Fakultät Design, Medien und Information Department Medientechnik

Bachelor-Thesis

zur Erlangung des akademischen Grades B.A.

Kianosh Momo Kinz

[redacted]

Feasibility study about maritime data transfer through lighthouses using optical error correcting and detecting techniques

Erstprüfer: Jan Mietzner
Zweitprüfer: Prof. Peter A. Höher

Eingereicht am September 26, 2022

Contents

Abstract

The goal of this thesis is to use existing maritime navigation lights to transmit data to multiple ships at once. Since the light source may rotate, have blinking patterns, or be obstructed by waves, vessels can not receive a continuous stream of packages. Only when the photo detector sees the light, packages can be received. This led to the use of LT codes (fountain codes). The receiver is then able to collect any sufficient amount of packages and still decode the original message. In order to strengthen the signal against interference LDPC codes (error correcting codes) and CRC codes (error detecting codes) are added as well. Those three representatives of their respective categories are going to be tested in various configurations in order to explore their combined or single effects on bit error ratios and decoding times. It was found out that using the symbiosis between Fountain codes and error detecting codes, we can achieve really low bit error ratios while at the same time keeping computational needs at a minimum. For better results at really low SNR's, error correction codes can be added, resulting in better BER but an increase in decoding time.

Zusammenfassung

Ziel dieser Abschlussarbeit ist es schon vorhandene Navigationslichter aus der Schiffahrt zur Datenübertragung an eines oder mehrere Schiffe gleichzeitig zu nutzen. Da diese Lichtquellen sich häufig drehen, blinken oder durch hohe Wellen kurzzeitig blockiert werden, können Schiffe keine durchgängig andauernde Verbindung aufbauen, um kontinuierlich Daten zu empfangen. Aus diesem Grund schauen wir uns LT Codes (Fountain Codes) an. Durch sie kann der Empfänger zu beliebigen Zeitpunkten genügend Datenpackete einsammeln, um die originale Nachricht zu dekodieren. Um die Übertragung gegenüber Störungen zu stärken werden LDPC Codes (Error-Korrektur-Codes) und CRC Codes (Error-Erkennungs-Codes) verwendet. Diese drei Vertreter ihrer jeweiligen Code-Kategorien werden in verschiedenen Konfigurationen getestet, um ihren Effekt allein oder untereinander auf die Datenübertragung, in Bezug auf Fehlerraten und Dekodierzeiten, zu testen. Es stellte sich heraus, dass mit Hilfe eines Fountain Codes und eines Error-Erkennungs-Codes sehr niedrige Error Raten erzielt werden können, bei denen gleichzeitig die Rechenintensität möglichst niedrig gehalten wurde. Für noch bessere Fehlerraten kann ein Fehler-Korrektur-Code hinzugefügt werden, welcher allerdings die Dekodierzeit, bzw. die Rechenintensität erhöht.

1 Introduction

1.1 Motivation

Living in the north of Germany I was always fascinated by lighthouses and their almost seemingly out of date character. A relict of the past still patrolling the coastal shores, waiting for a purpose.

They have been around since roughly 2200 years and were once the only way to guide sailors safely to the main land. The first lighthouse is said to be constructed around 280 BC on the island of Pharos and found its demise around the 14th century. Modern lighthouses may be headed in the same direction. Although the German coast holds an approximate of 220 lighthouses, many got faced out. Some got a push towards newer times through retrofitting LED lights instead of the old and power-hungry halogen lamps. To say it harshly though: They are a dying technology.

A lot of modern technologies like VHF (very high frequency) radio, satellite communication or GPS replaced the usage of the regular lighthouse to guide one in the dark. Their capabilities far outnumber the lighthouse's. GPS is a source of positional data around the globe while satellites and even regular mobile data networks like LTE can provide access to the internet.
In this thesis I want to explore the possibility of enhancing the already existing lighthouse infrastructure without huge constructional needs. The before mentioned retrofitting of LED lights can be used to our advantage in transmitting data through VLC (visual light communication) to ships. Even though lighthouses are old, they are reliable. Modern technologies are not failure resistant. Implementing data transfer through modulating the light of a lighthouse adds redundancy and therefore safety for sails people to avoid the dangers of sharp cliffs or unforeseen weather events.

More specifically, I want to explore how data can be sent to multiple receivers at the same time while eliminating the need for a back-channel. In the process creating a broadcast environment in which sender and receiver do not have to be time synchronous, meaning each can operate independently.

1.2 Goal and Structure of Thesis

The thesis is structured into three main parts.

First off, I am going to present necessary knowledge regarding the various topics this thesis is about. The main building blocks which are later going to be used in the simulation are given, explained and important aspects, for following use, emphasized. A goal would be to enable readers of varying expertise levels to be able to understand and learn about the concepts. The second part is going to explain in further detail what this thesis wants to research and how my approach differs from already existing methods used in the maritime scenario. Here I am going to contextualize the ensuing conception phase of the simulation. Thirdly we will see the accumulation of previously gained knowledge in the simulation and its results. Which will at last be put into context of said maritime setting.

2 Laying a Foundation

2.1 Informative lights

It is not easy to communicate with participants on the sea. Which does not take away from the importance to do so! Some lights are fixed on the ships themselves, some lead to way into a safe port, while others warn from impeding dangers like sand dunes or sharp underwater rocks. To recognize a light and match it with one on a map, lights use different rates of flashing, color or added audible clues. The following wants to give a brief overview of various communication lights used in the maritime setting.
Beginning at the ships themselves and their need to identify what kind of vessel it is and what direction it is heading. That information is mostly helpful for other participants to get an idea about possible speeds, size and before mentioned direction a vessel may be travelling. Otherwise it would be really hard to tell any of those three things during a low visibility day or simply at night. As seen in figure 2.1, it is really easy to tell, just by recognizing the previously learned lighting patterns, whether a vessel is moving away or towards you. Different vessel have to attach varying configurations of lights, depending on size or propulsion method.

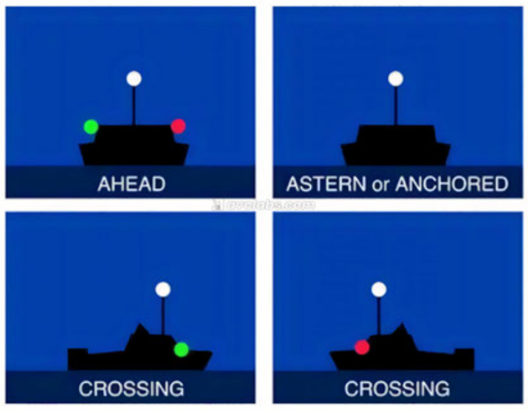


Figure 2.1: Different vessel orientations adopted from (*Sea Rules of the Road: The Display of Light & Audio Signals*, 2009)

Now heading to signals from shore to ships, so called leading lights, lead the way through treacherous canals or narrow bay entrances. They work by installing two lights in close proximity, but differing elevation right next to each other. Positioning them in a way, that when a ship is on the right path, one light appears to be right above the other. As a result of widening the Elbe in Hamburg, Germany, new leading lights, as seen in figure 2.2, had to be build in accordance with the newly established shipping route. An interesting video about the topic can be found here (HamburgPortAuthority, 2020).

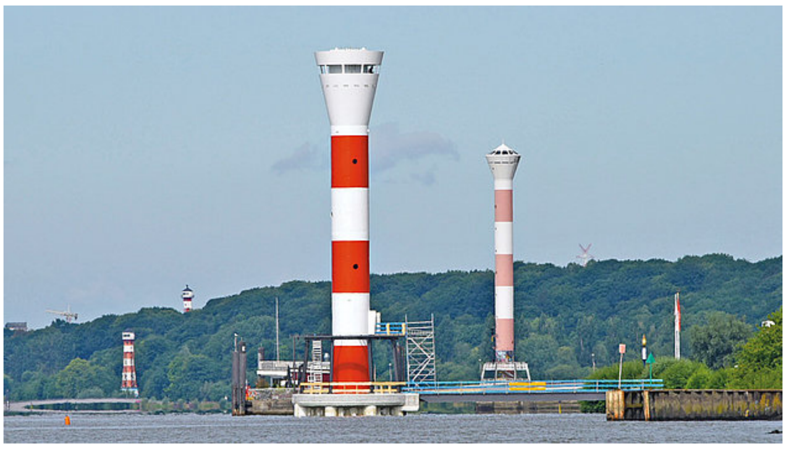


Figure 2.2: Two leading lights near Hamburg adopted from (*„Leuchtturmprojekte“ an der Elbe*, 2022)

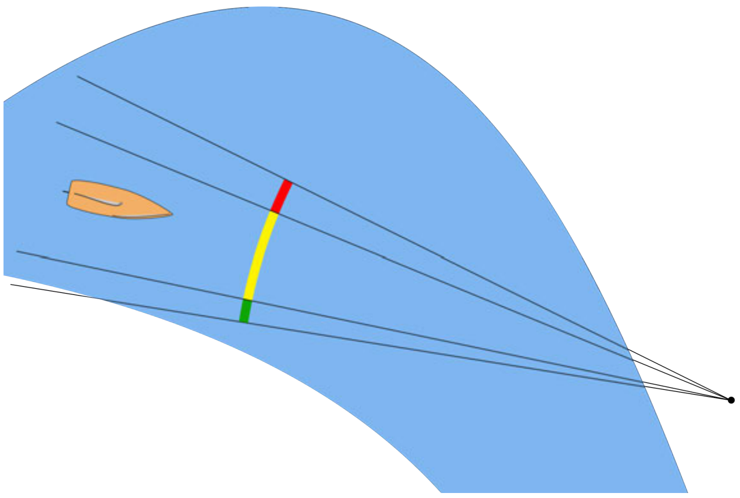


Figure 2.3: Explanation of a sector light

Sector lights are structurally and in their purpose similar to leading light towers, but work with a different method in mind. Instead of using two beacons and their respective location, sector lights use different colors displayed at different angles to indicate being on track or off course. Figure 2.3 depicts white/yellow light for correct angles in respect to the light source, while green and red signal port or starboard edge of the channel. Confusingly, the positions of red and green are polar opposites in USA standards compared to International ones.

The most prominent example of light being used in the maritime environment are however lighthouses. A single, really tall tower emitting a strong light, marking dangerous spots along the coastline. They can help add landmarks to an otherwise featureless shore during the day or simply state were shore is at night. Like mentioned earlier, lighthouses as well as the previously talked about guiding lights, emit their light in varying patterns. It is not enough to just see the light, the sails person needs to identify which lighthouse it is in order to know what it wants to warn about. Since additional information is needed to look that up, one could imagine the usefulness of a self identifying lighthouse using the already in place light source as a means to transmit that data.

Those same blinking light patterns result in a challenging transmission scenario for conventional systems in which the receiver and sender are always continuously talking with each other. A rotating lighthouse or heavy sea states may also disrupt the connection between sender and receiver. Chapter 4 will propose the use of Fountain Codes through which the receiver can collect packages whenever it eventually sees the light.

2.2 Visual Light communication VLC

"One of the key motivations is the fact that light can be used simultaneously for illumination as well as for communication and/or positioning purposes. [...] Endeavour to replace outdated light sources by LEDs can be combined with VLC technology. Compared to radio-based Wi-Fi, light based data transmission systems - dubbed Li-Fi if fully networked - offer distinct features: they [...] provide higher data security on the physical layer, and permit low-cost hardware components." (Hoeher, 2019).

Höher further states that light has always been used for data communication, although be it at very low data rates. From the signalling towers along coastal shores of Corsica, warning about the arrival of pirates, to semaphore systems reaching up to 550 km between Berlin and the Rhine Province. The idea if transmitting data over the light of a lighthouse blends in nicely with the historical context. We as humans can only see a small portion of the electro magnetic radio waves. Since the wavelength of visible light is much, much shorter than traditional radio waves, the resulting available bandwidth is therefore much wider. 5G networks operate between 300 MHz and 3 GHz, found here in figure 2.4 on the left hand side in the radio section. Visible light on the other hand has frequencies between 420 and 770 THz.

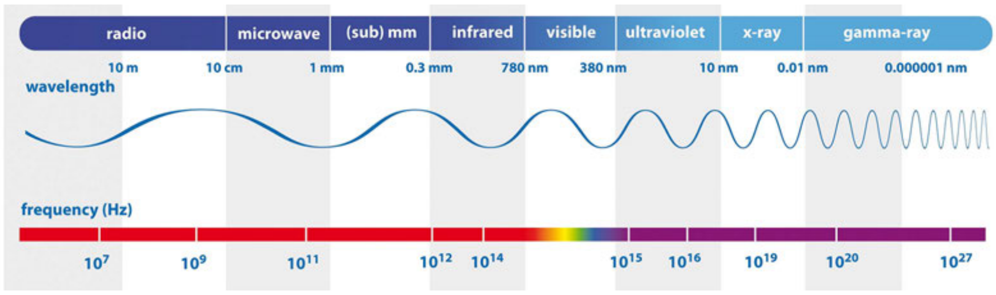


Figure 2.4: Spectrum of visible light adopted from (*Light: Electromagnetic waves, the electromagnetic spectrum and photons (article) | Khan Academy*, 2022)

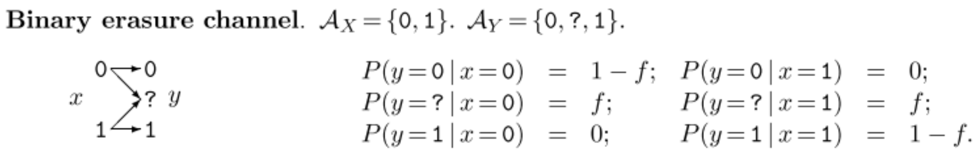
Modulating the light, adding a data signal to these available frequencies, can be achieved through different means. Generally speaking a signal is being converted, depending on the modulation scheme, to an analogue waveform which is than fed to the light source through a driver circuit. The receiver collects the photons with a photo detector and hands the resulting signal to the demodulation process (Hoeher, 2019).

2.3 Channels models

Depending on type of signal, the channel model can be an analog or a digital one. Channel models are used to guess and approximate real world phenomena to base correcting measurements upon. They help us to be prepared for any interference a signal, analog or digital, may receive. Generally speaking a channel model describes how an input gets changed in some way, resulting in a different output. In this thesis we are going to talk about the binary erasure channel (BEC, digital) and the binary input additive Gaussian white noise channel (BIAWGN, analog).

2.3.1 BEC

A binary erasure channel transmits a bit (zero or one), while the receiver sees the correctly transmitted bit or no bit at all, meaning the bit got erased. The following figure shows the error probability f in respect to all possible outcomes through the channel.



Binary erasure channel. $\mathcal{A}_X = \{0, 1\}$. $\mathcal{A}_Y = \{0, ?, 1\}$.

$$\begin{array}{lcllcl} P(y=0\,|\,x=0) & = & 1-f; & P(y=0\,|\,x=1) & = & 0; \\ P(y=?\,|\,x=0) & = & f; & P(y=?\,|\,x=1) & = & f; \\ P(y=1\,|\,x=0) & = & 0; & P(y=1\,|\,x=1) & = & 1-f. \end{array}$$

Figure 2.5: Binary erasure channel adopted from (MacKay et al., 2003)

If the probability of an erasure is $f = 0.30$, 30%, then the probability for a correctly received packaged is $1 - f = 1 - 0.30 = 0.70$, 70%. This is applicable in figure 2.5 for the transmission of a 1 or 0.

2.3.2 White Gaussian noise

Gaussian noise describes noise with the characteristic of a probability density function equal to the normal distribution. Meaning the noise values or amplitudes are Gaussian-distributed.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{2.1}$$

Gaussian white noise (GWN) has the same probability density function as Gaussian noise but now each value is also statistically independent from each other, resulting in a specific value regardless of the ones before or after (they are uncorrelated). This results in a constant or flat spectral density in the frequency domain. Meaning the power of the signal is the same regardless of time or frequency frame.

2.3.3 BIAWGN channel

The binary input additive white Gaussian noise channel, much like the additive white Gaussian noise channel uses white Gaussian noise to model real world interference. The only difference between the two models is, that the BIAWGN channel specifically models for a binary input.

$$P(Y = y|X = +1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(y-1)^2/2\sigma^2} \tag{2.2}$$

$$P(Y = y|X = -1) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(y+1)^2/2\sigma^2} \tag{2.3}$$

In his lecture about BIAWGN channels Tomáš Filler explains, that the channel "accepts a realization of a random variable $X \in -1, +1$ on its input and outputs a realization of a random variable $Y = X + Z$, where Z is a zero-mean Gaussian random variable with variance σ^2."(Filler, 2009). Zero-mean referring to μ being 0 like previously mentioned in Section 2.3.2 regarding formula 2.1.

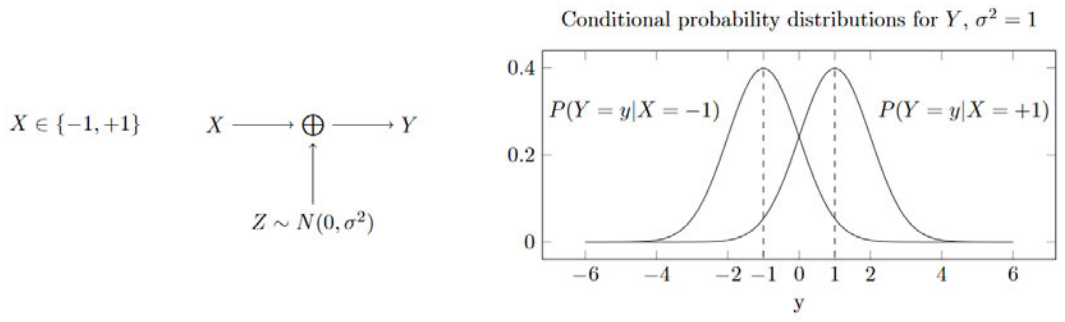


Figure 2.6: Binary input additive white Gaussian noise channel adopted from (Filler, 2009)

2.3.4 Signal to noise and bit error ratio

SNR also called signal to noise ratio is often depicted as E_b/N_0. It describes the relation between the input energy and the noise energy apparent in any given channel. It's usually shown in decibels (dB), $10\log_{10}(E_b/N_0)$. In figure 2.8, we can see the curve for a binary/quad phase shift keying. A binary phase shift key can transmit 1 bit at a time while a quad phase shift key can transmit 2. The curve tells us, that at an SNR of $4dB$ for example, we can expect about 1 error in 100 bits. Whereas at an SNR of $12dB$, that drops to 1 error in every $100th$ million bit. The ratio between corrupted and sent bits is the bit error ratio usually shown in a logarithmic scale.

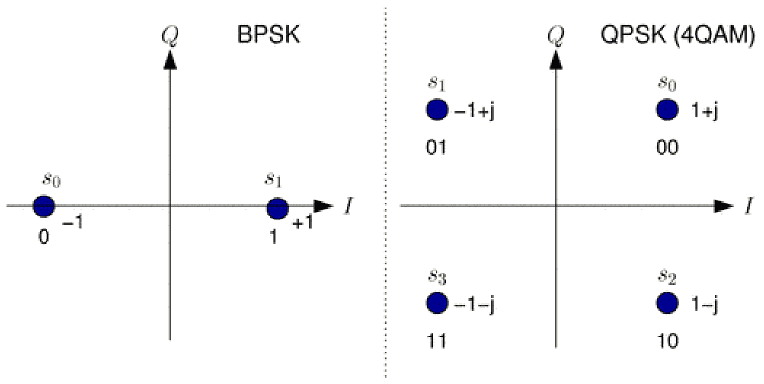


Figure 2.7: Binary phase-shift keying (BPSK) and Quadrature phase-shift keying (QPSK) as shown on the real and imaginary axis.

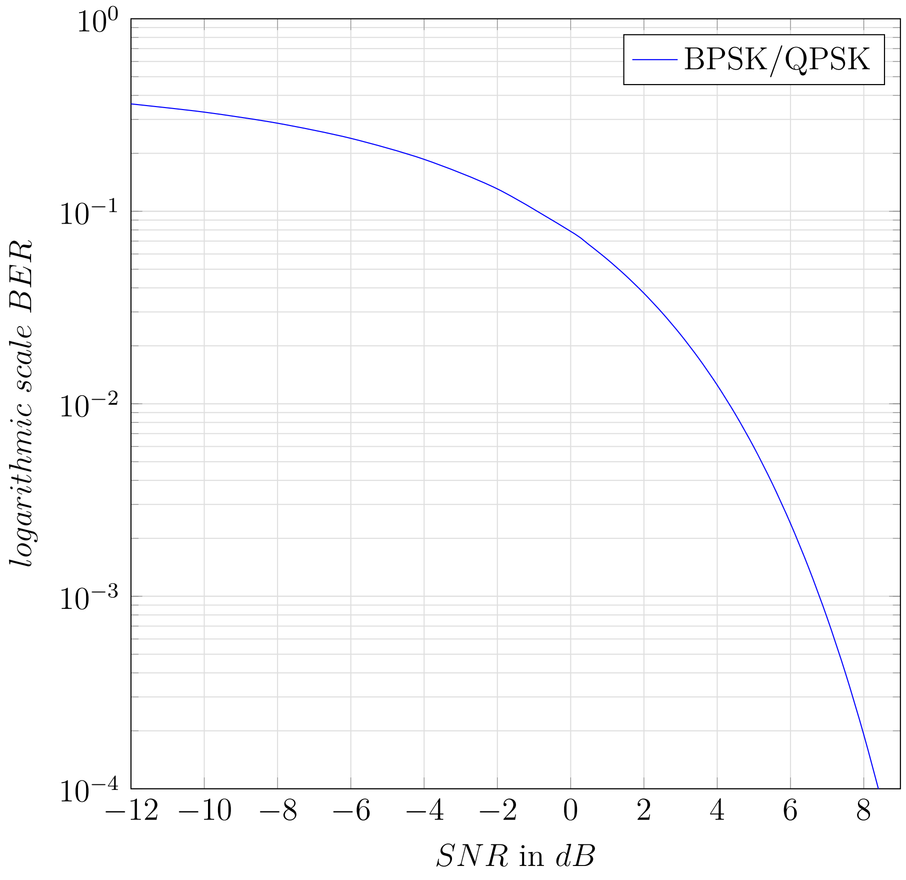


Figure 2.8: Bit error ratio over Signal to noise ratio

2.3.5 Channel coding theory

To tackle the naturally occurring errors while transmitting data a theory was created called channel coding. In it various different methods, to add redundancy to data bits, have been researched. The general idea is to give the receiver more information in order to recognize and decide if a transmission has been faulty. To transmit more data, the channel can either transmit for longer periods of time or increase the bandwidth, in the process shortening the symbols length. That would result in a reduction in channel dynamic and therefore higher bit error ratios. All those variables can be considered when designing a channel. A PCM speech transmission with a speed of 64 kBit/s has a BER of 10^-5 resulting in an error around every 1.56 seconds, which is for example deemed acceptable. One of the first successes of Channel coding was used in the Voyager probe and it's ability to sent data back to earth from around 4 billion kilometres away (Meyer, 1999).

2.4 Error Correcting/Detecting Codes

Different ways of correcting or detecting those errors occurring in data transmissions have been found over the years. To explain a few principles an example found on a blog post is being presented (*Linear Feedback Shift Registers for the Uninitiated, Part XV: Error Detection and Correction - Jason Sachs*, 2022):
Imagine a language containing three words: dry, imp and net. To add redundancy, the words are sent through a combination of three possible outcomes at each index. Letter 1 can only be d, i or n; Letter 2 only r, m or e and letter 3 can be represented as y, p and t. Instead of transmitting dry as 0, imp as 1 and net as 2, we now added redundancy to our encoder. A received symbol containing drt is recognized as incorrect and by calculating the distance d to the three valid words, we can determine that dry was most likely sent. The distance describes the combined change needed at every index to get a valid word. Drawing a graph where we include every possible received word and their distances to our three valid codewords, we get something that looks like figure 2.9.

The received words shown in the dotted line circles always have a distance of 1, meaning one letter has to change to get our valid words. For those words we can most likely attribute their respective valid words. The received words outside of the circles are ambivalent and not easily error corrected. They can only be detected as being faulty codewords. Summing it up, we can correct up to 1 error and detect up to 2.

We can now calculate the hamming distance h, which is the minimal distance of all distances in a code including only valid words, and get $h = 3$.
Inserting h the equation 2.4, we can see, that we can detect 2 or less errors in the above code. While with equation 2.5, we can see, that we can only correct up to 1 error.

$$h \geq e + 1 \text{ transformed to } h - 1 \geq e \tag{2.4}$$

$$h \geq 2e + 1 \text{ transformed to } \frac{h-1}{2} \geq e \tag{2.5}$$

With that knowledge at hand we can now evaluate some block codes.

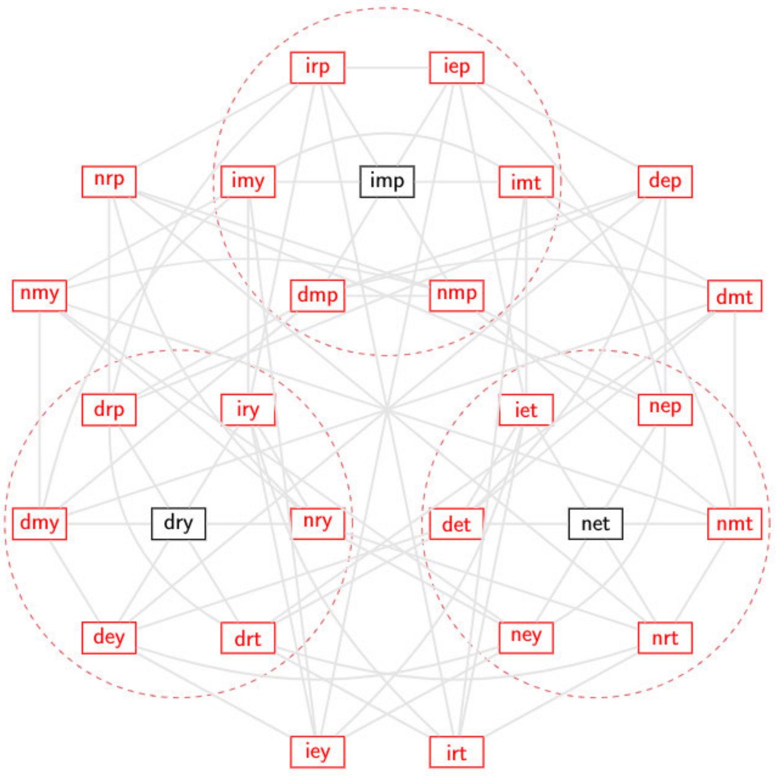


Figure 2.9: Graph showing the distances to received messages adopted from (*Linear Feedback Shift Registers for the Uninitiated, Part XV: Error Detection and Correction - Jason Sachs*, 2022)

2.4.1 Block codes

One of the simplest block code is the parity check code. It adds a parity bit to the original message indicating an even or uneven number of $1's$ in the message. It is achieved by XOR-ing the message bits. This example has $k = 2$ message or data bits, $n = 3$ total codeword bits and $n - k = 1$ parity bits, resulting in a code rate $R = k/n = 2/3$:

Message	Paritybit
0 0	0
0 1	1
1 0	1
1 1	0

Table 2.1: Creation of parity bits

If the receiver gets a codeword $\underline{c} = 010$, it can can immediately tell by the parity bit, that an error has occurred. The distance d between two valid codewords, for example 110 and 000 is 2. The hamming distance h for the above code would be 2, since the minimum distance from any valid codeword to another is at least 2. So the same evaluation principles in regards to error detecting and correcting capabilities as seen in section 2.4 apply.

The common notation for block codes is, to have a generator matrix [G] and a check matrix [H]. The valid words are depicted as $\underline{k}$ while the codewords are shown as a $\underline{c}$.
Our code is systematic, meaning the data/message bits k are included in the codewords before adding redundant bits $n - k$. Our four valid words are listed in table 2.1. To generate the generator matrix for a systematic block code, we have to have atleast $k = 2$ linear independent rows. It also has to consist of the identity matrix with size $k \times k$. Usually we have to solve the linear equations, but in this example it's quite easy since choosing the words 101 and 011 already gives us an identity matrix E^k.

$$\begin{gathered}[G] = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \\ [E^k] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } [G'] = \begin{bmatrix} 1 \\ 1 \end{bmatrix}\end{gathered} \tag{2.6}$$

Next we have to formulate our check matrix like so:

$$\begin{gathered}[H] = \begin{bmatrix}1 & 1 & 1\end{bmatrix} \\ [G'^T] = \begin{bmatrix}1 & 1\end{bmatrix} \text{ and } [E^{n-k}] = \begin{bmatrix}1\end{bmatrix}\end{gathered} \tag{2.7}$$

For block codes in which H has more rows and columns, and therefore more check nodes, a bipartite graph can help visualize which bits of the original message result in a check bit. Such a graph maps inputs to different, or in our example, single outputs. Our example has one check bit, easily noticeable since the parity check matrix H only has one row. The bipartite graph in figure 2.10 tells us, that all three bits of the original message are used to create the single check bit. Other bipartite graphs have more check bits with differing input variable nodes.

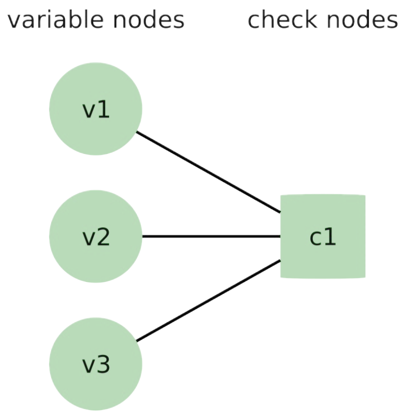


Figure 2.10: A bipartite graph for our single parity check example

By calculating the syndrome of a received word we can now check if it is part of the valid codewords, if it is $\underline{s} = 0$.

$$\underline{s} = \underline{c} * [H]^T \tag{2.8}$$

Here are all possible words we can receive and all valid words.

possible words	000	001	010	011	100	101	110	111
valid words	✓	x	x	✓	x	✓	✓	x

Calculating the syndrome of 011 and 001 by summing up (XOR) the also XOR'd elements of the vectors/matrices yields:

$$\underline{s} = [011] \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 0, \ \underline{s} = [001] \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 1 \tag{2.9}$$

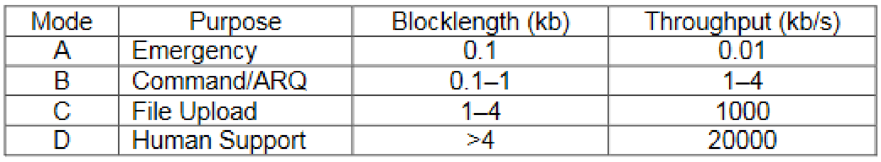
Meaning 011 is a valid word while 001 is not. Even tough you could have just checked the parity bit, I think this example explains why this notation can be helpful when it comes to larger block codes. For another example, please read (Meyer, 1999) chapter 4.3.2.2 about linear block codes.

2.4.2 Low density parity check code LDPC

In 1963 Robert G. Gallager set out to find an encoding algorithm that would work for the computational constraint given during that time. He elevated the parity check algorithm resulting in the low-density-parity-check code. Those error correcting codes are defined by the check matrix H to be of low density in regards to $1's$ and mostly filled it with $0's$ (Gallager, 1963). This way the decoding did not need a lot of computational power to succeed. To this day, they are widely used in WIFI standards like IEEE 802.11n and 802.11ac, Ethernet LAN and DVB (Beuth, Hanebuth, Kurz, & Lüders, 2001). Generally speaking, LDPC codes "are specified indirectly by an [n-k]-by-n parity-check matrix H consisting of [n-k] linearly independent rows. A coded sequence of n bits must satisfy all [n-k] parity-check equations corresponding to the [n-k] rows of H. An encoder maps an input frame of k=n-[n-k] information bits uniquely into a codeblock of n bits." (CCSDS, 2015). LDPC codes as implemented in the WIFI IEEE 802.11 standards are most effective at large block lengths. The three possible configurations of block lengths in this specific standard are $n = 648, 1296$ or 1944 (Mankar, Asutkar, & Dakhole, 2016).

2.4.3 Short block length LDPC

The CCSDS, short for the Consultative Committee for Space Data Systems, publishes so called Green Books in which they include collections of research covering a very broad range of topics in regards to telecommunication systems for space applications. The maritime transmission environment later described in this thesis resembles space in a lot of ways: "received signal power is correspondingly minuscule. Power efficiency remains a dominant priority. When a spacecraft emergency affects the telecommunications system, delivering any information at all can be difficult, and there must be a method to deliver short commands at an extremely low data rate when necessary." (Medova, Rybin, & Filatov, 2018). Therefore resulting in the need of short block length LDPC codes. The following figure 2.11 shows different modes and their purpose.



Mode	Purpose	Blocklength (kb)	Throughput (kb/s)
A	Emergency	0.1	0.01
B	Command/ARQ	0.1–1	1–4
C	File Upload	1–4	1000
D	Human Support	>4	20000

Figure 2.11: Four modes of operational uplink codes adopted from (Medova et al., 2018)

2.4.4 Cyclic redundancy check code CRC

Compared to the before mentioned regular, linear and systematic block codes, the error detecting CRC code can be calculated not by doing matrix calculations, but instead polynomial calculations (Meyer, 1999). The data bits are now represented as a polynomial:

$$k = 1001 \ , \ k(x) = 1x^3 + 0x^2 + 0x + 1 = x^3 + 1 \tag{2.10}$$

Evidently, the generator matrix turns into a generator polynomial $g(x)$. A generator polynomial is provided in the following example: $g(x) = x^3 + x + 1$. In order to generate the CRC check bits (parity bits) and therefore the codeword $c(x)$ we have to divide our data polynomial by the generator polynomial. This yields a quotient $z(x)$ and a rest $r(x)$ whereas we interpret $r(x)$ as the parity bits. The example in 2.11 starts with the fixed parameters of $k = 4$ and $n = 7$. The data polynomial is then prepared by left shifting it by multiplication with $b^{n-k} = x^3$. To better understand why we have to look at the bit representation of the resulting polynomial $x^6 + x^3$: 1001000. The purpose of this left shift by 3 $(n - k)$ positions is to make space for the parity bits to be added to the original message.

$$\frac{k(x)b^{n-k}}{g(x)} = \frac{(x^3+1)(x^3)}{x^3+x+1} = \frac{x^6+x^3}{x^3+x+1} = z(x) + \frac{r(x)}{g(x)} = x^3+x+\frac{x^2+x}{x^3+x+1} \tag{2.11}$$

$$c(x) = k(x)b^{n-k} + r(x) = x^6 + x^3 + x^2 + x \tag{2.12}$$

Figure 2.12 depicts the left-shifted data bits plus our rest $r(x)$ from our polynomial division. Choosing the right generator polynomial is vital to the codes error detecting capabilities. To generate one, primitive polynomials are chosen from a

tabular and if needed modified.
In order to detect an error at the receiver we need something similar to our parity check matrix in 2.4.1. CRC codes on the other hand use the fact, that by adding $r(x)$ to the data bits $k(x)$, we imprint $g(x)$ into our codeword. Meaning $g(x)$ is now a factor of all codewords $c(x)$. "Similar to the parity check example, where the receiver detects an error if the received vector has an odd number of ones, in this case the receiver will detect an error if g(x) is not a factor." (Höst, n.d.).
To see this more clearly, we have to represent the above presented equation a little differently by multiplying by $g(x)$ as shown in 2.13.

$$\begin{aligned} \frac{k(x)b^{n-k}}{g(x)} &= z(x) + \frac{r(x)}{g(x)} \\ k(x)b^{n-k} &= g(x)z(x) + r(x) \end{aligned} \tag{2.13}$$

"Equivalently, since addition and subtraction are identical in modulo 2 arithmetic, the codeword polynomial becomes [the multiplication of the generator polynomial with the quotient (equation 2.14)]"(Höst, n.d.). Now we can see, that if we were to divide $c(x)$ by the generator polynomial we would get a quotient $z(x)$ and no additional rest $r(x)$:

$$c(x) = k(x)b^{n-k} + r(x) = g(x)z(x) \tag{2.14}$$

Generally speaking, the received codeword $y(x)$ may have picked up an error during transmission. To check this we divide $y(x)$ by the generator polynomial $g(x)$.

$$\frac{y(x)}{g(x)} = z(x) + \frac{s(x)}{g(x)} \tag{2.15}$$

If the result yields a quotient with rest, an error has been detected. The rest polynomial is now called the syndrome $s(x)$. If the division concludes a rest of zero, no error has been detected. As we will later see, that doesn't mean no errors have occurred during transmission.

2.5 Fountain codes

A metaphor to describe fountain codes is the one of the digital fountain. Imagine a fountain being the sender, a bucket trying to catch the water the receiver, while the water being the packets to be received. The sender holds the bucket under the stream of water until the bucket is full resulting in enough packets, also called symbols n or s, received in order to decode the input symbols k or b.
The concept of a Fountain Code were first mentioned in "A Digital Fountain Approach to Reliable Distribution of Bulk Data" (Byers, Luby, Mitzenmacher, & Rege,

1998). They tried finding an alternative way of transmitting large amounts of data to multiple receivers. Prior methods used a back channel from the receiver to the sender in order for the receiver to ask for missing symbols. An increase of clients in a multicast in comparison to a unicast network can lead to an overloading of servers by too many request for missing packets. Many networks may also not have a reliable back channel, if at all.
Communication over satellite networks for example usually have high latency and limited capacity in regards to their back channel (Byers et al., 1998). Since the time period in where transmission is possible at all in extraterrestrial networks can be limited as well, it's especially advantageous to avoid sending redundant symbols. To achieve this, fountain codes can typically recover the original data set from $n = (1 + \epsilon)k$ received symbols. Where ϵ is the packet overhead, which can be as low as 3 % for $k > 100.000$ in LT codes, a member of the Fountain Code family (Mirrezaei, Faez, & Yousefi, 2014).

Fountain codes were first designed for binary erasure channels (BEC) where output symbols either get received correctly or lost in the process. Traditionally, in order to guarantee reliable communication, the channel may first be observed to measure it's erasure rate. Fountain Codes on the other hand are part of a class called rateless codes, meaning they can generate possibly infinite amounts of output symbols n given input symbols k. Here the rate of the code being k/n is not being predetermined by the sender probing the channel beforehand, but the receiver collecting a necessary amount of n in order to decode the original symbols k. Through this characteristic, the receiver dynamically adjusts to rate depending on the channels condition." (Mirrezaei et al., 2014). It may have to collect more output symbols n in order to decode k if the channel has a high erasure rate.

The authors from this article (Mirrezaei et al., 2014) also collected a lot of useful information regarding fountain codes over Noisy channels in their 5th chapter. One of those channels, the BIAWGN (binary input additive Gaussian noise), will become relevant later on.
This thesis wants to focus on comparing and studying the interaction between fountain codes, error correcting and detecting codes. Hence a more rudimentary handling of soft inputs has been chosen, which I am going to explain in further detail later on. In a future work I would like to implement the intricacies and underlying additional information of soft input symbols as a result of the BIAWGN channel in it's simulation.

2.5.1 About the LT code

LT codes where first published in "The 43rd Annual IEEE Symposium on Foundations of Computer Science" in 2002 by Michael Luby who previously worked with others on the general idea of a digital fountain in "A Digital Fountain Approach to Reliable Distribution of Bulk Data" (Byers et al., 1998). Since the LT code is a fountain code it inherits all it's properties and advantages. It is a rateless code that can generate as many symbols as needed, whenever needed. Any set from the encoded symbols is sufficient to decode the original data, as long as it is slightly larger than the data itself (Luby, 2002). That fact is immensely helpful in our transmission scenario since the receiver does not have to know when the sender started transmitting, it can just join in at any time and still be able to recover the data. It is also the first realisation of an universal erasure code, meaning the LT code is near optimal in erasure channels in regards to overhead and also very efficient with growing data lengths (Luby, 2002). In their paper "On the Performance of LT Codes over BIAWGN Channel" (Kharel & Cao, 2016) Amrit Kharel and Lei Cao investigated the performance of LT codes over BIAWGN channels. Their findings will not be implemented in this thesis' simulation, but are still very interesting and motivating to improve the simulation in a future works.

2.5.2 A deep dive into LT codes

In order to explain how LT codes work, we have to know that the encoder can be represented by using a bipartite graph much like the ones we know from block codes. Each symbol (packet), represented with an s, is being generated by XOR-ing a random number of data blocks with each one another. The number of packets being XOR'd can be quantified as the degree d of the symbol. $s1$ being of degree 1, $s2$ has $d = 2$, $s5$ has $d = 3$ and so forth.

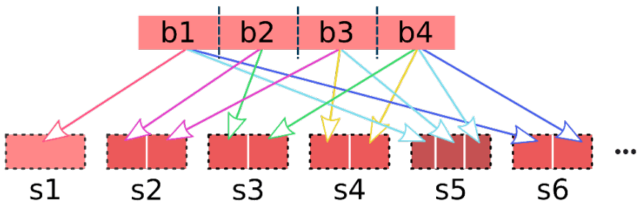


Figure 2.12: Bipartite graph of a LT code adopted from (*Introduction to fountain codes: LT Codes with Python*, 2019)

$$\begin{aligned} b1 &= s1 \\ b2 \oplus b3 &= s2 \\ b3 \oplus b4 &= s3 \\ b3 \oplus b4 &= s4 \\ b1 \oplus b3 \oplus b4 &= s5 \\ b1 \oplus b4 &= s6 \end{aligned}$$

Figure 2.13: Linear equation representation adopted from (*Introduction to fountain codes: LT Codes with Python*, 2019)

Like mentioned earlier, the number of symbols that can be generated are limitless since we can just keep combining any number of data blocks with each other to create more symbols. As seen for symbols $s3$ and $s4$, that doesn't mean newly generated symbols are entirely new. Since choosing input bits is based on random factors, repetitions can happen from time to time. The probabilities of choosing a specific number (the degree d) of data blocks to be XOR'd is manifested in the degree distribution. Its design is vital for an efficient LT code. More on that later. For now we have to understand why XOR-ing data blocks is such a great way to generate symbols.
Looking for example at symbol $s2$, which consists of $b2 \oplus b3$ and taking into account that XOR is reversible, we can recover $b2$ and $b3$ as follows: $b2 = b3 \oplus s2$, $b3 = b2 \oplus s2$.
Those equations only work out if we have $b2$ or respectively $b3$ in the beginning. To achieve this, the degree distribution has to include a number of symbols with degree one. Meaning they only contain information about one data block and can therefore be seen as equal $s1 = b1$.
Let's look at an example of a decoding process to understand it better:

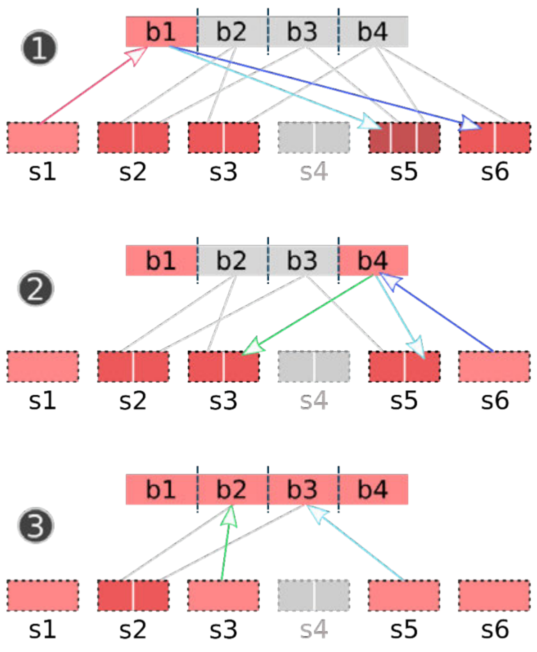


Figure 2.14: Decoding process adopted from (*Introduction to fountain codes: LT Codes with Python*, 2019)

Right away, symbol $s4$ got lost during transmission and is therefore greyed out. **Step 1** sees $s1 = b1$ and starts the process of reducing $b1's$ neighbours being $s5$ and $s6$. Reducing meaning to XOR $b1$ with its neighbours (new $s5$ = old $s5 \oplus b1$ and new $s6$ = old $s6 \oplus b1$). **Step 2** now sees $s6$ having a degree of 1 and starts the same process over resulting in $b4$ also being decoded and $s3$ and $s5$ being reduced. **Step 3** does the same thing with $b3$ and $b2$ hence finishing the decoding process and being able to decode the original message.

As we saw, the decoding process always needs at least one symbol of degree 1 at each step. If none can be found the decoding fails and the original data can not be recovered. To ensure a symbol of degree one at each stage Luby optimized the degree distribution and called it the robust soliton distribution. Luby described the set containing the to be processed symbols of degree 1 the ripple. At each step the symbols of $d = 1$ reduce their respective neighbours degrees before leaving the ripple potentially resulting in none, another or more symbols replacing it and joining the ripple to be processed. As mentioned before, the decoding fails if no symbols of degree 1 are in the ripple before all data blocks have been recovered (Luby, 2002). The robust soliton distribution tries to ensure a healthy ripple size, meaning at each step the ripple grows or stays the same as a result of symbols of degree 1 being processed, reducing their neighbours.

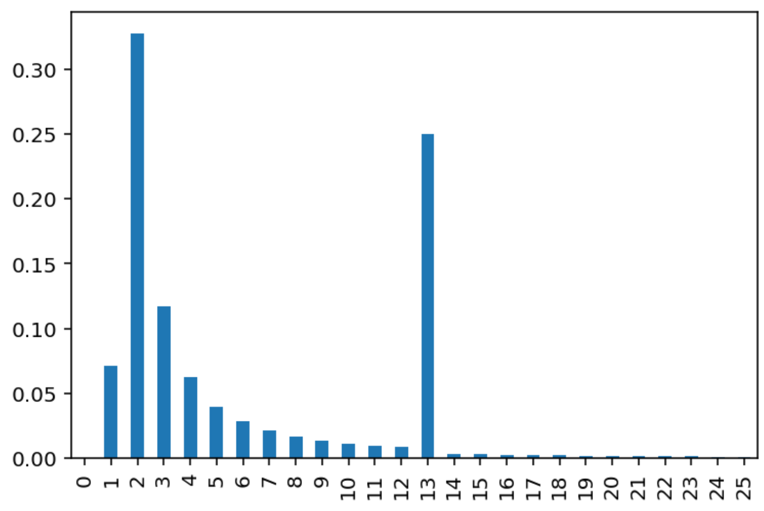


Figure 2.15: Graph of a robust soliton distribution adopted from (*Introduction to fountain codes: LT Codes with Python*, 2019)

Here we can see an example of such a distribution. The y-axis shows the probability of any degree being chosen at each encoding step resulting in the number of data blocks being XOR'd. On the x-axis we can see the corresponding degree to those probabilities.

3 Problem Analysis

3.1 Framing the question

This thesis tries to answer the following question:
How to transmit useful data to vessels along the sea shore, canals or the maritime setting in general while keeping economical as well as computational costs as low as possible.

The main goal would be to enhance already existing communication networks and build redundancy which in return strengthens the stability and security of the whole maritime communication system. It is always helpful to create options and systems to fall back onto in case of a failure in more capable and therefore more widely used ones. How can we warn sails people about already existing or spontaneous hazards? The UK government provides a list of potentially occurring problems ("Maritime safety: weather and navigation", 2016):

- damaged or broken lights, fog signals, buoys and navigational aids that affect main shipping lanes
- wrecks, reefs, rocks and shoals that might be a danger to main shipping lanes
- drifting hazards, such as derelict ships, mines, ice
- unexpected changes or closures of established routes
- cable or pipe laying, naval exercises or underwater operations that might be a danger for shipping lanes
- problems with radio navigation, radio or satellite maritime safety information services
- areas to avoid where search and rescue [...] activities are happening

Besides the hazards on water, vessels would also benefit from the transmission of meteorological data. Information about availability of port spaces before entering could be transmitted as well. The need for reliably communicating with ships is defiantly present and should be possible at all times. The next chapter tries to paint a picture of how we transmit data to vessels as of today and how we might supplement that in the future.

3.2 Gathering information

3.2.1 Maritime communication systems

Since there is no technology to "do it all", a variety of different systems work together in order to transfer data almost anywhere on the globe. In order to categorize the sea in respect to available communication systems, zones were defined ranging from A1 to A4.

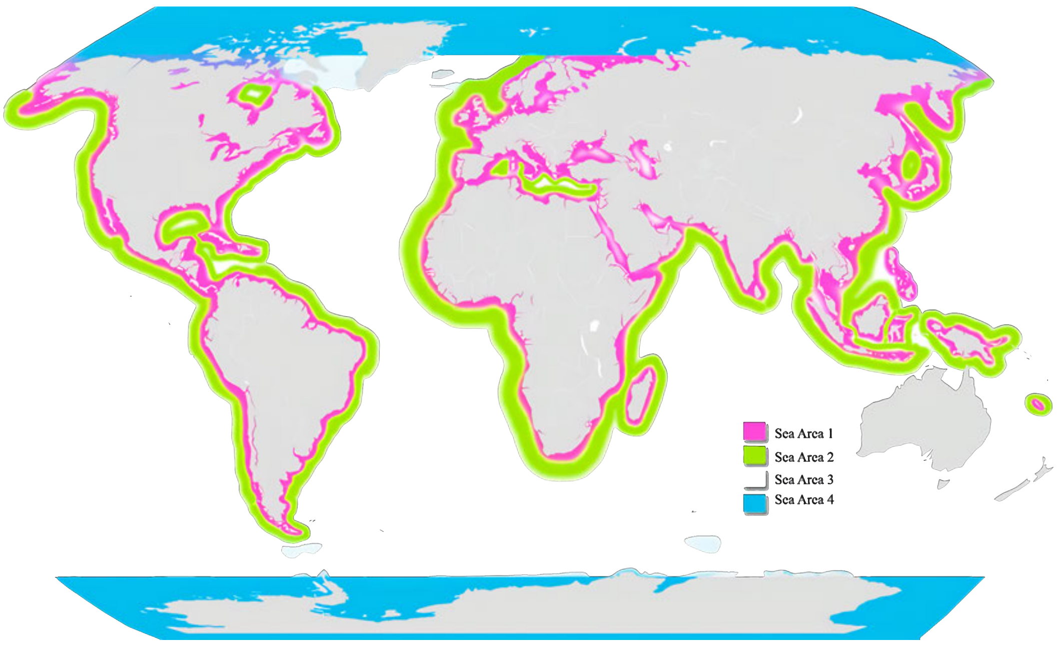


Figure 3.1: Transmission zones adopted from (*GMDSS Equipment carriage requirements for SOLAS ships. GMDSS Radio Survey Blog*, 2022)

Figure 3.1 depicts area 1 with a pink and area 2 with a green outline while the more generic zones called area 3 and 4 get represented by white and blue respectively. **Area 1** forms the area of about 20-30 nautical miles just of the shore. 1 nautical mile (nm) being 1852 metres. **Area 2** reaches up to 100 nm, while avoiding area 1. **Area 3** is basically the rest of the ocean up until 70 degrees north and south. **Area 4** only includes those remaining 20 degrees to the corresponding poles.

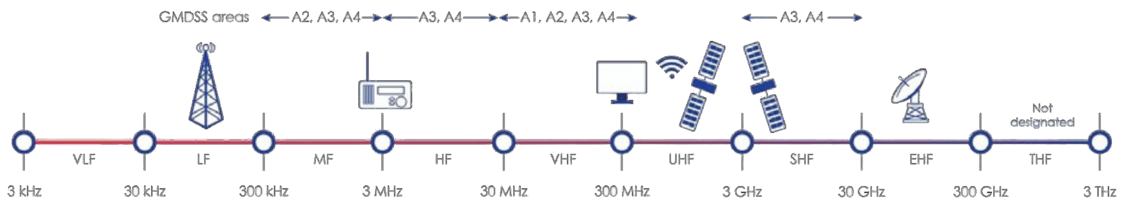


Figure 3.2: Transmission zones and their corresponding frequencies adopted from (*Marine Communication*, 2020)

Figure 3.2 compliments figure 3.1 since it shows which frequencies are being used for which zones. A3 and A4 are the only zones up at the SHF (super high frequency) spectrum. Those frequencies are used by satellite communication networks like INMASAT and COPSPAS-SARSAT. The first satellite system, INMASAT, can establish two way communication and is mandated while travelling in zone A3 (Rødseth, Tjora, Kvamstad, & Drezet, 2009). COPSAPS-SARSAT on the other hand is a one way only distress signal network. Ships traversing A4, being the most remote area on earth, need to carry a DSC-equipped HF radio-telephone. DSC stands for digital selective calling and allows to send distress signals.
Another useful tool for sails people is NAVTEX. It is a dedicated low-cost service for navigational and meteorological warnings and forecasts. While INMASAT is a carrier and capable of providing individualised message sending, NAVTEX is only used for the mentioned purpose. In their report "Ship-shore communication requirements" by (Rødseth et al., 2009) give a really good overview on maritime communication systems if you want to read more about that.

3.3 New approach

This thesis wants to explore data transmission in area 1, and the transition zones between area 1 and 2. Complimenting already existing networks in those zones by retrofitting various lighting sources like leading, or sector lights or simply lighthouses. The usage of wavelengths in the visible light section of the electromagnetic wave spectrum comes with advantages and disadvantages. Since the frequencies (see chapter 2.2) are really high in the visible light spectrum, a wide theoretical bandwidth can be achieved. A wider bandwidth can then lead to a theoretically higher throughput or higher resilience by shortening the transmission symbols (see 2.3.5). Since in our scenario we want to transfer data whenever we get the slightest chance to do it, a shortening of transmission symbols helps in delivering as much data as we can whenever possible.

A major drawback when using a VLC system, is that we need to see the light in order to transmit data. More extreme sea states can block the light from hitting the photo detector just as easily as simple weather events, like fog, can decrease the visible range. The simple blinking characteristic of some navigational lights inherently blocks the light from reaching the receiver. Since Fountain Codes can collect and process data whenever they get the chance, they are a perfect fit for a channel code in such a scenario. Originally designed for BEC channels, it is helpful to use a precode of error detecting and correcting codes in order to adapt to the BIAWGN channel given here. The transmission could now withstand very poor signal to noise ratios (SNR).

A large portion of ship traversing area 1 and 2 are small vessels with limited computational capabilities. The data transfer system should have a decoder that is working well within those constraints.

4 Conception

A simulation needs to be created, comparing different encoding schemes and integrating as much knowledge learned along the way. At the same time interpreting the resulting data with respect to real life facts about lights in maritime settings. The simulation should also aim to mimic real life in an abstract way. What channel model describes the scenario best? What kind of SNR and therefore errors do we expect? How can we best reduce or negate those errors by tweaking and combining detecting/correcting algorithms? The simulation should be able to answer those questions as well as enable us to make statements about the real life scenario or at least point future research in a general direction.

4.1 Channel model

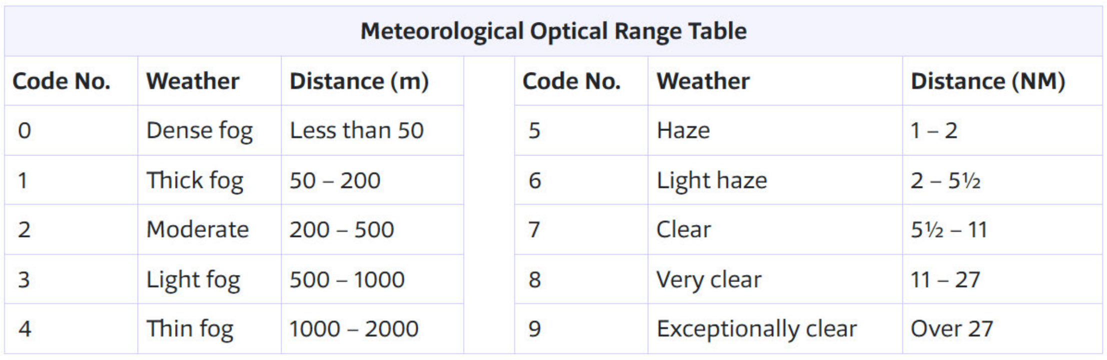
Since we are dealing with a direct line of sight wireless transmission that is relatively short in distance, an additive Gaussian white noise channel is sufficient in modelling the interference present from sender to receiver. Rayleigh and Rician fading channels would only be applicable for if the receiver experienced multiple reflections at different timings. Since there is a direct line of sight and no buildings to bounce the light source of from and create reflections, an AWGN channel was chosen. A future deep dive into this topic could look at light being reflected of waves onto the photo detector and see if that can cause interferences. In that case Rician fading, which still models a line of sight signal plus reflections, could be considered.

4.1.1 Modulation schemes

Section 3.1, Framing the question, stated that the communication should be made as robust and least affected by low SNR's as possible. Therefore we need to choose a modulation scheme were distances between bits are the furthest. If we add white gaussian noise, the change in position of a bit along the real axis is the lowest in respect to the other bit. Hence the binary phase shift key would be most optimal to use in our scenario. The light of a maritime signalling light, or that of any light for that matter, can not be less than 0 (off). Using BPSK is consequently impossible to implement. To solve this, we can add a DC offset to the BPSK and create a NRZ-OOK (non-return-to-zero on off key). "The DC offset sacrifices 50 % of the transmit power, [...]. Hence the power efficiency of NRZ-OOK is worse by 3dB"(Hoeher, 2019).

4.1.2 Effects on the SNR

The efficiency is directly tied to the visibility of the light source. The weaker the light hitting the photo detector, the lower the signal to noise ratio at the receiver. Two words need to be introduced in that context. **Luminous range**, "is maximum distance at which a given light signal can be seen by the eye of the observer at a given time, as determined by the meteorological visibility prevailing at that time."(Willemsen, 2022). Figure 4.1 shows how much different weather conditions can change the general visibility. The meteorological optical range table is defined by the World Meteorological Organization (WMO) and links weather events with the photometric luminosity function of the International Commission on Illumination (CIE). The photometric luminosity function is a reflection of our heightened or diminished perception of different wavelengths. In the table a distance is given at which a 2700 Kelvin collimated light beam is reduced to 5% of it's original strength (WMO, 2018).

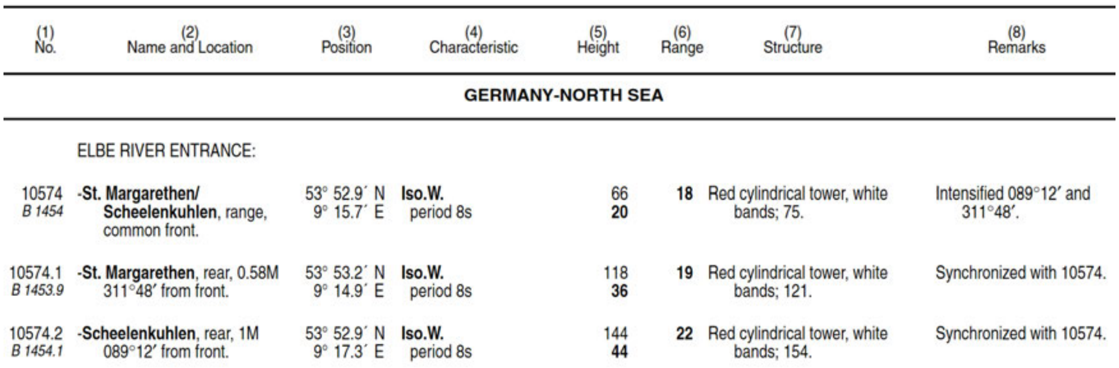


Meteorological Optical Range Table

Code No.	Weather	Distance (m)	Code No.	Weather	Distance (NM)
0	Dense fog	Less than 50	5	Haze	1 – 2
1	Thick fog	50 – 200	6	Light haze	2 – 5½
2	Moderate	200 – 500	7	Clear	5½ – 11
3	Light fog	500 – 1000	8	Very clear	11 – 27
4	Thin fog	1000 – 2000	9	Exceptionally clear	Over 27

Figure 4.1: Table depicting visibilities in different weather conditions adopted from (Willemsen, 2022)

Nominal range, is "the luminous range when the daytime meteorological visibility is 10 NM, which is 74% transmission. Nominal range is generally the figure used in official documentation such as nautical charts, lists of lights, etc. Nominal range assumes that the light is observed against a dark background, free of background lighting." (Willemsen, 2022). Equipped with that knowledge, we can consult a so called **list of lights**. Every country or general water body region has their own list in which major lights, used for navigational purposes, can be found. The one I found covers the whole coast of the British isles as well as for example the northern coast of Germany, the Netherlands and France.



(1) No.	(2) Name and Location	(3) Position	(4) Characteristic	(5) Height	(6) Range	(7) Structure	(8) Remarks
	GERMANY-NORTH SEA						
	ELBE RIVER ENTRANCE:						
10574 *B 1454*	-**St. Margarethen/ Scheelenkuhlen**, range, common front.	53° 52.9´ N 9° 15.7´ E	**Iso.W.** period 8s	66 **20**	**18**	Red cylindrical tower, white bands; 75.	Intensified 089°12′ and 311°48′.
10574.1 *B 1453.9*	-**St. Margarethen**, rear, 0.58M 311°48′ from front.	53° 53.2´ N 9° 14.9´ E	**Iso.W.** period 8s	118 **36**	**19**	Red cylindrical tower, white bands; 121.	Synchronized with 10574.
10574.2 *B 1454.1*	-**Scheelenkuhlen**, rear, 1M 089°12′ from front.	53° 52.9´ N 9° 17.3´ E	**Iso.W.** period 8s	144 **44**	**22**	Red cylindrical tower, white bands; 154.	Synchronized with 10574.

Figure 4.2: Exempt showing three lighthouse locations near Hamburg adopted from (*Pub. 114, List of lights, radio aids and fog signals*, 2022)

As seen in figure 4.2, our third lighthouse has a range of 22. Meaning it has a nominal range of 22 nautical miles. Let's say it is a lightly hazy to hazy day, then figure 4.1 informs us about a visibility of 2 nautical miles. Given this number is based on a nominal range of 10 nautical miles, not the 22 we got from our "List of lights".

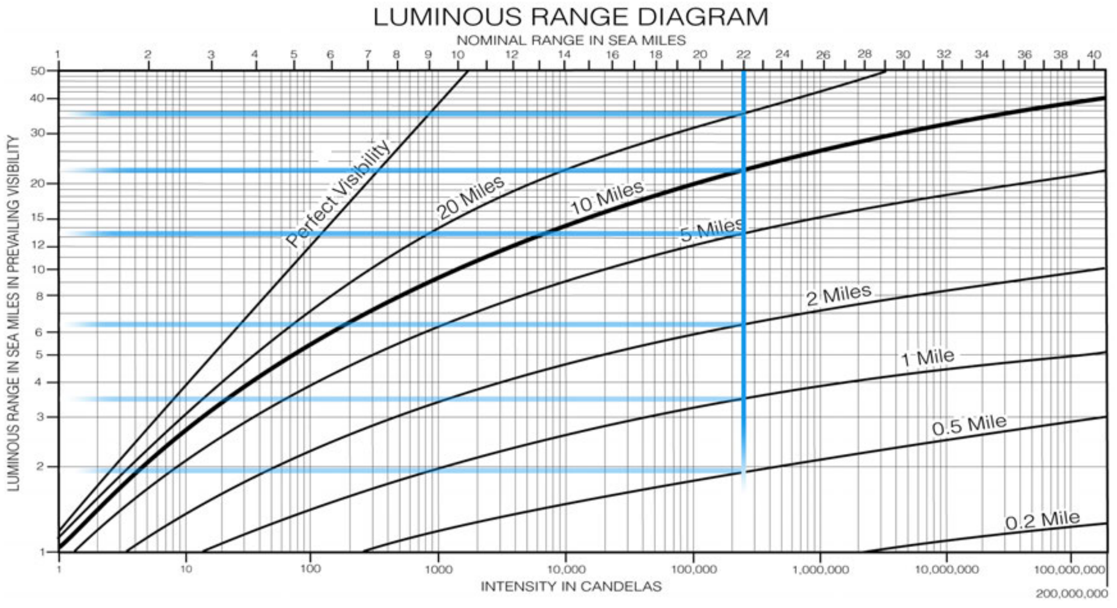


Figure 4.3: Graph depicting luminous ranges depending on nominal range and visibility adopted from (Willemsen, 2022)

We then have to look at figure 4.3, starting from the top, finding the nominal range of 22 on the upper x-axis, we can draw a line downwards and stop on the line that states a visibility of 2 nm. Drawing a line to the y-axis yields the actual luminous range of that specific lighthouse on that specific day: 6.5 nautical miles.

Using something called the Allards Law, as seen in equation 4.1, we can calculate the resulting visibility by inserting luminous intensity (I in candela), atmospheric transmissivity (T) and the threshold of illuminance (E in sea-mile candela). The latter variable refers to the ability of our eye to barely see a light given various background illuminations. A value of $E = 0.67$ corresponds to no background light, 6.7 sea-mile candela to minor background light and 67.0 to heavy background lighting (of Transportation, 1986).

$$E = \frac{I * T^D}{D^2} \tag{4.1}$$

Using this equation, we can calculate the nominal range of our 250.000 candela lighthouse. For nominal ranges an atmospheric transmission of 0.74 is assumed. After solving for the visible distance, we get: $D \approx 22.06$.

$$0.67 = \frac{250000 * 0.74^D}{D^2} \tag{4.2}$$

The point of this exercise was to emphasis on the ever changing and harsh conditions on sea that can have an effect on the signal to noise ratio. Another limiting factor regarding the visibility of a light can be something called the "Dipping range". Here the curvature of the earth blocks the view between lighthouse and vessel. An easy way to calculate the distance at which a vessel seems to be just about to dip behind the horizon is the following function: $2.08 \times (\sqrt{Elevation} + \sqrt{EyeHeight})$. Our example lighthouse has a height of 44 metres. A person standing on a ship at a height of 10 metres would experience this phenomena at around 20 nautical miles. Above this value, the lighthouse would no longer be visible. Attaching the photo detector as high as possible is a priority. We can also see through this example that the nominal ranges given in the "List of Light" are not always practical. The major limiting factor in our lighthouse example is the geographical and not the luminous range.

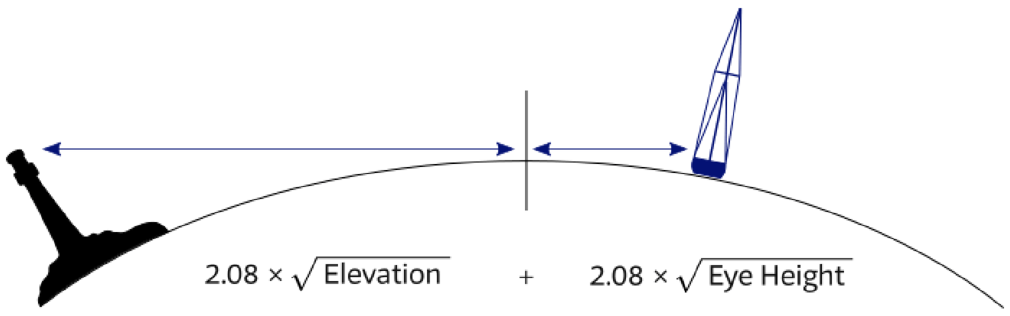


Figure 4.4: Simplified way to calculate dipping range

By using the information provided in a paper called "Joint Light Planning and Error-Rate Prediction for Dual-Use Lighting/Visible Light Communication" (Mietzner et al., n.d.), we can try to link the output of a lighthouse with a perceived signal to noise ratio at the receiver. We first need to calculate the expected Illuminance (E). Since we have a candela value given from our "List of Lights" via the luminous range diagram, we can calculate the Illuminance (E) for an example distance of 10 sea-miles as follows:

$$E = \frac{I}{D^2} = \frac{250000cd}{18520m^2} = 7.29 * 10^{-4} lx \tag{4.3}$$

The following list gives us the needed parameters in order to calculate the SNR γ_{el}.

- The average-power-to-squared-mean ratio is set to κ=2 both for (NRZ) OOK.
- For the blue core LED, a center wavelength of $\lambda_b = 450$ nm is assumed. The corresponding value of the eye sensitivity curve is given by $V(\lambda_b) = 0.04$.

- For the responsivity of the PD at this wavelength, a value of $R_\lambda(\lambda_b) = 0.28$ A/W is adopted from the data sheet of the Hamamatsu R S3590-08 PIN photodiode.
- The area of the test surface representing the VLC receiver is given by $A_{meas} = 100mm^2$.
- The matched filter bandwidth is set to $B_{MF} = 1$MHz in order to accommodate the whole visible light spectrum of an LED.

$$\gamma_{el} = \frac{P_{R,el}}{N_{R,el}} \tag{4.4}$$

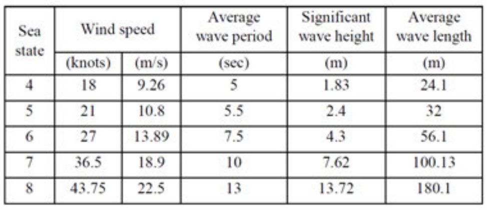
$$P_{R,el} = \kappa * (\frac{R_\lambda(\lambda_b) * A_{meas}}{683\frac{lm}{W} * V(\lambda_b)} * E_v)^2 \tag{4.5}$$

$$N_{R,el} = 2 * e_0 * R_\lambda(\lambda_b) * P_{R,amb} * B_{MF} \tag{4.6}$$

$$P_{R,amb} = \frac{A_{meas}}{683\frac{lm}{W} * V(\lambda_b)} * E_v \tag{4.7}$$

Resulting in a SNR $\gamma_{el} = 57$ dB.
If this were to be true we would see close to no errors in the transmission. The proposed scheme of using a Fountain Code is in this case still useful since it enables us to dip in and out of the ongoing transmission as described earlier. However, the signal to noise ratio of 57 dB was calculated without including information about different atmospheric conditions. Allards Law, as presented in formula 4.1, does not yield illuminance values in lux but instead something called sea-mile candela. The formula is helpful at linking perceived human eye sight with different weather conditions, but not applicable when it comes to exact light simulation values. The 57 dB could be read as a transmission during a perfectly clear night sky with zero atmospheric effect on transmission.

The before mentioned problem of rising and lowering waves blocking the Line-of-Sight path of lighthouse to photo detector was studied in the following paper. "Shore-to-Sea Maritime Communication with Visible Light Transmission" (H. Kim, Sewaiwar, & Chung, 2014) shows us that weather is not the only variable causing possible distortions at the receiver. They simulated the effects of different sea states 4.5, meaning different wind speeds or wave heights, on the bit error ratio of a maritime visible light transmission using LED's and photo diodes.



Sea state	Wind speed		Average wave period	Significant wave height	Average wave length
	(knots)	(m/s)	(sec)	(m)	(m)
4	18	9.26	5	1.83	24.1
5	21	10.8	5.5	2.4	32
6	27	13.89	7.5	4.3	56.1
7	36.5	18.9	10	7.62	100.13
8	43.75	22.5	13	13.72	180.1

Figure 4.5: Sea state parameters adopted from (H. Kim et al., 2014)

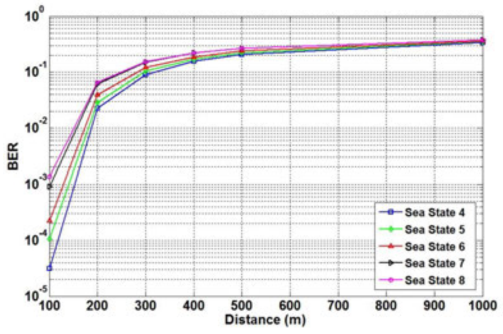


Figure 4.6: Performance analysis of an uncoded transmission adopted from (H. Kim et al., 2014)

Figure 4.6 shows us BER's of around 10^{-5} to 10^{-3}. In order to sill have reliable data transmission, we have to encode the data with different strategies that are going to be presented in the following chapter.

4.2 Comparison and Symbioses of Encoding Methods

This thesis ultimately wants to explore the interactions between error detecting and correcting codes in relation to fountain codes for BIAWGN channels. Because of that a rudimentary block diagram is proposed to visualize the basic set up. On the left of the added white Gaussian noise the data gets encoded while to the right decoded. The reasoning behind the order of codes will be explained in this chapter. An outer error detection code (CRC code) is used to firstly encode the data. After that the middle error correction code (LDPC code) is added. The final inner Fountain code (LT code) is the center piece in the encoding scheme. The LDPC as well as the CRC decoding works in symbioses with the LT code as indicated by the feedback-loop at the decoder. Through which the CRC decoder advises the LT decoder to collect more packages until enough CRC checks have been successful, resulting in a decoded message. The variable k bits correlates to the input message bit length while n1, n2 and n3 depict the resulting bit lengths of a package after the respective encoding steps.

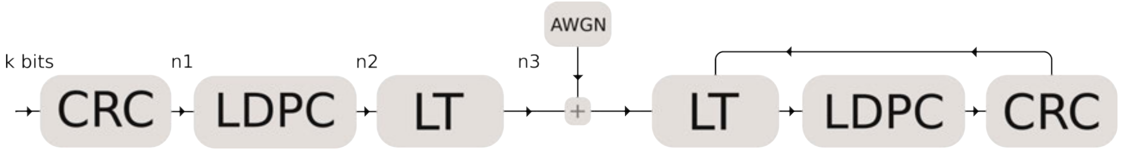


Figure 4.7: Basic coding scheme

4.2.1 Encoding order

Before settling on the above proposed encoding order of CRC into LDPC into LT encoding, others were thought about. In order to have a decoder that eliminates the most errors possible, it makes sense to first correct errors (LDPC code) before checking the package for remaining errors (CRC code). Deciding on the remaining order of encoding and therefore decoding was really challenging. I eventually settled on the order seen above in which we have an inner LT code followed by the combination of error correction and detection. This chain made the most sense at the time of writing the simulation code, since it integrated the feedback loop caused by the CRC decoder nicely within the while loop of the LT decoder (see section 5.2). It also enabled the sender and receiver to work independent of one another, meaning no time synchronisation was needed. The idea of having a precode before the LT code was also used in the development of Raptor codes, a powerful and newer version of LT, Fountain Codes.

With the thought in mind that the LT code was originally designed to operate in a BEC and not a BIAWGN channel, a strong case could be made for a LT $\rightarrow$ CRC $\rightarrow$ LDPC chain. This way the LDPC and CRC code could theoretically simulate an erasure channel by ultimately erasing the faulty packages before presenting the remaining to the LT decoder. We would not have an intuitive feedback loop based on the CRC decoder decision though.

A paper researching the use of "LT codes for Wireless Broadcast" employed a system in which they placed the LT code between the CRC and a channel block code (CRC $\rightarrow$ LT $\rightarrow$ channel code) (Jenkac, Mayer, Stockhammer, & Xu, 2005). That way they still made use of the feedback loop created by the CRC check while still minimising the amount of errors the LT decoder experiences through the use of a channel block code.
Another paper about a "Asynchronous Multi-Relay System" (Huang & Lei, 2013) employed a structure similar to my chosen one: CRC $\rightarrow$ Raptor code $\rightarrow$ LDPC code. The LDPC code in my encoding method is here replaced by a pre-code before the LT code resulting in something known as a Raptor code. The LDPC

code in the papers encoding chain is instead used as a channel code after Raptor encoding. The CRC encoder works in the same fashion as in my proposal, by removing received faulty packages.
The "Application Layer Forward Error Correction for Mobile Multimedia Broadcasting Case Study" also used a CRC code preceding a Turbo code in their PHY layer portion of the MBMS standard.

The usage of an outer CRC code is pretty common in a lot of papers and standards adopting Fountain Codes. As seen in table 4.1, two of 4 examples use some sort of channel code as their inner most encoding method. By using a Raptor code, they automatically pre-code an LT code, resulting in something that resembles my set up. From what I could gather in my literature research, a pre-code before the LT-code as well as some form of channel code coupled with an outer CRC code would have been a valid and interesting encoding chain also.

Paper reference				
EC	ED	FC	Order	Channel/OSI
Fountain Codes Design for Asynchronous Multi-Relay System (Huang & Lei, 2013)				
pre-code, LDPC	CRC	Raptor	CRC - Raptor (pre-code, LT code) - LDPC	BEC
MBMS Bearer Service over UMTS (Stockhammer, Shokrollahi, Watson, Luby, & Gasiba, 2008)				
Turbo	CRC	Raptor	Raptor - CRC - Turbo	PHY/Appl layer
Soft Decoding of LT-Codes for Wireless Broadcast (Jenkac et al., 2005)				
channel code	CRC	LT	CRC - LT - channel code	AWGN
Design Of Fountain Codes With Error Control (Kharel, 2013)				
Turbo		LT	Turbo - LT	BIAWGN

Table 4.1: EC = Error-correcting code, ED= Error-detecting code, FC = Fountain code

4.2.2 Encoding with LT Code

This quote back from chapter 2.5 about the rate of a fountain code is really important now: "Here the rate of the code being k/n is not being predetermined by the sender probing the channel beforehand, but the receiver collecting a necessary amount of n in order to decode the original symbols k. Through this characteristic, the receiver dynamically adjusts to rate depending on the channels condition" (Mirrezaei et al., 2014).
The fundamental idea of using a fountain code in a maritime setting comes back to the necessity and reliability and therefore a main attribute of a lighthouse. It's usually always on, hence available to transmit data. Ships should be able to just receive the signal whenever they can and decode it. Changing weather events can result in a varying degree of signal to noise ratios, so it is beneficial to employ a rateless code that can react to the channels condition on the fly. Since ships pass lighthouses rather slowly, the receiver has a lot of time to collect enough packages for decoding.

Earlier in chapter 2.4.3, about short length LDPC codes, I compared space travel and it's challenges to ship to shore communication and complications. Both environments share the hostility, empty space/sea, as well as great distances between the crew and earth or the main land. Another similarity can be the really low data rates, caused by weather events. Satellite communication can be hindered by weather occluding the area between the radio dish and the satellite, "[...]delivering any information at all can be difficult, and there must be a method to deliver short commands at an extremely low data rate when necessary." (Medova et al., 2018). In the maritime setting, weather can also have a huge effect on the visibility of the light source as seen in 4.1.2. The need for short messages holds true for the maritime setting as well in case of an emergency. Table 9 in (Rødseth et al., 2009) shows various types of communications and their data requirements. A lot of those are non emergency applications but still require fairly low data rates of 0.01 to 0.2 kbps.

The result of this being that we have to be able to transmit small packages, therefore have an LT code that can satisfy those requirements by having a small block length. Section 4.2.4 will present a short LDPC code that has to work hand in hand with the LT block length of 64 suggested here. 64 bits were chosen while experimenting with the simulation on my laptop. It could not handle file large file sizes, resulting in me choosing a small package size in order to be able to simulate small file sizes. I later learned that an LT code could also work based on bits instead of bits in packages. If I were to restart the thesis, I may have tried to code such an LT code to drive the simulation file size further down. A more detailed break down of the set up will be shown in chapter 5 "Implementation".

The following codes (CRC and LDPC) try to improve and supplement the characteristics of the LT Code. The CRC code tries to exploit the undefined rate while the LDPC code aims to improve general performance in regards to the bit error ratio.

4.2.3 Encoding with CRC

Since a CRC code can detect a varying amount of errors depending on its design, we can employ it in collaboration with the LT code. By being able to basically throw away bad packages, we can use the inherent undefined rate of the LT code to our advantage. As seen in figure 4.7, we want to add the CRC code right in-between the LT and LDPC code. After decoding the packages with an LDPC decoder, the CRC decoder checks for remaining errors. We also want to compare an overall encoder version where we skip the LDPC encoding and only encode with a CRC encoder. Regardless of the LDPC encoding step, the CRC encoder adds 8 bits to the LT encoded messages with a size of 64 bits yielding 72 bits. If the CRC decoder finds an error, the resulting package is disregarded and thrown into the void, never able to be decoded by the LT decoder. The most important aspect of this partnership would be the amount of errors the CRC code can detect.

In his course about error detecting using a cyclic redundancy check code, Stefan Höst provides a formula to choose a primitive polynomial, as mentioned in 2.4.4, depending on error correcting needs (Höst, n.d.). In order to detect any error combination containing 1 or 2 changed bits, we have to choose the primitive polynomial as shown below. Additionally, in order to detect error combinations containing 3, 5, or generally all errors containing an uneven number of inverted bits, we can multiply the resulting primitive polynomial by $(x + 1)$.

$$k + L < 2^L - 1 \tag{4.8}$$

k is the length of data bits while L the degree of the primitive polynomial. For a signal with data length $k = 64$ we can check two primitive polynomials $L = 6, 7$ and see if they would be sufficient:

$$\begin{aligned} 64 + 7 &< 2^7 - 1 \qquad & 64 + 6 &< 2^6 - 1 \\ 71 &< 127 & 70 &< 63 \end{aligned} \tag{4.9}$$

$p(x)$	$p(x)$
$x^2 + x + 1$	$x^{10} + x^3 + 1$
$x^3 + x + 1$	$x^{11} + x^2 + 1$
$x^4 + x + 1$	$x^{12} + x^6 + x^4 + x + 1$
$x^5 + x^2 + 1$	$x^{13} + x^4 + x^3 + x + 1$
$x^6 + x + 1$	$x^{14} + x^{10} + x^6 + x + 1$
$x^7 + x^3 + 1$	$x^{15} + x + 1$
$x^8 + x^4 + x^3 + x^2 + 1$	$x^{16} + x^{12} + x^9 + x^7 + 1$
$x^9 + x^4 + 1$	$x^{17} + x^3 + 1$

Figure 4.8: List of primitive polynomials adopted from (Höst, n.d.)

The results indicate a primitive polynomial of degree 7 or higher to be sufficient. Höst also provides a list of well known CRC generator polynomials, see figure 4.9. Equipped with the knowledge from my previous calculation, I chose the CRC-8 polynomial. My goal was to use as many CRC bits as necessary for my scenario while using the least amount possible.

$$g(x) = x^8 + x^2 + x + 1 \tag{CRC-8}$$
$$g(x) = x^{10} + x^9 + x^5 + x^4 + x^2 + 1 \tag{CRC-10}$$
$$g(x) = x^{16} + x^{12} + x^9 + x^5 + x^4 + x^2 + 1 \tag{CRC-16}$$
$$g(x) = x^{16} + x^{15} + x^2 + 1 \tag{CRC-16}$$
$$g(x) = x^{16} + x^{12} + x^5 + 1 \tag{CRC-CCITT}$$
$$\begin{aligned} g(x) = {} & x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} \\ & + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \end{aligned} \tag{CRC-32}$$

Figure 4.9: List of generator polynomials adopted from (Höst, n.d.)

A way to measure the effectiveness of a CRC generator polynomial is its Hamming distance. The chosen CRC-8 polynomial has a hamming distance of at least 4 for data word length below 128 bits. The Hamming distance in CRC polynomials describes the minimum amount of errors that have to occur in a data word whereas the CRC decoder is then not able to detect the error. "For example, if a CRC polynomial has HD=6 for a given network, that means there are no possible combinations of 1-, 2-, 3-, 4-, nor 5-bit errors [...] that can result in an undetected error, but there is at least one combination of 6 bits that, when corrupted as a set within a message, is undetectable by that CRC." (Koopman & Chakravarty, 2004).

Koopman and Chakravarty describe, in their paper "Cyclic redundancy code (CRC) polynomial selection for embedded networks", a folklore like situation when it came/comes to choosing the right generator polynomial for your CRC. They stated, that some networks would pick CRC codes that are far from optimal for the given data length. Figure 4.10 shows a comparison between three CRC polynomials, whereas my chosen one is close to 0x97 in performance. 0x97 is a hexadecimal number and directly corresponds to the polynomial if written in binary, 10010111, and then as the actual polynomial, $x^8 + x^5 + x^3 + x^2 + x + 1$.

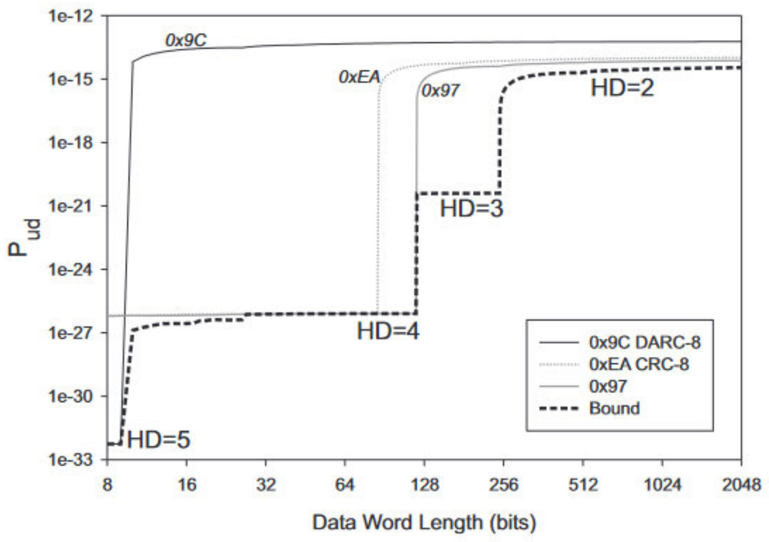


Figure 4.10: 8 bit, HD=4 CRC performance with an assumed BER of 10^{-6} adopted from (Koopman & Chakravarty, 2004)

The figure shows "[t]he probability of undetected error P_{ud} [which] is summed from the probability of successively higher numbers of bit errors at an assumed Bit Error Rate (BER) of 10^{-6} weighted by the percentage of errors caught per corresponding polynomial weights for each data word length. The bound line shown assumes a different, optimal, polynomial is selected for each length, and thus is a firm bound on performance. Lower numbers are better, indicating a lower probability of undetected error slipping past the CRC."(Koopman & Chakravarty, 2004).

Their results clearly show that word length plays a huge role in deciding which polynomial to go for. Since this thesis' scenario uses 64 bit word lengths for CRC encoding, the chosen CRC-8 code should work well for this application. Koopman's and Chakravarty's paper has a whole table with optimal polynomials based on data polynomial and word lengths to choose from.

4.2.4 Adding LDPC Code

In order to enhance the proposed system, an LDPC code is used as a third step after LT and CRC encoding. That way, the error correcting LDPC code can eliminate a lot of errors before the CRC code has to find the remaining. Two variants of LDPC codes have to be implemented, so we can compare the system with LDPC and CRC, or without LDPC but with CRC or only with LDPC. The first variant would be the LDPC encoder followed by the LT encoder resulting in 64 bits as the LDPC input. The second variant would include the CRC encoder in between the LT and LDPC encoder, resulting in the LDPC encoder seeing 72 bits at it's input.

Earlier I mentioned the proposed LDPC codes from specification CCSDS 231.1-O-1. They introduced three codes of varying sizes: 64x128, 128x256 and 256x512. Here the 64x128 code fits our requirements for a short length LDPC code perfectly and therefore will be used as well as a 72x144 code for the scenario of all three encoding methods being used at the same time. The 231.1-O-1 specification lists the parity check matrix and methods to calculate the generator matrix.

We can now implement our three codes (LT, CRC and LDPC) to study their respective effects on each other.

5 Implementation

Creating a simulation that made sense in respect to results from which we could derive meaningful information was really challenging. Throughout the process, I revised and changed the code multiple times. Complete overhauls, were knowledge gained up until that point was used to create a cleaner simulation, were not uncommon. The basis remained the same, a LT code simulation written in python by Francois Andrieux. He was also the author of a blog post explaining LT codes in detail (*Introduction to fountain codes: LT Codes with Python*, 2019). During my whole process, a CRC code and a fitting LDPC code were added into the existing framework, as well as an AWGN channel model, thereby creating my simulation. Understanding the difference between software development and simply coding was vital to this project. It is really easy to just code something without much sense behind it. Taking moments to reflect upon the written code and juxtapose it with the plan to create a reflection of the real world was the most difficult part of entire process.

5.1 Coding Environment

I used Trello in order to keep track of what had to be done and what needed to be worked on. Trello is an online project management tool that can be customized and therefore can fit any project. Cards can be created with captions and further detail if needed. I created four columns to keep track of my progress and organise myself better. First column denoted the "To-Do’s" while the second only held the cards with what I worked on at the very moment. The third one held all the finished cards, which was quite satisfying to see that column grow larger and larger. The fourth column acted similar to a trash bin in which all no longer necessary cards were put. Throughout the process, small projects had to be stopped or were changed into something else in order to fulfil the goal, prescribed earlier.

The code was written in a program called Visual Studio Code using the programming language python. Both were a commonly used during my bachelor courses and were chosen out of this reason. Working with an environment you are comfortable with is really important in order to create something new.
Everything was run on my 2019 Huawei Matebook 14, which had it's pro's and con's. Since this computer is rather weak in regards to computational power, smaller and smaller file lengths had to be simulated with each consecutive adding of encoding algorithm. In the end I simulated about 1000 bits, which should be totally sufficient. Using a consumer laptop had it's upsides when it came to researching the scenario of a sailor on a small vessel using a small computer, just like mine, in order to decode the incoming message. I had first hand reference on how long it may take someone to decode the message.

5.2 Code Implementation

Figure 5.1 depicts an overview of the written code. The simulation starts of by creating a random bit sequence of fixed length. Since the inner most code is the LT code which is block based, we need to divide the bit sequence into chosen sizes. Here we create blocks with a length of $k = 64$ bits. Meaning our input of 1000 bits gets broken up into $1000/64 = 15.625 \approx 16$ blocks sized 64 bits each. The simulation works with the thought in mind, that we encode and decode each block created by the LT process independently of one another. Earlier versions of my simulation coupled multiple blocks of 64 bits together in order to have larger input lengths for an LDPC encoder. Since I included a short-length LDPC code, this no longer needs to happen. It also better reflects a real life scenario in which each received, and fully encoded symbol gets processed one at a time.
Beforehand, in the prefix, the simulation can be set to different configurations. Depending on those settings, four if-statements conclude the next steps of the simulation. **(1)IF** we want to research the interaction between CRC and LDPC code we first encode each block with the CRC encoder resulting in $n1 = 72$ bits and then the result with the LDPC encoder $n2 = 144$ bits. **(2)IF** we only want the CRC encoder to encode our blocks, the second option does just that. **(3)IF**-statement number three gives us the equivalent of the second one, but instead of only CRC, we only encode the blocks with the LDPC encoder. **(4)IF**-statement four disables all outer codes and only encodes the blocks with the LT process. Regardless of chosen if-statement, the LT encoder follows them all resulting in a potentially $n3 =$ infinite amount of output symbols. Here we use the LT process to create so called symbols by XOR-ing different blocks with each other.

That concludes the simulation's encoding process, after which white Gaussian noise is being added onto every symbol we created. In order for our LT decoder to work properly, we sadly have to make a hard decision on each soft "bit" in order to get hard bits of 0 and 1. Sadly, because while reverting back to hard bits, we lose a lot of useful information about the channel the data has been transmitted, that could have been used by our decoding process. Later in chapter 5.2.4, we will further discuss the topic of soft and hard bits as inputs.

Since we can not simulate infinite packages, the R_{LT} variable limits the amount of symbols processed at the decoder. The decoding process start with the LT process searching for symbols of degree 1 in order to start the decoding process as a whole. If no symbols of degree 1 are found in the received symbols array, we stop the decoding. If, on the other hand, we find a symbol of degree 1, the simulation is presented with the corresponding if-statements from earlier in the encoding.

(1)IF the CRC and LDPC code are activated, we use the symbol and check which blocks have been XOR'd in order to produce the given symbol. Those other blocks are called the neighbours of the given symbol. Knowing that our given symbol of degree 1 directly corresponds to an original message block, we iterate over our other received symbols and XOR our given symbol/block with symbols that have our given symbol/block as their neighbour. In the process creating lower degree symbols than before, thereby new symbols with degree 1. This process is called "reducing the neighbours" and is done for all four configurations in the simulation. Still being in configuration (1) we then decode the input bits with the LDPC decoder in order to receive a set of message bits with added CRC check-bits at the end. Using those check bits, we confirm if our message data has been transmitted correctly or if there has/have been recognisable flip bits during transmission. If the CRC decoder finds no errors, the data gets successfully recovered and our symbol subsequently deleted from our pool of to-be-decoded symbols. If it detects and error, the symbol gets removed from the pool without recovering the data. The iterative search for yet another symbol with degree 1 starts all over again.

(2)IF we have only been encoding with a CRC code before, we now still have to reduce the neighbours in order to produce more symbols of degree 1. After that we check for errors using the 8 check-bits added onto our original 64 bits. During the same process as in configuration (1), we either find errors or not, ergo recovering our data before removing the symbol or immediately removing the symbol from the pool. The search for degree one continuous.

(3)IF we only encode with a LDPC code we reduce the neighbours and afterwards decode the LDPC encoded message. We directly recover the data and remove the symbol from the pool.

(4)IF neither a CRC or LDPC code has been used during the encoding process, we only reduce the neighbours before recovering the data, removing the symbol and starting the search for another symbol with degree 1.

The iterative search for symbols with degree 1 stops, as described earlier, if no symbol of degree 1 can be found, or if all data/message blocks have been recovered. After the second option we calculate the bit error ratio for our received bits and comparison to the sent ones.

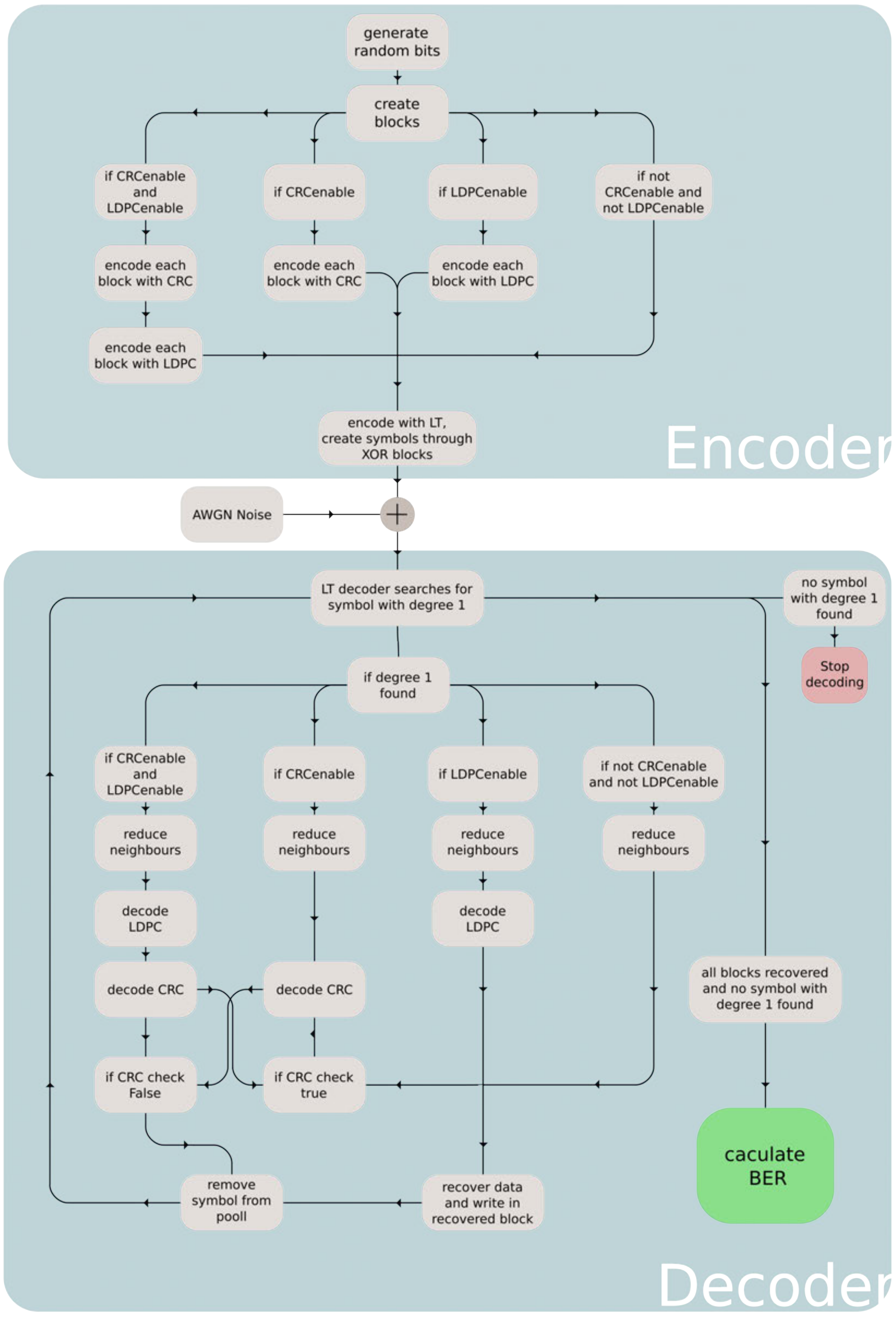
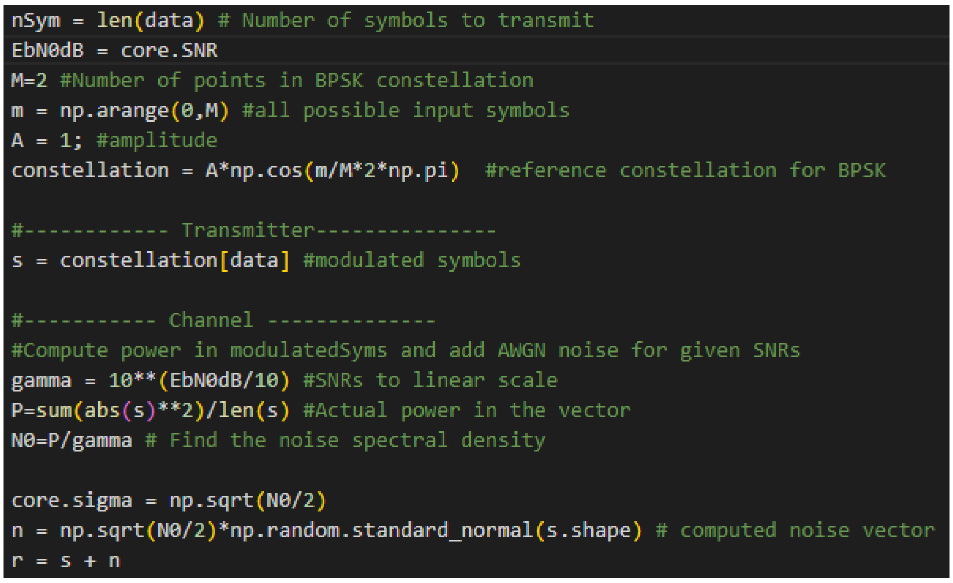

generate random bits
create blocks
if CRCenable and LDPCenable
if CRCenable
if LDPCenable
if not CRCenable and not LDPCenable
encode each block with CRC
encode each block with CRC
encode each block with LDPC
encode each block with LDPC
encode with LT, create symbols through XOR blocks
Encoder
AWGN Noise
LT decoder searches for symbol with degree 1
no symbol with degree 1 found
Stop decoding
if degree 1 found
if CRCenable and LDPCenable
if CRCenable
if LDPCenable
if not CRCenable and not LDPCenable
reduce neighbours
reduce neighbours
reduce neighbours
reduce neighbours
decode LDPC
decode LDPC
decode CRC
decode CRC
all blocks recovered and no symbol with degree 1 found
if CRC check False
if CRC check true
caculate BER
remove symbol from pooll
recover data and write in recovered block
Decoder

Figure 5.1: Blockschaltbild des Codes

5.2.1 Noise Implementation

The noise generator works by projecting the data onto a transmission signal, in this case BPSK (1, -1), and then adding noise of predetermined strength. The predetermined strength is given by the SNR we want to simulate. We first convert the SNR back to the linear scale from logarithmic after which we calculate the power spectral density in our data vector. By rearranging the SNR function $\gamma = \frac{P}{N_0}$ to $N_0 = \frac{P}{\gamma}$, we can calculate N_0 and therefore our noise vector which we then add to our data vector. I would have liked to implement a transmitter using OOK (NRZ) but I did not manage to do that. As mentioned in section 4.1.1 about modulation schemes, a light can only turn off and on. It is impossible to transmit a -1 since we can not have a light source which is more than off.

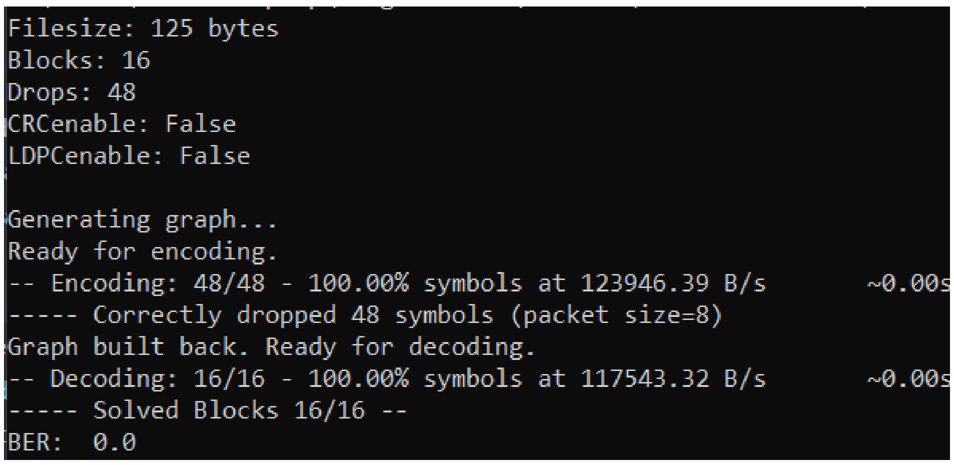


```
nSym = len(data) # Number of symbols to transmit
EbN0dB = core.SNR
M=2 #Number of points in BPSK constellation
m = np.arange(0,M) #all possible input symbols
A = 1; #amplitude
constellation = A*np.cos(m/M*2*np.pi)  #reference constellation for BPSK

#------------ Transmitter---------------
s = constellation[data] #modulated symbols

#----------- Channel --------------
#Compute power in modulatedSyms and add AWGN noise for given SNRs
gamma = 10**(EbN0dB/10) #SNRs to linear scale
P=sum(abs(s)**2)/len(s) #Actual power in the vector
N0=P/gamma # Find the noise spectral density

core.sigma = np.sqrt(N0/2)
n = np.sqrt(N0/2)*np.random.standard_normal(s.shape) # computed noise vector
r = s + n
```

Figure 5.2: Example of noise generating code

5.2.2 LT Code

Andrieux originally created a LT simulation, that could be interacted with through the console of windows (Andrieux, 2022). Since I felt that was a convenient way of interacting with a program, I continued with that approach and build on top of that. The original program wanted a file as an input to start the LT encoding and decoding process. Now we can chose the size as well as generate a file with random bits. Entering a string similar to this one: *python lt_codes.py dummy_file* $-s$ 125 $-r$ 3, starts the simulation. $-s$ represents the file size, while $-r$ responds to the LT-Code-Redundancy R_{LT}, or the amount of symbols we collect at the receiver. Figure 5.3 shows a randomly generated file of roughly 1000 bits, 125 bytes, being generated and the receiving end of the simulator collecting three times as many bits as the file size before decoding. The simulation doesn't simulate a constantly generating fountain code (LT), but instead we simulate the point of view of the decoder by already selecting a chunk of transmitted data to run our algorithms on. We could also run the program with a LT-Code-Redundancy ($-r$) of 100 and pick out 3% of those symbols to start the decoding. Since that would have the same result, we can just run our simulation for the amount of data the receiver will collect, to safe on computing time on the simulations end.



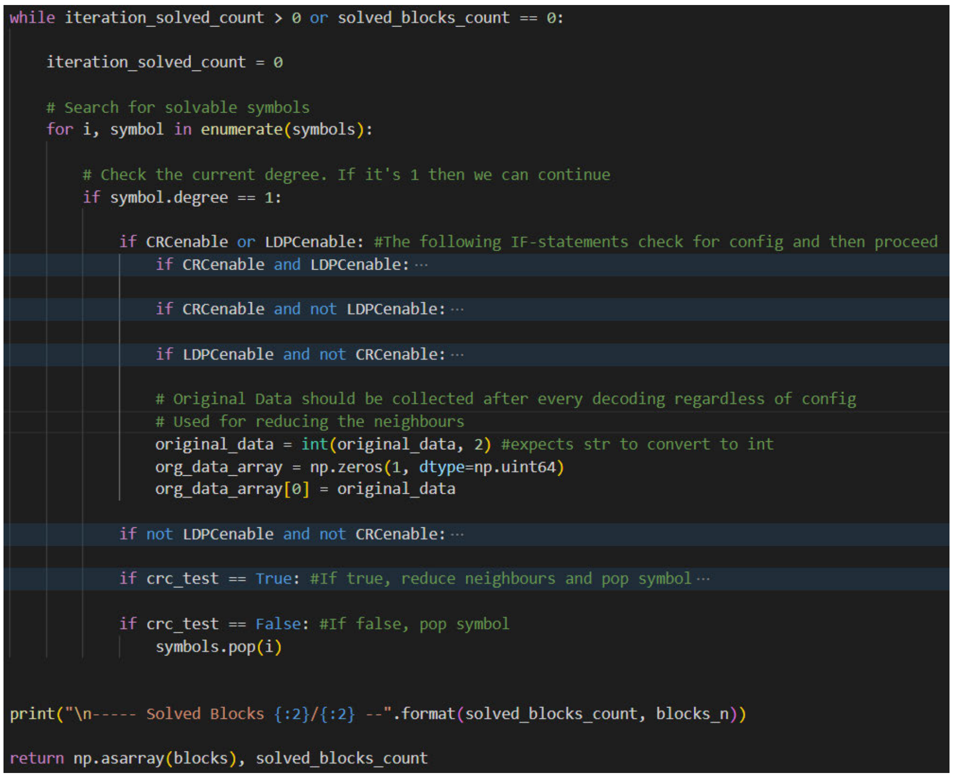
```
Filesize: 125 bytes
Blocks: 16
Drops: 48
CRCenable: False
LDPCenable: False

Generating graph...
Ready for encoding.
-- Encoding: 48/48 - 100.00% symbols at 123946.39 B/s       ~0.00s
----- Correctly dropped 48 symbols (packet size=8)
Graph built back. Ready for decoding.
-- Decoding: 16/16 - 100.00% symbols at 117543.32 B/s       ~0.00s
----- Solved Blocks 16/16 --
BER:  0.0
```

Figure 5.3: Console output after an LT code without CRC and LDPC

Figure 5.3 also shows the generated blocks (1000 bits /64 bits $= 15,75$) as well as the amount XOR'd symbols, being 48, dependant on the redundancy ($16 * 3$). CRCenable and LDPCenable are pretty self explanatory, but have to be hard set in the actual code itself as they can not be set in the cmd line as of now. After that we see a few statistics on how the decoding process is going, or went, before finding a line depicting the bit error ratio (BER) calculated at the end. The SNR in dB has to be set, just like CRCenable and LDPCenable in the code itself.

5.2.3 LDPC and CRC Code

The LDPC decoder as well as the CRC decoder were implemented in the LT decoding part of the code. In figure 5.4, we can see the while loop that iterates over the received symbols and searches for a symbol with degree 1. We then decode, depending on configuration, the data and proceed as described earlier.



```python
while iteration_solved_count > 0 or solved_blocks_count == 0:

    iteration_solved_count = 0

    # Search for solvable symbols
    for i, symbol in enumerate(symbols):

        # Check the current degree. If it's 1 then we can continue
        if symbol.degree == 1:

            if CRCenable or LDPCenable: #The following IF-statements check for config and then proceed
                if CRCenable and LDPCenable: ···

                if CRCenable and not LDPCenable: ···

                if LDPCenable and not CRCenable: ···

                # Original Data should be collected after every decoding regardless of config
                # Used for reducing the neighbours
                original_data = int(original_data, 2) #expects str to convert to int
                org_data_array = np.zeros(1, dtype=np.uint64)
                org_data_array[0] = original_data

            if not LDPCenable and not CRCenable: ···

            if crc_test == True: #If true, reduce neighbours and pop symbol ···

            if crc_test == False: #If false, pop symbol
                symbols.pop(i)


print("\n----- Solved Blocks {:2}/{:2} --".format(solved_blocks_count, blocks_n))

return np.asarray(blocks), solved_blocks_count
```

Figure 5.4: LT decoder while loop

The implemented LDPC code was written by (DelfiSpace, 2022) with the specification 231.1-O-1 in mind. In the LDPC decoder code we can find a few variants on how we want to decode the message. Methods (algorithms) called "BitFlip", "SumProductAlgorithm" or "MinimumSumAlgorithm" can be as our LDPC decoder. Through trial and error I concluded on the Sum Product Algorithm, since it produced the best results.

The CRC code was written by (Sneath, 2022) and basically performs a polynomial division if given two polynomials in binary form. The function can also check for errors if given the received polynomial together with the received check bits. The output, being true or false, was then used in my simulation to either recover the received data or throw it away by throwing the symbol out of the symbols array.

5.2.4 Soft and hard input

The LDPC decoder is able to use soft bits in order to decode the original message. By an iterative process, the additional knowledge about the channels condition inherent in soft bits, can be used by the decoder in order to yield a better error correcting result. The LT code has to reduce the neighbours by XOR-ing the current symbol with symbols that have the current symbol as their neighbour. An XOR operation is not possible with soft inputs, therefore we can not use soft inputs in the simulation at all.
To solve this, I would either have to implement the LDPC code after creating the symbols with the LT code, in order to decode every symbol before starting the LT process of searching for degree 1, or I implement an LT code that can handle soft inputs. Such a code has been proposed by (Zhang, Huang, & Shen, 2008).

5.3 Results of Simulation

In the following I am going to present results from my simulation, comparing relative data, transfer speeds and bit error ratios for the different configurations. I am also going to look at the effect, an increase or decrease of received packages at the receiver has on transfer speed and bit error ratio. The base line in regards to BER will be a transfer of 125 bytes whereas I only encode with the LT process. Figure 5.5 shows the theoretical bit error ratio over signal to noise ratio for a transmission over a BIAWGN channel in blue, and the results of the simulation for only the LT encoder being active in red. As expected, the LT encoding comes with a slight decrease in performance of about 2 dB. Since the LT encoder does not do anything helpful with errors, but instead just spreads them to other packages through the process of eliminating the neighbours, we see this decrease in performance. We also see a sort of hugging to the theoretical curve, indicating a true simulation with a shift.

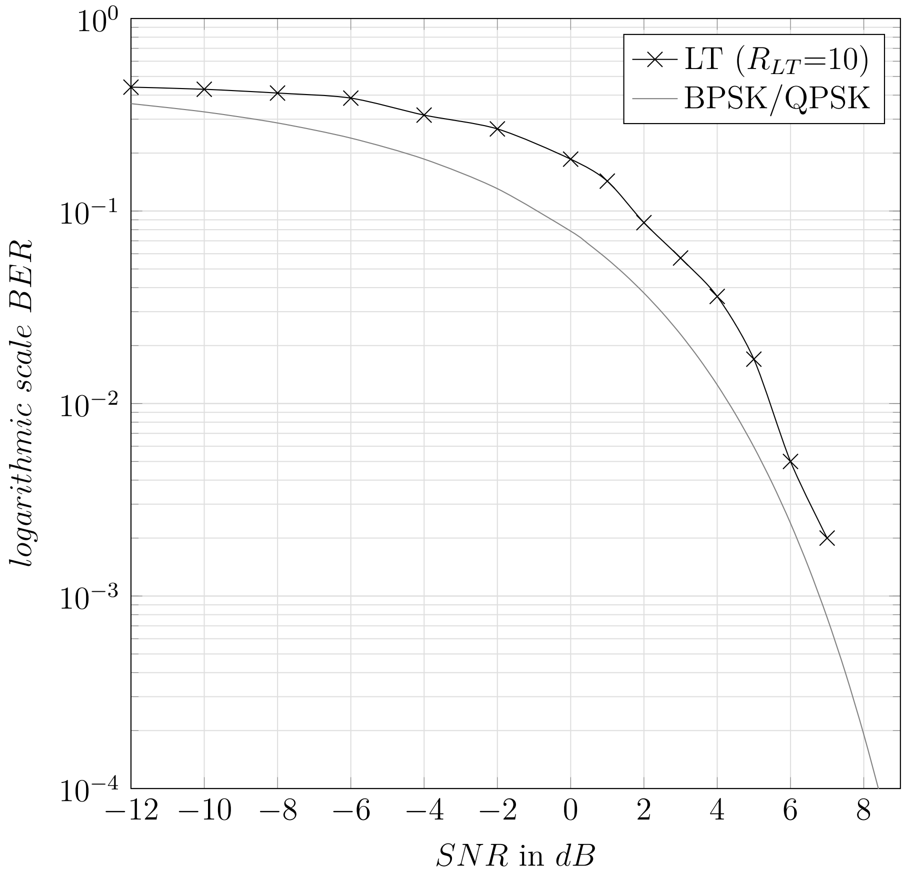


Figure 5.5: Bit error ratio over Signal to noise ratio for theoretical BPSK and LT encoded BPSK

The curve for LT encoding ends abruptly after 7 dB SNR, since there were no errors to be found at SNR values higher than 7. Out of visual clarity, the following graphs will handle no detected errors similarly. If we were to increase the package size, we would see errors even at higher SNR's akin to the theoretical curve. 64 bits on the other hand do not suffice for simulations with higher SNR's. If we then were to increase the package size and then widen the x-axis, we would see the same break at possibly 10 dB. Since it would just shift the same "problem" to the right, I decided to leave the simulation at a package size of 64 bits at LT encoding.

5.3.1 Bit error ratio

Let's remind ourselves of the three remaining possible configurations besides LT encoding: Only LDPC and LT encoding, only CRC and LT encoding, LDPC, CRC and LT encoding. The subsequent graph show good results at pre concluded redundancies. The specific effect of changing this variable will be discuss later on. The theoretical BER curve for a BPSK signal over an AWGN channel will also be displayed for a better visual comparison of the results.

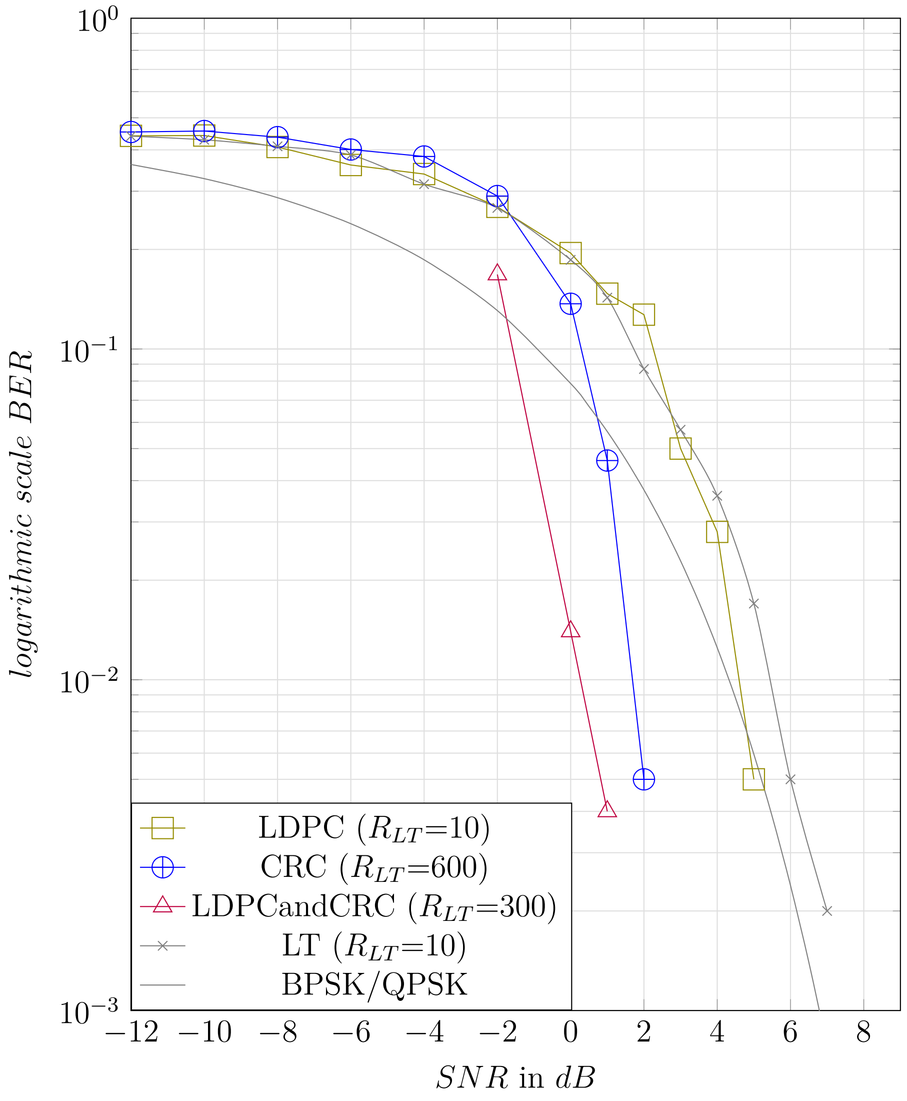


Figure 5.6: BER over SNR for LDPC + LT, CRC + LT, LDPC + CRC + LT and only LT code

Just like we did not see any errors above 7dB at LT encoding, we do not see any above 5dB if we add LDPC code into the encoder. One could say that we have generated a BEC channel at 5 dB or above by eliminating all errors allowing the LT encoder to work on error free data. The addition of an LDPC code shows benefits starting from 3 dB and gradually get better from there. This encoding scheme decoded the messages by only collecting an amount of packages equal to 10 times of the original message ($R_{LT} = 10$).
Even better results in respect to the bit error ratio can be achieved by only employing a CRC code and a LT code. Here we do not see any errors above 2dB. Again, just like with the addition of an LDPC code, we theoretically created a BEC channel in which faulty packages are not taking into consideration by the LT decoder since none reach it. The main goal of the cyclic redundancy check code was to throw away corrupted packages. So it makes total sense that we will not see any errors at higher SNR's, given a sufficient amount of packages to process ($R_{LT} = 600$). The remaining errors at low SNR values stem from the CRC code not being able to detect all errors, especially errors where multiple bits have been flipped. Since we employed a CRC code with a hamming distance of 4, meaning we can not detect some or all error combinations including 5 errors or higher. The check code then falsely flags error combinations stating it hasn't found any errors, therefore proceeding with recovering the data, resulting in a faulty decoded message. By increasing the CRC code bits, less packages would slip under the radar, and the process would eliminate more faulty packages.
I tested this by adding a stronger CRC code, this time a CRC-16, at the encoder. Graph 5.3.1 nicely shows the predicted effect. We achieve lower bit error ratios at lower signal to noise ratios. The better performing CRC code was now able to detect more error combinations which previously slipped under the radar by our CRC-8 code. At an SNR of -2 dB, the CRC-8 and CRC-16 code are both overwhelmed again, therefore flagging packages as false-positive. In order to get any packages with degree 1 at low SNR's, we had to collect a lot more packages at the receiver. Instead of $R_{LT} = 600$ times the original message, it has to collect $R_{LT} = 50000$ times the original message.

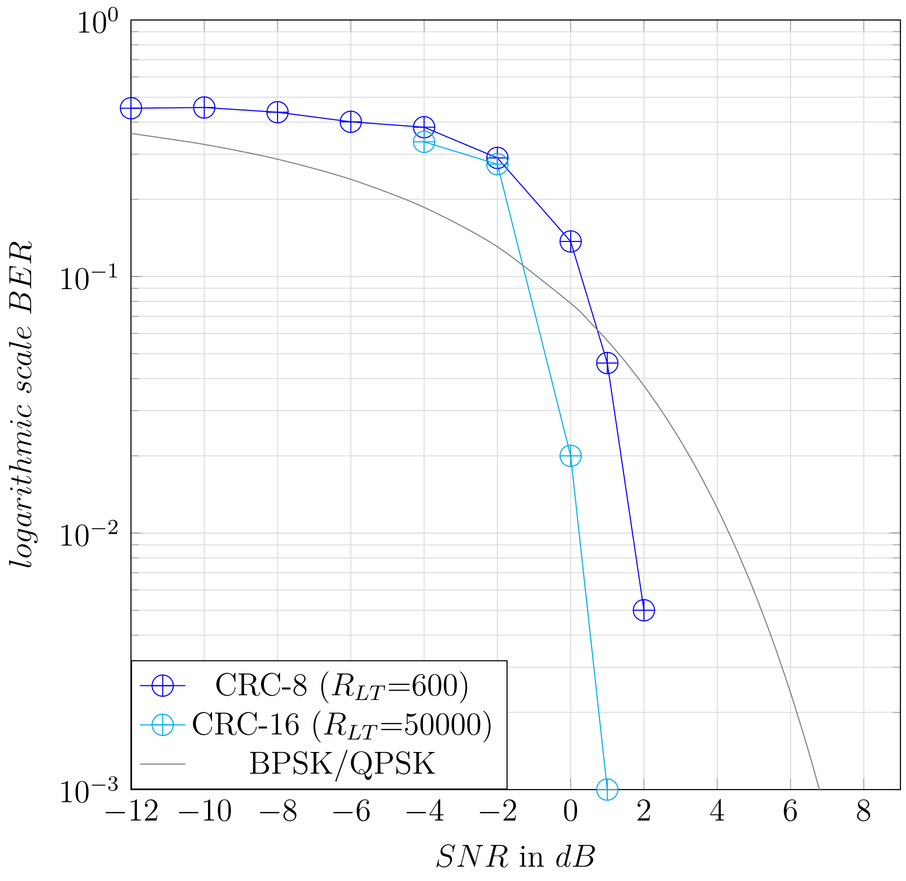


Figure 5.7: Stronger CRC code

Regardless of this effect, we can see a steadily decrease in errors after adding encoding algorithms. If we add the LDPC code and the CRC code before LT encoding, we see an even better performance than previously seen with only CRC encoding. This time we don't see any errors above the 1 dB mark. The amount of packages, in respect to the original message packages, needed drops as well in comparison to only CRC and LT encoding. Instead of $R_{LT} = 600$ we only need to process 400 times the packages of the original message packages in order to decode successfully.

5.3.2 Effects of LT-Code-Redundancy

To further study and evaluate our transmission scenario, we are going to change the LT-Code-Redundancy value and see how the different configurations may benefit, in respect to a better bit error ratio, or not. By changing this parameter, we can try to study the effect heavy sea states or thick fog can have on our ability to decode the message, depending on our chosen set up. Heavy sea states can make it harder to receive any packages at all, meaning in certain scenarios we need to be able to decode the message with the bare minimum of received packages.

Looking at the plain LT code as a basis again, we see that it does not react to a changing LT-Code-Redundancy value. Table 5.1 depicts the results of an average of 5 simulation runs for each field in the table. The coloured column corresponds to graph 5.6. The slight changes between different LT-Code-Redundancy values can be referred to the stochastic characteristic of our noise being added while simulating the transmission channel. Graph 5.3.2 shows even clearer that no significant trend can be derived from increasing or decreasing the LT-Code-Redundancy.

R_{LT}/ SNR	2	5	10
-12	0.452	0.441	0.440
-10	0.445	0.429	0.429
-8	0.406	0.430	0.410
-6	0.409	0.378	0.386
-4	0.363	0.337	0.315
-2	0.335	0.287	0.267
0	0.103	0.175	0.186
1	0.161	0.129	0.143
2	0.098	0.100	0.087
3	0.047	0.052	0.057
4	0.037	0.036	0.036
5	0.019	0.015	0.021
6	0.009	0.007	0.005
7	0.001	0.004	0.002
8	0.001	0	0
9	0	0	0

Table 5.1: LT-Code-Redundancy over signal to noise ratio and the corresponding bit error ratios for only LT encoding.

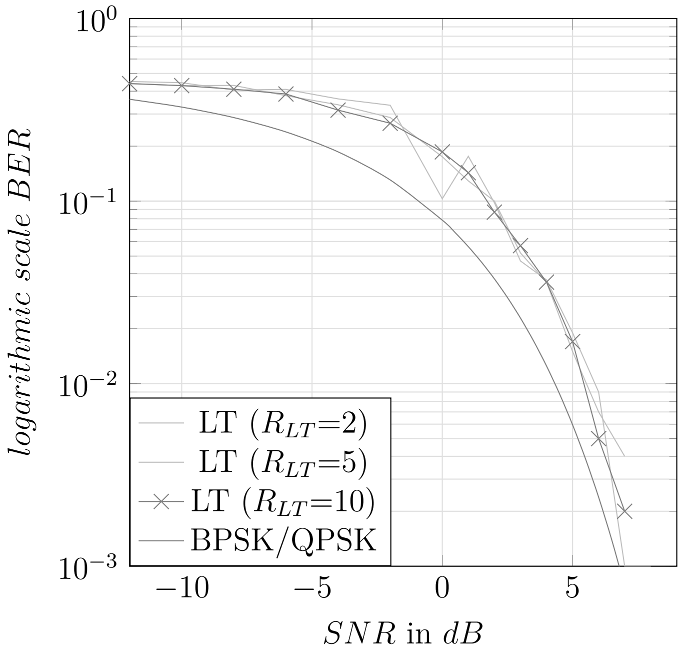


Figure 5.8: LT-Code-Redundancy over signal to noise ratio and the corresponding bit error ratios for only LT encoding

After adding the LDPC code as a precode to the LT code, we have seen in figure 5.3.1, that the LDPC code in symbiosis with the LT code yields better results in comparison to a lone LT code. A benefit from increased LT-Code-Redundancy, on the other hand, can not be detected. As seen in table 5.2 and graph 5.3.2, we see no significant change in BER across different SNR's if we modify the LT-Code-Redundancy. If we were to derive our results from average runs of 10 or more per field, we would see even less differences between varying LT-Code-Redundancy values.

R_{LT}/ SNR	2	5	10
-12	0.464	0.438	0.441
-10	0.456	0.430	0.442
-8	0.434	0.400	0.408
-6	0.425	0.375	0.360
-4	0.377	0.346	0.338
-2	0.313	0.284	0.269
0	0.272	0.187	0.195
1	0.164	0.145	0.147
2	0.127	0.098	0.127
3	0.050	0.056	0.050
4	0.022	0.018	0.028
5	0.008	0.004	0.005
6	0.015	0	0
7	0	0	0

Table 5.2: LT-Code-Redundancy over signal to noise ratio and the corresponding bit error ratios for LT and LDPC encoding.

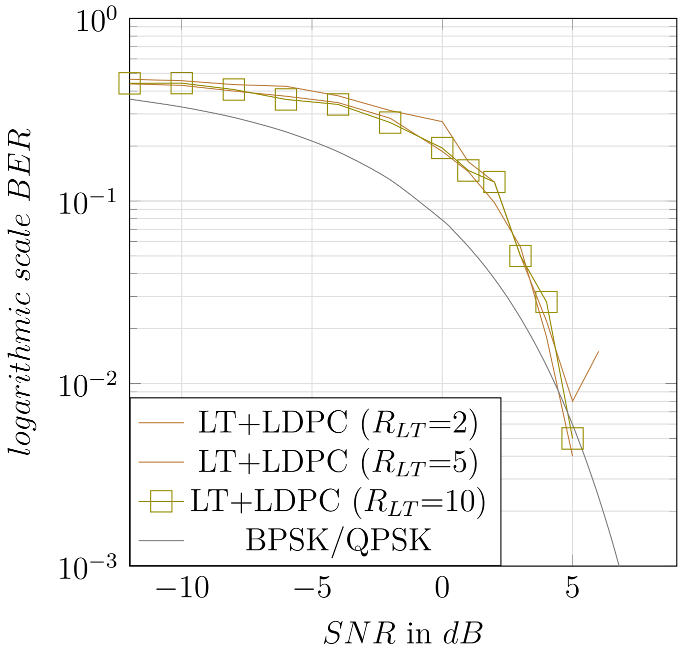


Figure 5.9: LT-Code-Redundancy over signal to noise ratio and the corresponding bit error ratios for only LT and LDPC encoding

By adding only a CRC code before the LT code and employing our "throw away if error detected" scheme, we can achieve significantly better results compared to LDPC and LT encoding. Since we throw away a lot of received packages, we have to collect a lot more of them at the receiver side, therefore increasing the LT-Code-Redundancy in our simulation. To better understand table 5.3, case1, 2 and 3 have to be explained. Case 1 refers to the decoder not being able to decode because it could not find any symbols with a degree of 1 after the CRC elimination process. Case 2 is similar in that a few symbols of degree 1 have been found, but not enough to finish the decoding process. Case 3, just like case 1 and case 2, refers to majority cases in the 5 simulations runs. Here the majority of sims achieved a BER of 0, but some were stuck in case 2 or 1. If no case number is given for the BER, all 5 sim runs for that value were able to decode the original message. Breaking it down, the higher the case number, the better the algorithm performed given the current LT-Code-Redundancy. Starting from a LT-Code-Redundancy of 100, enough packages were available to throw away the ones with detected errors and still have enough to decode the message properly. As mentioned in section 5.3.1, the errors at 1 dB at a LT-Code-Redundancy of 100 stem from the CRC code not being able to detect some errors and falsely flagging them as correctly transmitted. Raising the amount of packages to be processed to 600 yields results for all our chosen SNR values. Here we can observe the effect of wrongly flagging packages as error-free even better. Way to many packages slip through the CRC check. If we want to see better BER values at 1 dB or lower, we would have to implement a stronger CRC code.

R_{LT}/ SNR	2	5	10	50	100	600
-12	case1	case1	case1	case1	case1	0.453
-10	case1	case1	case1	case1	case1	0.456
-8	case1	case1	case1	case1	case1	0.437
-6	case1	case1	case1	case1	case1	0.401
-4	case1	case1	case1	case1	case2	0.382
-2	case1	case1	case1	case2	case2	0.290
0	case1	case1	case1	case2	case3	0.137
1	case1	case1	case1	case2	0.024	0.046
2	case1	case1	case1	0.012	0	0.005
3	case2	case2	case3	0.001	0	0
4	case2	case3	0	0	0	0
5	case2	0	0	0	0	0
6	case3	0	0	0	0	0
7	case3	0	0	0	0	0
8	case3	0	0	0	0	0
9	case3	0	0	0	0	0

Table 5.3: LT-Code-Redundancy over signal to noise ratio and the corresponding bit error ratios for LT and CRC encoding.

Table 5.4 shows the results for LT plus LDPC and CRC encoding. This configuration yields the best results, even at lower LT-Code-Redundancy values then CRC encoding. The few remaining errors that previously slipped through the CRC error detection process got eliminated by the LDPC error correction process before they could even reach the error detection process. Comparing the case of LT-Code-Redundancy 2 of LT+CRC and LT+LDPC+CRC encoding, we see all 5 simulation runs producing a BER of 0 at SNR value of 5 dB, while LT+CRC encoding does not produce BER's of 0 for any simulated SNR over all 5 runs. Meaning at an SNR of 5 dB, our receiver is now able to collect a lot less packages to successfully decode the original message. If we only use LT+CRC encoding, we have to gather 2.5 times (LT-Code-Redundancy of 5 instead of 2) as many packages in order to achieve a successful transmission.

R_{LT}/ SNR	2	5	10	50	300
-2	case1	case1	case1	case2	0.168
0	case1	case1	case1	case2	0.014
1	case2	case2	case2	0.003	0.004
2	case2	case2	case3	0	0
3	case2	case3	0.002	0	0
4	case2	0	0	0	0
5	0	0.009	0	0	0
6	0.009	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0

Table 5.4: LT-Code-Redundancy over signal to noise ratio and the corresponding bit error ratios for LT, LDPC and CRC encoding.

The LT + LDPC + CRC combination is a clear winner in regards to BER at low SNR values. It also nicely combines the good BER performance of only CRC encoding while keeping the amount of needed packages in respect to the original message to a minimum. All configurations have their up and downsides though. If we only consider the bit error rate in rating the configurations, we will get a distorted picture of a clear winner (LT+LDPC+CRC). For a more thorough comparison, we also have to look at the time it took the decoder to decode the message or, in other words, what kind of relative transmission speeds were achieved for the different configurations.

5.3.3 Data transfer speeds

The top performing configuration in respect to BER was the LT encoder plus LDPC and CRC. Table 5.5 shows the best results for each configuration based on determined LT-Code-Redundancy. But now instead of looking at the bit error ratio, we take the decoding time into consideration. The results are given in seconds. The top performing configuration (LT+LDPC+CRC) has a really high decoding time of 198 seconds at an SNR of 1 dB in comparison to the 0.69 seconds for LT plus CRC encoding. It is also worth mentioning that at really low SNR values of -4 or below, using the LT+LDPC+CRC encoding technique, my computer was not able to decode the message in a timely sufficient manner. Even transmitting the 125 bytes over a channel with an SNR of -2 dB took 70 minutes.

R_{LT}/	LT	LT+LDPC	LT+CRC	LT+LDPC+CRC
SNR	R_{LT}=10	R_{LT}=10	R_{LT}=600	R_{LT}=300
-12	0	31.91	1.44	unable
-10	0	28.17	2.64	unable
-8	0	29.97	1.99	unable
-6	0	26.64	2.03	unable
-4	0	24.78	1.99	unable
-2	0	29.11	1.74	4245.35
0	0	25.97	1.75	1045.99
1	0	19.98	0.69	198.57
2	0	16.77	0.30	73.21
3	0	9.13	0.19	46.99
4	0	5.31	0.14	26.03
5	0	2.57	0.11	6.22
6	0	1.42	0.09	1.21
7	0	1.12	0.12	1.34
8	0	1.23	0.12	1.58
9	0	1.19	0.10	1.24

Table 5.5: Comparison of decoding times for the best results for all configurations in seconds.

So if we were to only look at the decoding time, our clear winner would be the LT plus CRC configuration instead. Keeping in mind that the LT plus CRC combination has a much weaker performance in respect to the BER when compared to LT plus CRC plus LDPC encoding. The next chapter is going to reflect upon those results.

6 Evaluation

The results suggest different configurations work best for different scenarios. If our goal would be to transmit a low sized emergency message during bad weather conditions (low SNR), the results advocate for a LT+CRC plus LDPC encoding for added protection. If we were to transmit files of larger size, and the meteorological visibility on that day were good (high SNR) we could instead use the LT encoder plus only the CRC mechanism. Greatly reducing the decoding time and therefore being able to transmit lager files.

Using an error detection code in symbiosis with a Fountain code works best, regardless of file size, as long as the weather condition is good. Adding a more computational intensive error correcting code into the mix, results in a possible lower throughput, but gain in regards to the bit error ratio. So worse weather conditions can not solely rely on the special relationship between error detection and fountain code, but have to receive help by adding error correction on top.

6.1 How feasible is it?

Throughout writing this thesis I realized just how many variables I would have to include to give a truthful answer towards the feasibility of data transmission using light emitting navigational maritime structures. The list of unresearched topics regarding the scenario just grew the more I informed myself: Many lights blink at different rates or may be turned off entirely during the day. Many guiding lights emit light of varying strength, or color, thereby have fluctuating reach and bandwidth. How much light (input) does a photo detector need in order to receive a useful signal, and how is the a signal to noise ratio linked to changing degrees of light input. Just simply answering: "yes, you can transmit data through light-houses", would disregard the complexity of differing characteristics of guiding lights.

I also noticed, that writing a simulation requires a lot of time and knowledge, the latter of which I slowly gained more throughout the project. If I were to start all over again, I would now like to write everything on my own (resulting in a much longer development process) and not rely on pre-existing code to aid me. That way I would have more creative freedom over the way the simulation is set up.

Some ideas were hard to implement, mostly because of my lack of experience in coding. That meant that sometimes the most logical concept or idea did not find it's way into my simulation, simply because I did not know how to implement it. This feeling of being restricted by myself was the worst part about writing this thesis.

I can however say that using a Fountain code with error detecting capabilities, adding error correction if needed, is a good approach.

6.2 Future Research

An improvement upon the Fountain code used in this thesis, LT code, is the Raptor code. (Shokrollahi, 2006) Just like I used a precode in form of a CRC and LDPC code, the Raptor code inherently uses a different precode to improve performance.

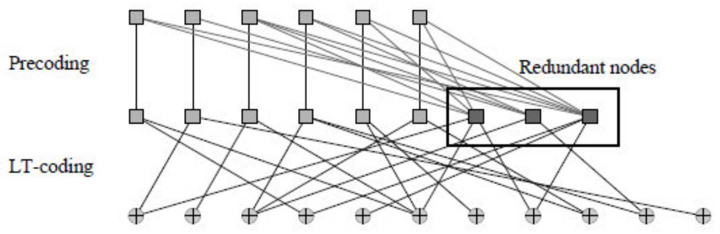


Figure 6.1: Schematic or Raptor Code adopted from (Shokrollahi, 2006)

Looking to improve the ability to decode soft symbols would be interesting as well. Speaking of the effects which produce certain signal to noise ratios: VLC based V2V communication (vehicle to vehicle) studies regarding weather conditions and reliability of data transmission could help refine and predict this connection in future research (Singh, Srivastava, & Bohara, 2019), (Ivascu, Ursutiu, & Samoila, 2020).

Another paper called "Multi-hop relay-based maritime visible light communication" (H.-J. Kim, Tiwari, & Chung, 2016) proposes a system of VLC using relay stations in order to achieve larger sea coverage. Through this clever approach the authors were not only able to increase the transmittable distance, but also improve the performance in respect to BER.

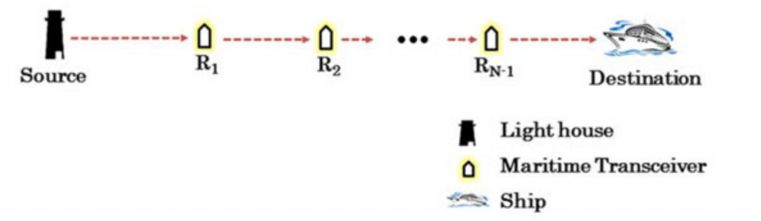


Figure 6.2: Proposed multi-hop system adopted from (H.-J. Kim et al., 2016)

I will look forward to reading papers about visible light communication in maritime settings in the future, since I think this area of research is going to produce some really interesting results.

List of Figures

List of Tables

Bibliography

Andrieux, F. (2022, August). *lt-codes-python.* Retrieved from `https://github.com/Spriteware/lt-codes-python` ([Online; accessed 1. Aug. 2022])

Beuth, K., Hanebuth, R., Kurz, G., & Lüders, C. (2001). Nachrichtentechnik-elektronik 7. *Vogel Buchverlag, Würzburg.*

Byers, J. W., Luby, M., Mitzenmacher, M., & Rege, A. (1998). A digital fountain approach to reliable distribution of bulk data. *ACM SIGCOMM Computer Communication Review*, *28*(4), 56–67.

CCSDS. (2015). Low density parity check code specification. *Short block length LDPC codes for TC synchronization and channel coding.*

DelfiSpace. (2022, July). *LDPC-Simulation.* Retrieved from `https://github.com/DelfiSpace/LDPC-Simulation` ([Online; accessed 13. Jul. 2022])

Filler, T. (2009). *Binary Additive White-Gaussian-Noise Channel.* Retrieved from `http://dde.binghamton.edu/filler/mct/lectures/25/mct-lect25-bawgnc.pdf` ([Online; accessed 17. Jul. 2022])

Gallager, R. G. (1963). Low-density parity check codes. cambridge. *Massachusttes.*

GMDSS Equipment carriage requirements for SOLAS ships. GMDSS Radio Survey Blog. (2022, July). Retrieved from `https://gmdsstesters.com/radio-survey/general/gmdss-equipment-carriage-requirements-for-solas-ships.html` ([Online; accessed 20. Jul. 2022])

HamburgPortAuthority. (2020, August). *New leading light line on the Elbe.* Youtube. Retrieved from `https://www.youtube.com/watch?v=3OeRF5gJsr4` ([Online; accessed 21. Jul. 2022])

Hoeher, P. A. (2019). *Visible light communications: theoretical and practical foundations.* Carl Hanser Verlag GmbH Co KG.

Höst, S. (n.d.). Error detection by crc.

Huang, Y., & Lei, J. (2013). Fountain codes design for asynchronous multi-relay system. In *Proceedings of 2013 3rd international conference on computer science and network technology* (pp. 752–756).

Introduction to fountain codes: Lt codes with python. (2019, June). ([Online; accessed 12. Jul. 2022])

Ivascu, C.-O., Ursutiu, D., & Samoila, C. (2020). Visible light communication for automotive market weather conditions simulation. In *International conference on remote engineering and virtual instrumentation* (pp. 637–651).

Jenkac, H., Mayer, T., Stockhammer, T., & Xu, W. (2005). Soft decoding of lt-codes for wireless broadcast. *Proc. IST Mobile Summit 2005*.

Kharel, A. (2013). Design of fountain codes with error control.

Kharel, A., & Cao, L. (2016). On the performance of lt codes over biawgn channel. In *2016 international conference on computer, information and telecommunication systems (cits)* (pp. 1–5).

Kim, H., Sewaiwar, A., & Chung, Y.-H. (2014, 09). Shore-to-sea maritime communication with visible light transmission..

Kim, H.-J., Tiwari, S. V., & Chung, Y.-H. (2016). Multi-hop relay-based maritime visible light communication. *Chinese Optics Letters*, *14*(5), 050607.

Koopman, P., & Chakravarty, T. (2004). Cyclic redundancy code (crc) polynomial selection for embedded networks. In *International conference on dependable systems and networks, 2004* (pp. 145–154).

„Leuchtturmprojekte“ an der Elbe. (2022, July). Retrieved from `https://www.thb.info/rubriken/detail/news/leuchtturmprojekte-an-der-elbe.html` ([Online; accessed 19. Jul. 2022])

Light: Electromagnetic waves, the electromagnetic spectrum and photons (article) | Khan Academy. (2022, July). Retrieved from `https://www.khanacademy.org/science/physics/light-waves/introduction-to-light-waves/a/light-and-the-electromagnetic-spectrum` ([Online; accessed 19. Jul. 2022])

Linear Feedback Shift Registers for the Uninitiated, Part XV: Error Detection and Correction - Jason Sachs. (2022, July). Retrieved from `https://www.embeddedrelated.com/showarticle/1180.php` ([Online; accessed 18. Jul. 2022])

Luby, M. (2002). Lt codes. In *The 43rd annual ieee symposium on foundations of computer science, 2002. proceedings.* (pp. 271–271).

MacKay, D. J., Mac Kay, D. J., et al. (2003). *Information theory, inference and learning algorithms.* Cambridge university press.

Mankar, M., Asutkar, G., & Dakhole, P. (2016). Reduced complexity quasicyclic ldpc encoder for ieee 802.11 n. *International Journal of VLSI design & Communication Systems (VLSICS)*, *7*(5/6).

Marine Communication. (2020, October). Retrieved from `https://shipip.com/marine-communication` ([Online; accessed 20. Jul. 2022])

Maritime safety: weather and navigation. (2016, October). *GOV*. Retrieved from `https://www.gov.uk/maritime-safety-weather-and-navigation/navigational-guidance-and-warnings`

Medova, L. R., Rybin, P. S., & Filatov, I. V. (2018). Short length ldpc code-candidate for satellite control channel. In *2018 engineering and telecommunication (ent-mipt)* (pp. 163–166).

Meyer, M. (1999). Kommunikationstechnik. *Vieweg-Herter/Lörcher: Nachrichtentechnik, Hanser*.

Mietzner, J., Harlakin, A., Hoeher, P. A., Fast, T., Liedtke, C., Heinze, R., & Greule, R. (n.d.). Joint light planning and error-rate prediction for dual-use lighting/visible light communication..

Mirrezaei, S. M., Faez, K., & Yousefi, S. (2014). Towards fountain codes. *Wireless Personal Communications*, *77*(2), 1533–1562.

of Transportation, U. D. (1986). *The Coast Guard Engineer's Digest.* United States Coast Guard. Retrieved from `https://books.google.de/books?id=JPz0hF5uHqkC&printsec=frontcover&hl=de#v=onepage&q&f=false`

Pub. 114, list of lights, radio aids and fog signals (Tech. Rep.). (2022). Springfield VA: National Geospatial-Intelligence Agency.

Rødseth, r., Tjora, s., Kvamstad, B., & Drezet, F. (2009, 12). D1.3 ship-shore communication requirements.. doi: 10.13140/RG.2.2.11488.20480

Sea Rules of the Road: The Display of Light & Audio Signals. (2009, May). Retrieved from `https://www.brighthubengineering.com/seafaring/35064-basics-of-sea-collision-regulations-lights-shapes-and-sound-signals` ([Online; accessed 19. Jul. 2022])

Shokrollahi, A. (2006). Raptor codes. *IEEE transactions on information theory*, *52*(6), 2551–2567.

Singh, G., Srivastava, A., & Bohara, V. A. (2019). Impact of weather conditions and interference on the performance of vlc based v2v communication. In *2019 21st international conference on transparent optical networks (icton)* (pp. 1–4).

Sneath, E. (2022, July). *Performs a cyclic redundancy check implemented in Python 3.3 with examples.* Retrieved from `https://gist.github.com/evansneath/4650991` ([Online; accessed 13. Jul. 2022])

Stockhammer, T., Shokrollahi, A., Watson, M., Luby, M., & Gasiba, T. (2008). Application layer forward error correction for mobile multimedia broadcasting. In *Handbook of mobile broadcasting* (pp. 239–278). Auerbach Publications.

Willemsen, D. (2022, July). *Lights, buoys – aids to navigation.* Retrieved from `https://www.sailingissues.com/navcourse9.html#nominal-range` ([Online; accessed 21. Jul. 2022])

WMO. (2018). *Guide to instruments and methods of observation.* World Meteorological Organization Geneva.

Zhang, K., Huang, X., & Shen, C. (2008). Soft decoder architecture of lt codes. In *2008 ieee workshop on signal processing systems* (pp. 210–215).

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangaben eindeutig kenntlich gemacht.

Ort, Datum	Unterschrift