have the first two bits as the message bits, a reordering of some bits in the codewords three and four was carried out. Thus, the similar code obtained and the one that will be implemented is:

$$C = \begin{cases} (0,0,0,0,0,0,0,0) \\ (0,1,0,0,1,1,1,1) \\ (1,0,1,1,0,0,1,1) \\ (1,1,1,1,1,1,0,0) \end{cases} \tag{17}$$

For this case, the code is able to correct up to two errors because the minimum distance between the given codewords is five. Now, with the selected codewords, as shown in section 4.2.3, it is possible to form the generator matrix and the parity-check matrix using the selected candidates. Thus, the matrices are:

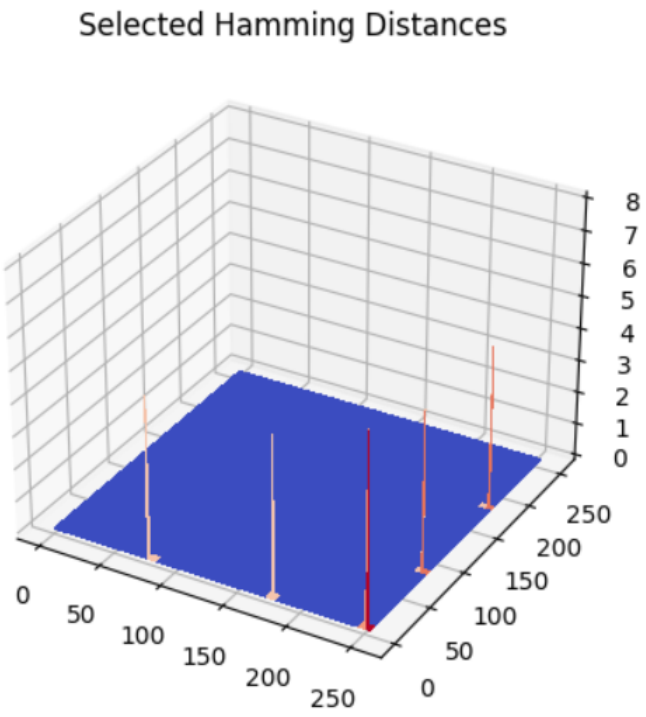


Figure 9: Case (2, 8): Hamming distances between the selected codewords

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{18}$$

4.3.2 Case (3, 8)

For the selection of codewords for this case, a simulation that calculated all the Hamming distances between the possible codewords took place. The algorithm's objective was to determine the set of codewords that could have

the maximum minimum distance between codewords.

Thus, the set of codewords obtained was:

$$C = \begin{cases} (0,0,0,0,0,0,0,0) \\ (0,0,0,0,1,1,1,1) \\ (0,0,1,1,0,0,1,1) \\ (0,0,1,1,1,1,0,0) \\ (0,1,0,1,0,1,0,1) \\ (0,1,0,1,1,0,1,0) \\ (0,1,1,0,0,1,1,0) \\ (0,1,1,0,1,0,0,1) \end{cases} \tag{19}$$

In order to obtain a generator matrix and a parity check matrix that contained the identity matrix as defined in section 4.2.3 and to have the first two bits as the message bits, a reordering of some bits in the codewords three and four was carried out. Thus, the similar code obtained and the one that will be implemented is:

$$C = \begin{cases} (0,0,0,0,0,0,0,0) \\ (0,0,1,0,1,1,0,1) \\ (0,1,0,0,1,0,1,1) \\ (0,1,1,0,0,1,1,0) \\ (1,0,0,0,0,1,1,1) \\ (1,0,1,0,1,0,1,0) \\ (1,1,0,0,1,1,0,0) \\ (1,1,1,0,0,0,0,1) \end{cases} \tag{20}$$

Figure 10 shows how the Hamming distances between the codewords are selected.

For this case, the code is able to correct up to one error because the minimum distance between the given codewords is four. The generator and

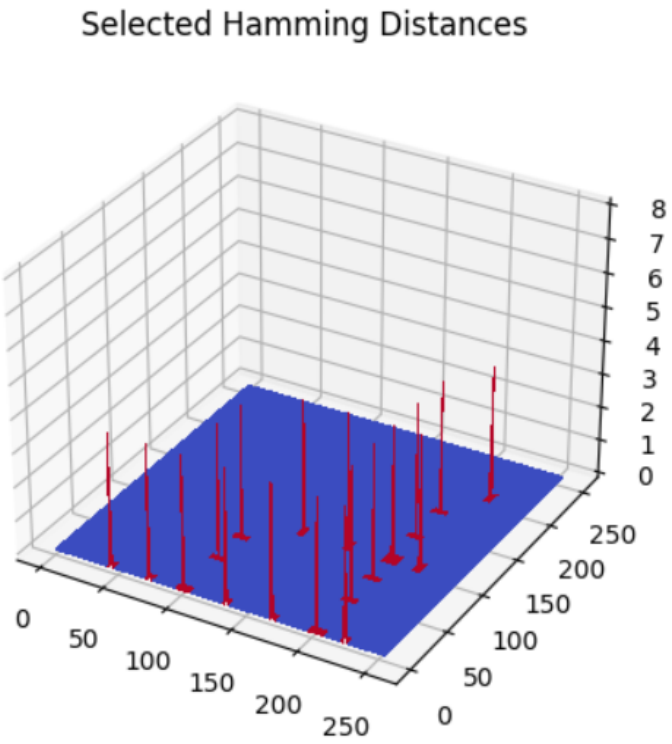


Figure 10: Case (3, 8): Hamming distances between the selected codewords

parity-check matrices will not be calculated for this case because, as it will be shown in the next sections, it will not be implemented as it could correct the same amount of errors than the (4, 8) case while sending 12.5% less information.

4.3.3 Case (4, 8)

For the selection of codewords for this case, a simulation that calculated all the Hamming distances between the possible codewords took place. The algorithm's objective was to determine the set of codewords that could have the maximum minimum distance between codewords.

Thus, the set of codewords obtained was:

$$C = \begin{cases} (0,0,0,0,0,0,0,0) \\ (0,0,0,0,1,1,1,1) \\ (0,0,1,1,0,0,1,1) \\ (0,0,1,1,1,1,0,0) \\ (0,1,0,1,0,1,0,1) \\ (0,1,0,1,1,0,1,0) \\ (0,1,1,0,0,1,1,0) \\ (0,1,1,0,1,0,0,1) \\ (1,0,0,1,0,1,1,0) \\ (1,0,0,1,1,0,0,1) \\ (1,0,1,0,0,1,0,1) \\ (1,0,1,0,1,0,1,0) \\ (1,1,0,0,0,0,1,1) \\ (1,1,0,0,1,1,0,0) \\ (1,1,1,1,0,0,0,0) \\ (1,1,1,1,1,1,1,1) \end{cases} \tag{21}$$

In order to obtain a generator matrix and a parity check matrix that contained the identity matrix as defined in section 4.2.3 and to have the first two bits as the message bits, a reordering of some bits in the codewords three and four was carried out. Thus, the similar code obtained and the one that will be implemented is:

$$C = \begin{cases} (0,0,0,0,0,0,0,0) \\ (0,0,0,1,0,1,1,1) \\ (0,0,1,0,1,0,1,1) \\ (0,0,1,1,1,1,0,0) \\ (0,1,0,0,1,1,0,1) \\ (0,1,0,1,1,0,1,0) \\ (0,1,1,1,0,0,0,1) \\ (1,0,0,0,1,1,1,0) \\ (1,0,0,1,1,0,0,1) \\ (1,0,1,0,0,1,0,1) \\ (1,0,1,1,0,0,1,0) \\ (1,1,0,0,0,0,1,1) \\ (1,1,0,1,0,1,0,0) \\ (1,1,0,1,0,1,0,0) \\ (1,1,1,0,1,0,0,0) \\ (1,1,1,1,1,1,1,1) \end{cases} \tag{22}$$

In Figure 11 the Hamming distances between the codewords selected are shown.

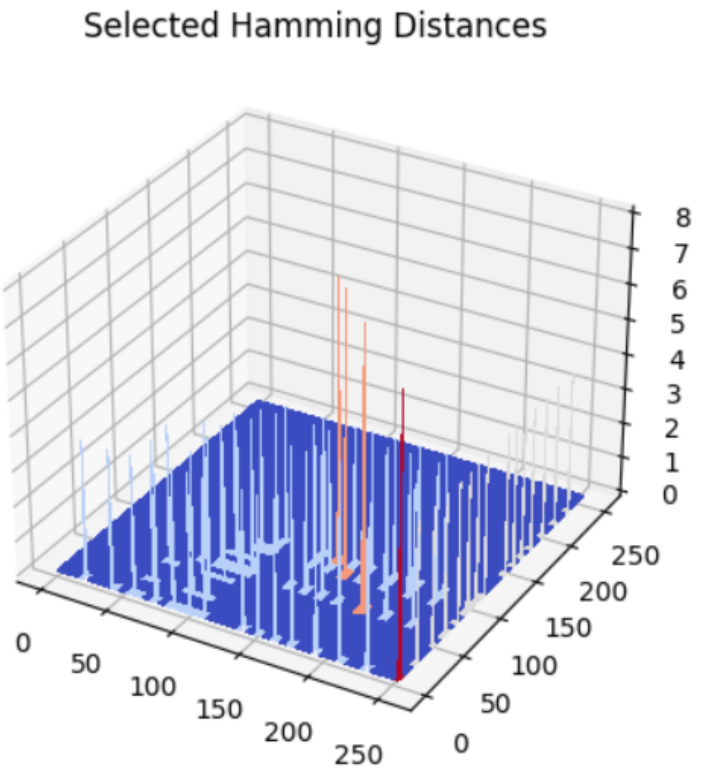


Figure 11: Case (4, 8): Hamming distances between the selected codewords

For this case, the code is able to correct up to one errors because the minimum distance between the given codewords is four. Now, with the selected codewords, as shown in section 4.2.3, it is possible to form the generator matrix and the parity-check matrix using the selected candidates. Thus, the matrices are:

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{23}$$

The next example will ilustrate how the algorithm is supposed to work for case (4, 8) but it could be applied to the other two cases. For example, lets imagine that we want to transmit the message $x = 1, 0, 0, 1$. The next step in order to obtain our codeword is to multiply the message using the G matrix. Thus:

$$C = G \cdot x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{24}$$

Now that we have our codeword it should be sent from the buoy to the receiver. If the codeword is received without any errors on the eigth bits, the parity P of the codeword C and the matrix H should be 0. Thus:

$$P = H \cdot C = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{25}$$

This will happen if the transmission was achieved with no errors. If a mistake happened during the process, a bit would be flipped. In order to find how the algorithm can solve it, lets go back to the previous codeword defined in equation 24, $C = [1, 0, 0, 1, 1, 0, 0, 1]$. For example, the bit that will change its value is going to be the third bit, this will make $\tilde{C} = [1, 0, 1, 1, 1, 0, 0, 1]$. Now that $\tilde{C}$ has been received at the modem, the algorithm calculates the parity to see if the message received is correct.

$$P = H \cdot \tilde{C} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{26}$$

Given the parity $P = [1, 0, 1, 1]$ that is equal to the third column, we can know for a fact that the bit that needs to be flipped is the third bit from the original information received, so that the information can be perfectly recovered.

5 BER simulation environment

In this section we will explain the development of the simulation environment that must imitate real-life situations where errors can appear randomly and with different rates depending on the propagation distance, channel quality, interference and so on. The functions defined and explained in this section are defined in section 10

The simulation environment needed four different steps. First, there was a selection of the test data which mainly consisted in an example of the string that could be sent in a real life situation. After selecting the data, it must be run through the transmission function from either the current algorithm or the proposed one using linear codes. Here is the code used first for the current algorithm and second for the algorithm using linear codes.

Next, once the BERs chosen (from 10^{-3} to 10^{-2} in steps of 10^{-3} and from 10^{-2} to 10^{-1} in steps of 10^{-2}), the data should be run through a function which will introduce the errors in the string randomly. This function is called *add_ber*. Here is the code used for the function in Python:

Finally, the new string with the BERs applied, will be received and checked by the receive function that simulates the algorithm that is included in the modem receiver on board the ship. This function will decide whether the result obtained after the decoding is good or bad. Here are the receiver functions for both the current and the linear code algorithms.

6 Simulation results

In this section, a comparison between all the different algorithms will be shown. The two main metrics are the efficiency and the probability of receiving an error-free message for a given bit error rate (BER).

6.1 Theoretical BER curves

Once the simulation environment has been developed and the sets of codewords have been selected, the next step is to come up with a theoretical formulation that mimics the expected performance curves and then comparing them with the simulated ones.

A summary of the hamming distance of each of the sets and the maximum number of errors that can be corrected per case is shown in table 6.1.

Message Bits k	**Codeword Bits** n	**Hamming Distance** d	**Max Errors** e
2	8	5	2
3	8	4	1
4	8	4	1

Table 5: Summary of the cases studied

With all this information, it is posible to confirm that the options (3, 8) and (4, 8) must have a similar BER performance given that both cases can only correct up to one error. Looking at Table 6.1, we know that our codes will fail when there is at least one event within our entire message that shows greater number of errors than our maximum allowed. We can think of this as two layered binomial distributions, so given a certain BER, the first one is the probability of a certain number of bit errors within a codeword (byte).

$$P(0) = (1 - BER)^8 \tag{27}$$

$$P(1) = C(8, 1) \cdot BER^1 \cdot (1 - BER)^7 \tag{28}$$

In general for a number of i errors:

$$P(\frac{i}{byte}) = C(8,i) \cdot BER^i \cdot (1-BER)^{8-i} \tag{29}$$

Where $C(p,q)$ are the combinations of p elements taking q at a time:

$$P(errors > \frac{i}{byte}) = P_i = 1 - (P(0) + P(1) + ... + P(i)) \tag{30}$$

The final metric of the BER performance is the probability that we have at least one of these events (more than i errors in one byte) in the whole of our message. This is the second binomial distribution. If our message has N bytes, the transmission has $M = \lceil \frac{N}{r} \rceil$ bytes where r is the rate of the code $r = \frac{k}{n}$. Thus, the final metric will be:

$$P_m = (1 - P_i)^M \tag{31}$$

Figure 12 shows the results for the cases of maximum number of errors one and two. From the obtained Hamming distances, the case of more than one error would be similar to the (3,8) and (4,8) cases and the case of more than 2 errors would be similar to the (2,8) case. The X axis has all the different BERs labeled using the logarithm scale. The Y axis represents the probability of success in percentage

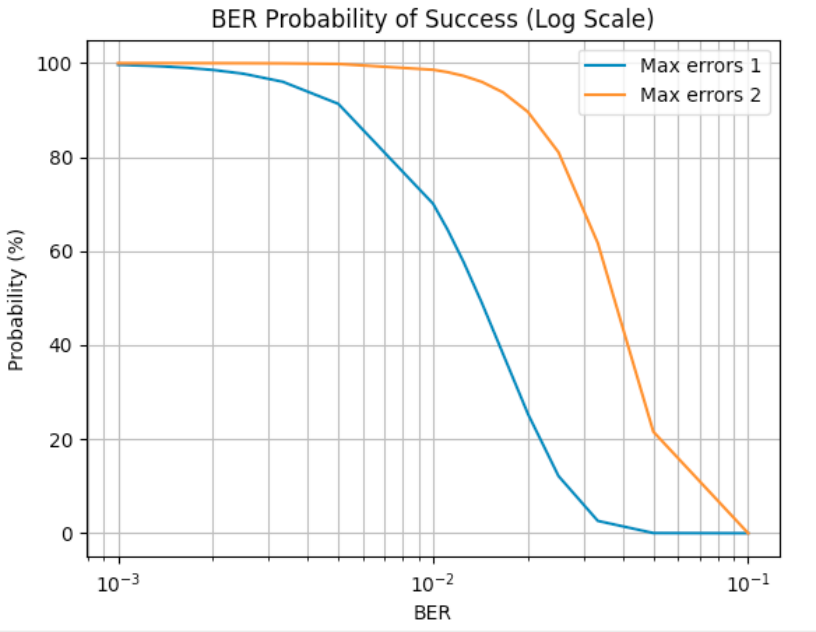


Figure 12: Theoretical BER curves for one and two errors

6.2 Comparison between algorithms

Now that the theoretical curves for the Hamming algorithm have been plotted, the next step is to find which will be the real performance in our simulation environment described in section 5.

First, it is important to know that for the comparison with the current algorithm, two different versions of it have been taken into account. The first one is perfectly defined in section 2.3 and the other version has a modification that consists in sending the string three times instead of five. This choice was motivated by the use of a channel in Japan whose regulatory requirements limited the transmission rate to 100 bauds (as opposed to 1200 bauds in other markets) and forced to use less redundancy in order to respect the timing of each transmission slot. The rate in this case is 33% of information sent with a 66% of the message being redundancy, while in the original version it is just a 20% rate.

Second, the expectations for both algorithms are to have a better performance once the BER increases if the message rate is the lowest possible in

the same algorithm. Thus, for the original algorithm the one that has a 20% rate will obviously have a higher probability of recovering the string correctly than the case with a 33% rate, as explained in previous sections the bigger the redundancy bits per byte, the easiest for the receiver to achieve its goal. The cost of opportunity relies in the need of having to transmit more data in order to obtain the whole message.

Knowing all this information and after running each algorithm a thousand times per case and per BER through the simulation environment explained in 5 in order to obtain a reliable probability of what could happen, here are the results obtained represented in Figure 13. The X axis has all the different BERs labeled using a logarithm scale. The Y axis represents the probability of success in percentage.

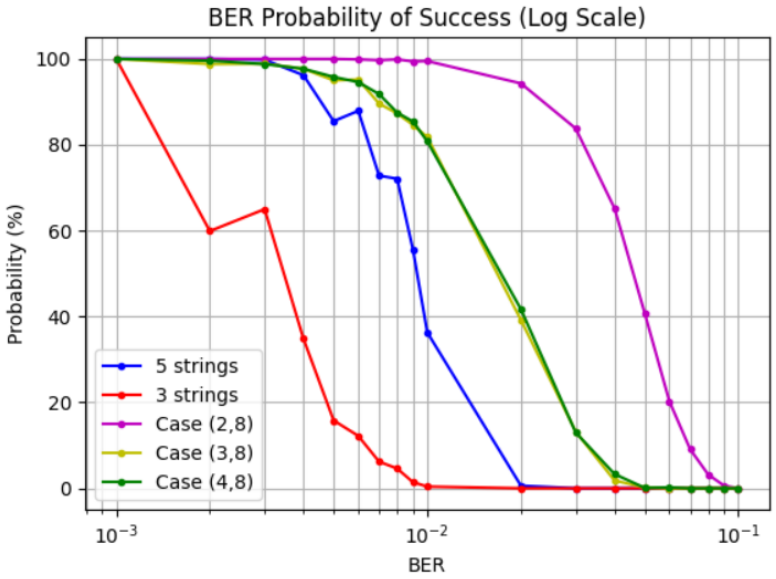


Figure 13: Simulation results

From Figure 13 the most important conclusion that can be obtained is that the Hamming algorithm improves the performance of the current algorithm in all cases. If we happen to compare cases (3, 8) and (4, 8) with case (2, 8) we can observe how for high value BERs the difference is very significant, having a perform difference of almost 60%. Another good conclusion from this figure is that as commented in previous sections, both cases (3,8) and (4, 8) can only correct up to one error per byte, so basically the performance needed to be the same. This hyphotesis can be confirmed by looking at how both curves are superimposed.

It is also possible to see how all the curves show a decreasing tendency as the BER increases. From the 3 strings case in comparison with the 5 string case (both from the current algorithm), there is a huge difference related to the performance, i.e. when the BER value is 0.004 the difference goes up to almost 70% from one to another. We can also see how both of them have a very linear form as the curves decrease while also having the both cases some peaks because of how it works when recovering the information.

6.3 Theoretical vs. Simulation curves

The next Figure will compare both the theoretical curves and the Hamming cases' curves.

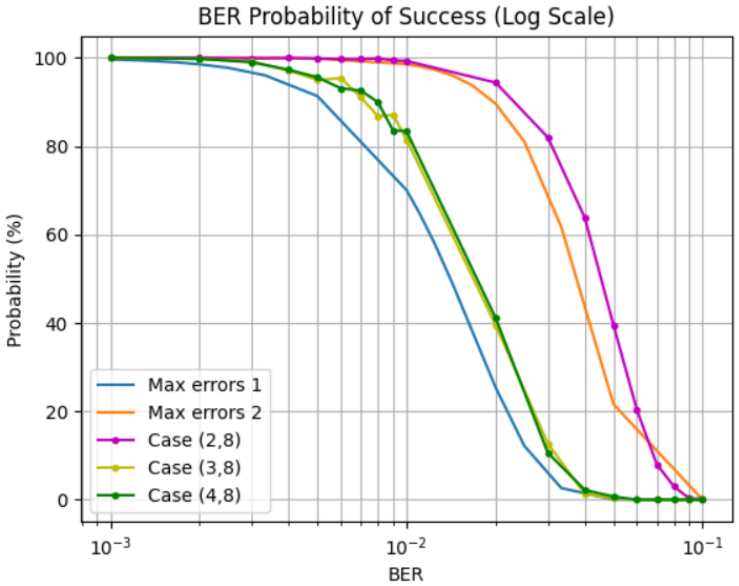


Figure 14: Theoretical vs Simulation curves

In Figure 14 the comparison between what was achieved in section 6.1 and the results obtained from a simulation can help us to better understand how the algorithm works. As we can see, the simulation is more optimistic than what the theory indicates. Also, it is possible to see how the shape of the curves is the same in both cases. Between having one or two errors, in Figure 14 we can see how the case with two errors is more similar to the theoretical curve than the Max errors 1 case.

7 Implementation in the Commercial Product

In this section we will explain the implementation in the commercial product. The actual functions used in the final implementation are shown in section 11. For the encoding in the buoy, a simple Look Up Table (encoding map) in the form of a C array is used to prevent any matrix manipulation and thus speed up the execution. For the decoding, there is in fact a matrix-vector multiplication between the received codeword $\tilde{C}$ and the parity check matrix H. This is of course a more expensive operation as it can be seen in the obtained decoding times in tabble 6. In any case, a matrix multiplication over a GF2, can be implemented by just doing the XOR of the columns where the received codeword has ones. Doing the XOR of two eight bit numbers is a one instruction cycle operation in most microcontollers.

For the implementation in the commercial product the algorithm needs to complete under a certain time since the radio-buoy system is a TDMA (Time Division Multiple Access) as explained in section 1.4.1. At the moment of writing this project, the time per buoy needed for the transmision is ten seconds, so that every receiver can have up to sixty different buoys that are able to transmit in a period of ten minutes.

In order to precisely measure the time that the algorithm uses for transmiting and recovering the information, MP Lab provides a "stopwatch" tool that outputs the number of cycles that each C code instruction takes. From Microchip documentation, we know that one instruction cycle takes 4 clock cycles and we know the exact frequencies of the oscillators that drive the two microprocessors in the buoy and the modem so a deterministic time can be found for our algorithms. We will measure the cases (2, 8) and (4, 8). In Figure 15 the MPLAB environment is shown, and the measurements of the stopwatch can be seen in the window displayed under the code.

For the case (2, 8) the time measured for the transmission of one single byte was 36.77ms. The string which is going to be used will have a total of 276 bytes to transmit but could possibly have up to 316 bytes, having a total time of transmission of 2.53s and a maximum of 2.90s. The bytes are received

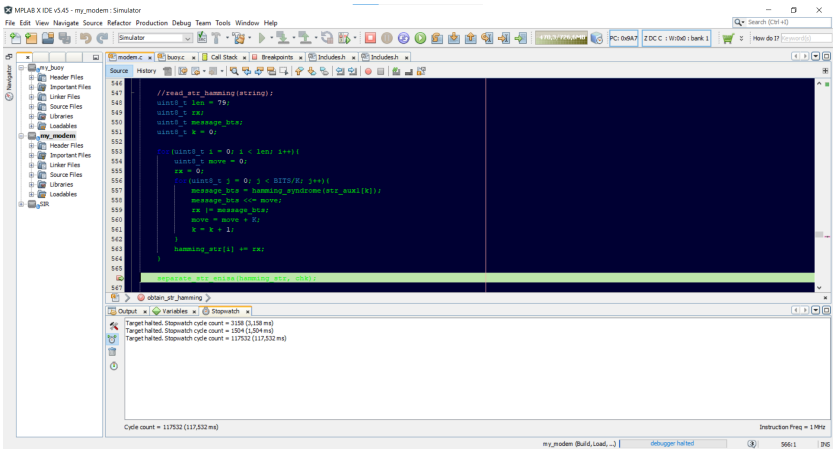


Figure 15: MPLAB Stopwatch environment

and storaged while the transmission takes place. Finally, the receiver would take a total time of 489ms to obtain the string with the Hamming algorithm for this case. The total time of the process is 3.02s.

For the case (4, 8) the time measured for the transmission of one byte was 18.39ms, this value concurs with the idea of transmiting half of redundant information compared with case (2, 8). The total time of transmission will then have a value of 1.27s for a string of 69 bytes before going through the algorithm (total number of bytes sent is 138 bytes with a 50% redundancy). The receiver will take 1.41ms to recover every byte from the original string, which will mean a total time of 232ms for the recovering of the string. The total time of the process is 1.50s.

Case	**Transmission/byte**	**Transmission time**	**Decoding**	**Total time**
	ms	s	ms	s
(2, 8)	36.77	2.53	489	3.02
(4, 8)	18.39	1.27	232	1.50

Table 6: Algorithm times measured during the implementation

8 Final Conclusions and Future Work

As sustainable fishing techniques are starting to become more important in our society due to the damage that traditionals techniques such as massive nets are causing, the aim of this project was to improve the performance of a radio-buoy system that used for longlines by studying different FEC codes that could be applied to the current system. The key metric in this study is the probability of recovering the message given a BER that will produce errors during the transmission.

By developing first a simulation environment and then measuring the times of action explained in section 7, it was proved that the algorithms can recover the message as expected. Also, as shown in Figure 13, a significant improvement was made in terms of the probability of recovering the information while increasing the code rate. From the three cases studied, two were finally implemented. Case (3, 8) was not include because it has the same performance as case (4, 8) although it had a difference of 12.5% with regard to the code rate. The implementation of cases (2, 8) and (4, 8) will help to increase the efficiency of the system in two different ways. Depending on the trade-off wanted, if what the user seeks is to have a bigger number of buoys linked to the receiver thanks to the TDMA, the case (4, 8) will allow to increase this number as it needs less time for the recovering of the message. On the other hand, case (2, 8) can be used in environments with a higher BER given that it has a higher probability of recovering the information when BER increases. This can also be translated into an increasement of the distance range.

Future research related to this project may relay on the study of a more sophisticated FEC technique such as Reed-Solomon codes, but it might involve the use of a new microprocessor because of the limitation of the current 8-bit microchip. Also, the difference between the theoretical and the simulation curves shown in Figure 14 could be studied more deeply. Finally, in order to see if the algorithm works in a real-life environment, sea tests should take place.

9 References

[1] Mary K. Wootters, Any errors in this dissertation are probably fixable: topics in probability and error correcting codes. PhD Thesis, Department of Mathematics, University of Michigan 2014

[2] Richard W. Hamming, Error Detecting and Error Correcting Codes. The Bell System Technical Journal, Vol. 29, April 1950

[3] Resumen de la reglamentación para el uso de la banda ciudadana CB27 en España. https://www.cb27.com/legal/reglamentocb Last accessed 10/06/2022.

[4] Visual Studio Code, Microsoft. https://visualstudio.microsoft.com/ Last accessed 22/03/2022.

[5] MPLAB, Microchip. https://www.microchip.com/en-us/tools-resources/develop/mplab-x-ide Last accessed 22/03/2022.

[6] Peter Elias, Processing and Transmission of Information. Research Laboratory of Electronics (RLE) at the Massachusetts Institute of Technology (MIT), 1955

[7] Irving S. Reed & Gustave Solomon, Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics, Vol. 8, No. 2, June, 1960

[8] Robert G. Gallager, Low-density parity-check codes. IRE Transactions on Information Theory, Vol. 8, No: 1, January 1962

[9] GF(2), Wikipedia. https://en.wikipedia.org/wiki/GF(2) Last accessed 26/06/2022.

[10] Pickpoint Products, https://www.enisa.com. Last accessed 6/07/2022.

10 Appendix A: Simulation Environment Source Code: Python

10.1 Transmission functions

```
1
2  def tx_enisa(test_st, n):
3      '''
4      This function takes a test string and calculates the 5
       checksums
5      Replicates the resulting string five times as the buoy
       does and returns it
6      '''
7
8      chksum_bounds = get_commas(test_st)
9      ck = ''
10     for idx in range(len(chksum_bounds)-1):
11         ck += '{:02X}'.format(chksum_chunk(test_st,
       chksum_bounds[idx], chksum_bounds[idx+1]))
12
13     test_st = ck + '/' + test_st
14
15     return [test_st] * n
16
```

```
1  def tx_hamming(enisa_str, bytes, codewords, n_case):
2      '''
3      This function takes a test string and separates it in
       groups related
4      to the number of codewords selected.
5      Thus, it can introduce the codewords before sending the
       new string to the modem.
6      The function emulates the buoy.
7      '''
8
9      enisa_str = enisa.tx_enisa(enisa_str, 1)
10     enisa_str = ''.join(enisa_str)
11     enisa_bytes = toBinary(enisa_str)
12     new_list = []
```

```
    length = len(enisa_bytes)

    for i in range(0, length , n_case):
        bits = enisa_bytes[i : (i + n_case)]
        if len(bits) < n_case:
            x = n_case - len(bits)
            bits = '0' * x + bits
        for j in range(len(bytes)):
            if bits == bytes[j]:
                new_list += [codewords[j]]

    ascii_list = Ber.bin2ascii(new_list)

    return ascii_list


```

10.2 BER function

```

import random

def add_ber(my_string, ber):
    # This function flips a number of bits in a string (bit errors)
    # The number of flipped bits is commensurate with the input BER

    byte_list = toBinary(my_string)
    n_bits = len(my_string) * 8
    bad = int(n_bits * ber)

    #Selects randomly the position of the wrong bit
    pos = random.sample(range(n_bits), bad)

    pos_int = [int(pos[i] / 8) for i in range(len(pos))]
    bit = [pos[i] % 8 for i in range(len(pos))]

    for i in range(len(pos_int)):
```

```
        byte = byte_list[pos_int[i]]

        #Changes the value of the bit
        if byte[bit[i]] == '1':
            byte = list(byte)
            byte[bit[i]] = '0'
            byte = "".join(byte)
        else:
            byte = list(byte)
            byte[bit[i]] = '1'
            byte = "".join(byte)

        byte_list[pos_int[i]] = byte

    #Next function changes the string from binary to
    characters
    ber_list = bin2ascii(byte_list)
    ber_string = "".join(ber_list)

    #Returns the new string
    return ber_string


```

10.3 Receiver functions

```

def rx_enisa(test_strs, string, n):
    '''
    This fuctions emulates the reception of the modem with
    the enisa string
    Reconstructs a valid string recovering from the five
    strings
    Uses the same algorithms as the modem
    '''

    decoded_ok = True
    chk_tests = [test_strs[i][0:10] for i in range(len(
    test_strs))]
```

```
    ckint = [hex2int(chk_tests[i][0:10]) for i in range(n)]
    ckcorrect = obtain_ckcorrect(ckint, n)
    strs = [test_strs[i][11:] for i in range(len(test_strs))]

    if ckcorrect is None:
        decoded_ok = False
    else:
        enisa_str = calculate_str(ckcorrect, strs, n, string)

    if (decoded_ok and enisa_str[66] == 'A' and enisa_str[46]
    == ',' and enisa_str[48] == ',' and enisa_str[40] == '.'
   and (enisa_str[47] == 'E' or enisa_str[47] == 'W') and
        enisa_str[32] == ',' and enisa_str[34] == ',' and
   enisa_str[26] == '.' and (enisa_str[33] == 'S' or
   enisa_str[33] == 'N') and
        enisa_str[21] == ',' and enisa_str[18] == '.' and
   enisa_str[11] == '/' and enisa_str[2] == ','):
            return True
    else:
        return False

```

```
def rx_hamming(bad_list, bytes, codewords, correct_str):
    '''
    This function takes a test string and separates it in
   groups related
    to the number of codewords selected.
    Thus, it can use the codewords storaged in the modem for
   recovering
    the string sent from the buoy.
    The function emulates the receiver.
    '''

    bad_list = toBinary(bad_list)
    good_str = ''

    for byte in bad_list:
        dist = [hamming_distance(byte, codewords[i]) for i in
    range(len(codewords))]
        pos = dist.index(min(dist))
```

```
16          good_str += bytes[pos]
17
18      good_list = []
19      good_list += [good_str[i:(i+8)] for i in range(0, len(
    good_str), 8)]
20      y=len(good_list)
21      x=len(good_str)
22      enisa_list = bin2ascii(good_list)
23      enisa_str = ''.join(enisa_list)
24      checksum = calculate_chk(enisa_str[11:])
25      enisa_checksum = hex2int(enisa_str[0:10])
26
27      if enisa_checksum == checksum:
28          return True
29      else:
30          return False
31
```

11 Appendix B: Implementation code: C

11.1 Buoy functions

```
1
2  void transmit_hamming(void){
3      // This function divides each byte into K codewords and then
       transmits the
4      // all the bytes that form the Hamming string.
5
6      uint8 length = strlen(str_enisa);
7      for(uint8 i=0; i<15; i++){
8          char2tone('U');
9      }
10
11     uint8 mask = (1 << K) - 1;
12
13     char2tone('/');
14
15     for(uint8 i=0; i<10; i++){
16         char2tone(str_chcksum[i]);
17     }
18
19     char2tone('/');
20     unsigned char aux_str[MAX_LENGTH];
21     uint8 k = 0;
22     uint8 lsbs;
23
24     for(uint8 i = 0; i < length; i++){
25         uint8 byte = str_enisa[i];
26         if(K == 2){ // Case (2, 8)
27             for(uint8 j = 0; j < BITS/K; j++){
28                 lsbs = byte;
29                 lsbs &= mask;
30                 uint8 codeword = enc_map2[lsbs];
31                 char2tone(codeword);
32                 byte >>= K;
33                 aux_str[k] += codeword;
34                 k++;
```

```
            }
        }else if(K == 4){ // Case (4, 8)
            for(uint8 j = 0; j < BITS/K; j++){
                lsbs = byte;
                lsbs &= mask;
                uint8 codeword = enc_map4[lsbs];
                char2tone(codeword);
                byte >>= K;
                aux_str[k] += codeword;
                k++;
            }
        }
    }

}


```

11.2 Receiver functions

```

uint8_t hamming_syndrome(unsigned char byte){
    // This function calculates the parity of the codeword
    received
    // and returns the message bits recovered

    uint8_t synd = 0;
    uint8_t codeword;
    uint8_t idx = 0;
    uint8_t aux = 0;
    codeword = byte;

    for(uint8_t i = 0; i < BITS; i++){
        if ((byte & 128) == 128){
            if (K == 2){ // Case (2, 8)
                synd ^= H_2[i];
            }else if(K == 4){
                synd ^= H_4[i];
            }
```

```
        }
        byte <<= 1;
    }
    if(K == 2){ // Case (2, 8)
        if (synd != 0){
            aux = Synd_matrix[synd];
            codeword ^= (1 << aux);
        }
        for(uint8_t j = 0; j < (K*K); j++){
            if(codeword == enc_map2[j]){
                idx = j;
            }
        }
    }else if(K == 4){ // Case (4, 8)
        if (synd != 0){
            for(uint8_t i = 0; i < BITS; i++){
                if(synd == H_4[i]){
                    aux = i;
                }
            }
            codeword ^= (1 << aux);
        }
        for(uint8_t j = 0; j < (K*K); j++){
            if(codeword == enc_map4[j]){
                idx = j;
            }
        }
    }
    return idx;
        }

```

```
uint8_t obtain_str_hamming(char str_aux[]){
    // This function recovers the string

    uint8_t chk[10];
    char hamming_str[MAX_LENGTH];
    uint8_t str_valid;

    read_str_hamming(string); // Reads the string received
```

```
    uint8_t len = strlen(string);
    uint8_t rx;
    uint8_t message_bts;
    uint8_t k = 0;

    for(uint8_t i = 0; i < len; i++){
        uint8_t move = 0;
        rx = 0;
        for(uint8_t j = 0; j < BITS/K; j++){
            message_bts = hamming_syndrome(str_aux1[k]);
            message_bts <<= move;
            rx |= message_bts;
            move = move + K;
            k = k + 1;
        }
        hamming_str[i] += rx;
    }

    separate_str_enisa(hamming_str, chk);

    if((hamming_str[46] == ',') && (hamming_str[48] == ',') && (hamming_str[40] == '.') && ((hamming_str[47] == 'E') || (hamming_str[47] == 'W'))
    && (hamming_str[32] == ',') && (hamming_str[34] == ',') && (hamming_str[26] == '.') && ((hamming_str[33] == 'S') || (hamming_str[33] == 'N'))
    && hamming_str[21] == ',' && hamming_str[18] == '.' && hamming_str[11] == '/' && hamming_str[2] == ','){
        str_valid = 1;
    }else{
        str_valid = 0;
    }
    return str_valid;
}

```

```
void HammingMode(){
    // This function represents the mode from the menu that initiates the recovering the
    // information and prints it in the screen of the receiver
```

```
4       TRISC5 = 0;
5       RC5 = 1;
6
7       while(1){
8           obtain_str_hamming(str); // Recovers the string
9           LED_GREEN_PIN = 1;
10          uint8_t j = 0;
11          LED_RED_PIN = 0;
12          while(enisa_str[j] != '\0'){
13              LED_GREEN_PIN = 1;
14              RC5 = 1;
15              UART_Transmit(enisa_str[j]);
16              j++;
17          }
18          UART_Transmit('\r');
19          UART_Transmit('\n');
20      }
21  }
22
```

12 Appendix C: Auxiliary Functions: Python

12.1 3D Hamming Plots

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import numpy as np
import itertools as it
import argparse
from num_ones import counts


def hamming_distance(byte1, byte2):
    return counts[byte1^byte2]

def heat_map(arr):
    map = []
    for el1 in arr:
        row = []
        for el2 in arr:
            row.append(hamming_distance(el1, el2))
        map.append(row)
    return np.array(map)


if __name__ == '__main__':

    parser = argparse.ArgumentParser(description="Script to
    plot the hamming distances of a set of 8-bit codewords")
    parser.add_argument('--code-words', help="String of
    codewords separated by spaces", default="0 15 51 60 85 90
    102 105")
    myargs = parser.parse_args()
    print("Codewords {}".format(myargs.code_words))

    code_words = [int(w) for w in myargs.code_words.split()]

    map = []
```

```
    for byte1 in range(256):
        row = []
        for byte2 in range(256):
            row.append(hamming_distance(byte1, byte2))
        map.append(row)
    map = np.array(map)

    ham_map = np.zeros((256,256),dtype=np.int64)
    for i,j in it.combinations(code_words, 2):
        ham_map[i,j] = map[i,j]

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    X, Y = np.meshgrid(range(256), range(256))
    surf = ax.plot_surface(X, Y, map, cmap=cm.coolwarm,
   linewidth=0, antialiased=False)
    ax.set_zlim(0, 8)
    plt.title("Hamming Distances")

    fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
    X, Y = np.meshgrid(range(256), range(256))
    surf = ax.plot_surface(X, Y, ham_map, cmap=cm.coolwarm,
   linewidth=2, antialiased=False)
    ax.set_zlim(0, 8)
    plt.title("Selected Hamming Distances")

    plt.show()


```

12.2 Theoretical Curves

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator
import numpy as np
import itertools as it
import argparse
from num_ones import counts

```

```
9
10 def hamming_distance(byte1, byte2):
11     return counts[byte1^byte2]
12
13 def heat_map(arr):
14     map = []
15     for el1 in arr:
16         row = []
17         for el2 in arr:
18             row.append(hamming_distance(el1, el2))
19         map.append(row)
20     return np.array(map)
21
22
23 if __name__ == '__main__':
24
25     parser = argparse.ArgumentParser(description="Script to
       plot the hamming distances of a set of 8-bit codewords")
26     parser.add_argument('--code-words', help="String of
       codewords separated by spaces", default="0 15 51 60 85 90
       102 105")
27     myargs = parser.parse_args()
28     print("Codewords {}".format(myargs.code_words))
29
30     code_words = [int(w) for w in myargs.code_words.split()]
31
32     map = []
33     for byte1 in range(256):
34         row = []
35         for byte2 in range(256):
36             row.append(hamming_distance(byte1, byte2))
37         map.append(row)
38     map = np.array(map)
39
40     ham_map = np.zeros((256,256),dtype=np.int64)
41     for i,j in it.combinations(code_words, 2):
42         ham_map[i,j] = map[i,j]
43
44     fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
```

```
45      X, Y = np.meshgrid(range(256), range(256))
46      surf = ax.plot_surface(X, Y, map, cmap=cm.coolwarm,
    linewidth=0, antialiased=False)
47      ax.set_zlim(0, 8)
48      plt.title("Hamming Distances")
49
50      fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
51      X, Y = np.meshgrid(range(256), range(256))
52      surf = ax.plot_surface(X, Y, ham_map, cmap=cm.coolwarm,
    linewidth=2, antialiased=False)
53      ax.set_zlim(0, 8)
54      plt.title("Selected Hamming Distances")
55
56      plt.show()
57
```

13 Appendix D: Sustainable Development Goals (SDG)

The Sustainable Development Goals are a common ground embraced by all the United Nations Member States in 2015 with the objective of contributing to the prosperity of the planet and its inhabitants. The 2030 Agenda for Sustainable Development has in its hearth the 17 SDG. They recognize the urgency of acting in order to reduce the inequality and pollution all around the world while applying efficient techniques that help to improve the economic growth, education and health.

The sustainable main objective of this project has to do with SDG no. 12, which is to ensure sustainable consumption and production patterns. This will be achieved by helping to develop longline fishing systems. Longline fishing is a technique that uses a main line from where several branches full of fish hooks separated in regular intervals depart. There are diverse types of lines, mainly differenced by its depth within the water. The use of this fishing system requires of a radio-buoy system in order for fishing companies to be able to locate their longlines, as shown in Figure 16.

Longlines are a much more sustainable fishing gear than, for example, trawlers that drag a massive net across the sea bed causing severe damage to marine habitats. Trawler nets are also not selective, fishing every creature in the sea bed. This is known as "by-catch" and it is minimized by the use of long lines. In addition, having to tow these big nets, trawlers use much more fuel than longlines.

The need for developing sustainable methods of fishing appears as a way of guaranteeing the ocean's wildlife in the future. It is known that fishermen catch more than 77 billion kilograms of marine species every year. In 2018, the United Nations Food and Agriculture Organization estimated that up to 33.1% of the world's fishing production is exposed to overfishing.

Concerning the environmental advantages of this technique, the most important benefits are reducing polution and protecting marine fauna while rejecting the indiscriminate capture of animals without commercial value. It

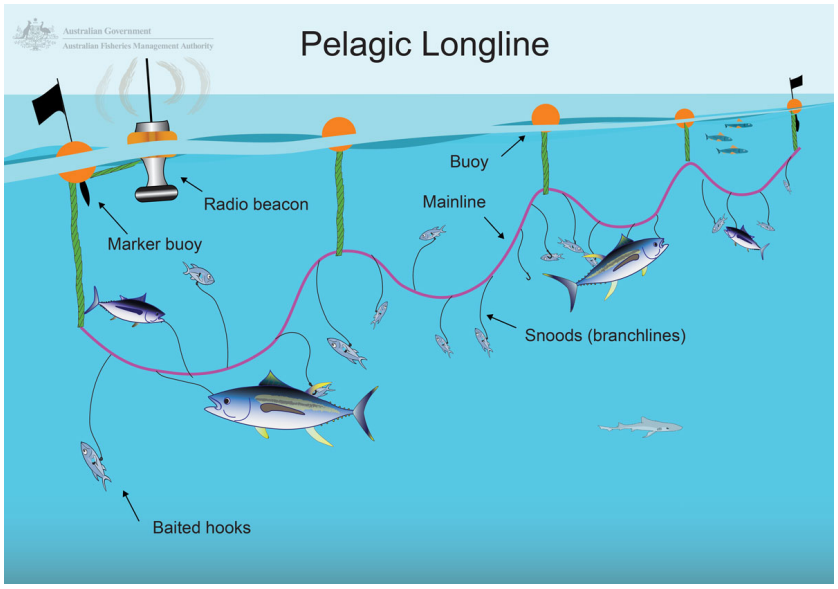


Figure 16: Longline fishing system (Australian Fisheries Management Authority)

is also suitable with saving 90% of the actual fishing industry's employment. The potential savings of using and developing longline fishing could grow up to saving 40% of the animals in every capture.

14 Appendix E: Pickpoint^{TM} Image Gallery

This section shows some images of the radio buoy system that have been taken from [10]

Figure 17: Radio Buoy with Antenna

Figure 18: Float for Radio Buoy

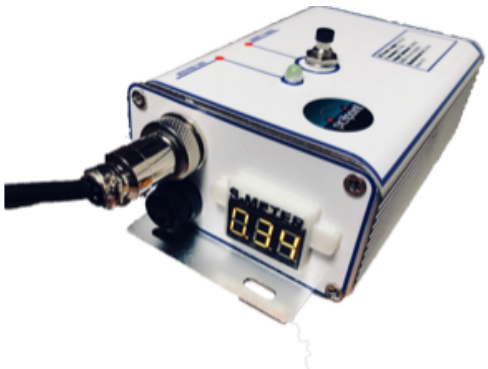


Figure 19: Modem Receiver Onboard