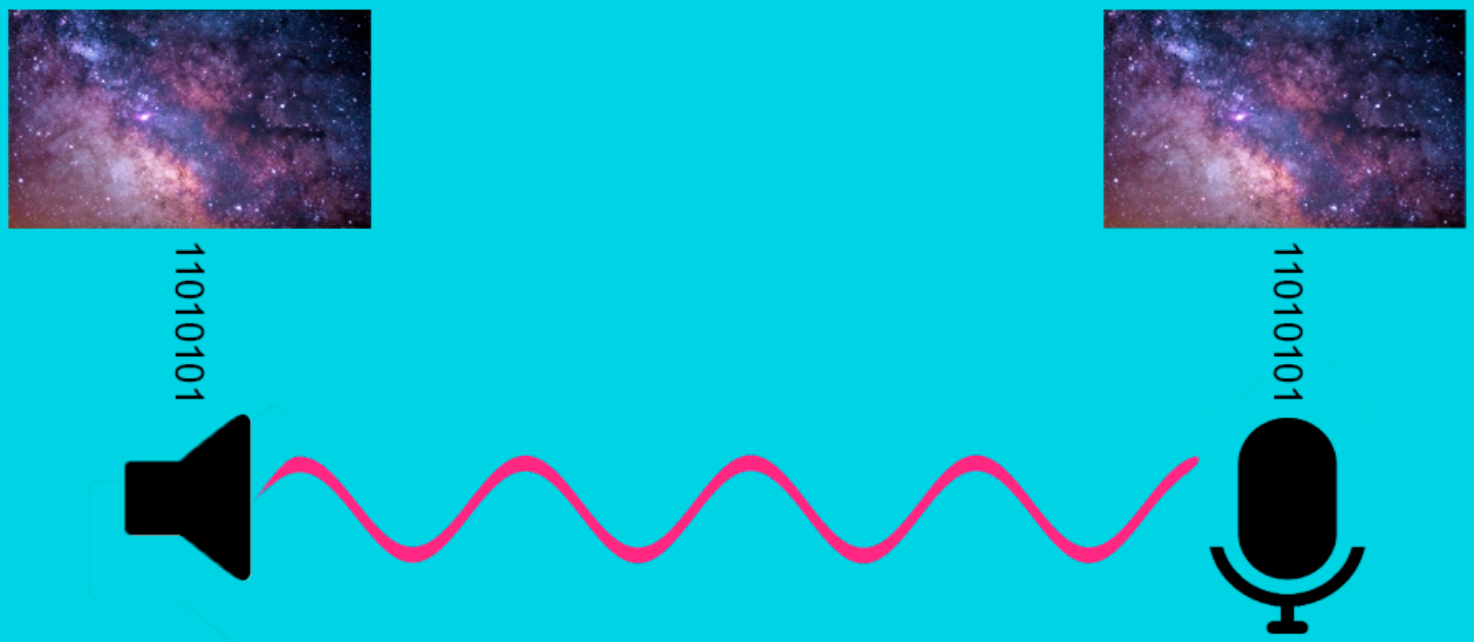


Development of a Coding Scheme for Acoustic Transmission of Images

A Study on Underwater Wireless Communication

Andreas H. Paludan, Casper Jensen, Emilie S. Steinmann, Ibrahim A. Abu Rached, Julie H. Bengtsson, Nicolai K. Hansen and Søren S. Christiansen

Software, B2-13, 2026-05

Semester Project

Development of a Coding Scheme for Acoustic Transmission of Images

A Study on Underwater Wireless Communication

Andreas H. Paludan, Casper Jensen, Emilie S. Steinmann, Ibrahim A. Abu Rached, Julie H. Bengtsson, Nicolai K. Hansen and Søren S. Christiansen

Software, B2-13, 2021-05

Semester Project

Here you can write something about which tools and software you have used for typesetting the document, running simulations and creating figures. If you do not know what to write, either leave this page blank or have a look at the colophon in some of your books.

Software
Aalborg University
http://www.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Acoustic Transmission of Images

Theme:
Algorithms in Digital Communication

Project Period:
Spring Semester 2021

Project Group:
B2-13

Participant(s):
Andreas H. Paludan
Casper Jensen
Emilie S. Steinmann
Ibrahim A. Abu Rached
Julie H. Bengtsson
Nicolai K. Hansen
Søren S. Christiansen

Supervisor(s):
Wei Fan

Copies: 1

Page Numbers: 75

Date of Completion:
February 18, 2026

Abstract:

The purpose of this report was to develop an underwater transmission scheme based on acoustic signals, making it possible to perform an offline transmission of images of approx. 1,000 kb over 1,500 m within a reasonable amount of time. This should have been done so that an operator could use the images for navigation.
To achieve this, a modulation scheme was established and a transmitter as well as a receiver was developed. The transmitter captures an image, converts the image into a binary string and encodes the binary string to audio using the developed modulation scheme. The receiving unit gathers the audio and decodes the audio back to a binary string, followed by a conversion back to the image. The image will then be presented on a simple HTML page.
It can be concluded that it was not possible to transmit images of the desired size, nor with the desired speed, though a modulation scheme was successfully implemented, making it possible to transmit a single image via an acoustic channel.

Contents

Chapter 1

Motivation

This project will be focusing on video/image compression for use in an underwater communication system. Traditionally, transmission of larger amounts of data has happened via a tether, due to the limitations of underwater communication. This project, however, seeks to challenge these limits. In the present section, an introduction to submersible vehicles will be given, with emphasis on unmanned underwater vehicles, as these come with certain inabilities that support the need for a wireless solution.

1.1 Submersible Vehicles

1.1.1 The First Submersible Vehicle

The concept and development of submersible vehicles dates back to 1775, where the first American submarine, "Turtle", was built. It was a primitive wooden and metal construction, operated by one person, with the capability to dive, control buoyancy and stay underwater for 30 minutes while air supply lasted. It was a military vehicle, first used in 1776 during a naval battle in New York Harbour. This became the first engagement in history to involve a submarine [69].

Since then, many more submersible vehicles have been conceptualized and constructed for various purposes. Along with submarines, came the development of torpedoes which under the current definition of autonomous underwater vehicles (see section 1.1.2), stand as some of the first [11].

1.1.2 Unmanned Underwater Vehicles

There are many different types of unmanned underwater vehicles (UUVs). Overall, they can be split into two groups: autonomous underwater vehicles (AUVs) and remotely operated vehicles (ROVs). As the terms indicate, the AUVs are preprogrammed vehicles that operate independently once deployed, whereas the ROVs throughout the entirety of their mission are controlled by an operator located above the surface.

In general, both AUVs and ROVs are equipped with various instruments and sensors, that are designed to carry out tasks often related to data-gathering and research. The vehicles are constructed and designed to fulfill tasks that require a range of operation that exceeds that of a diver or manned vehicle, e.g., tasks that require an extended period of time submerged, or in treacherous conditions, where a manned vehicle would be at risk [63, 65]. Since AUVs are fully autonomous once preprogrammed, they require little to no human supervision or maintenance once submerged. These vehicles carry on-board electricity to power their specific range of instruments, sensors, propellers, and maneuvering systems. The power consumption of the AUV largely determines its range and capabilities [63]. ROVs, on the other hand, are usually connected to a vehicle on the surface by a tether (cable) that facilitates both power supply and high speed communication to the ROVs. It is, however, also possible to connect to an ROV wirelessly.

Compared to the AUVs, the benefits of the ROVs lie in their ability to stay underwater for as long as it is needed, along with a much higher data rate for transferring data, as well as not having to rely on a battery, since it is powered by the tether. The AUVs on the other hand, have the advantages of not requiring a human operator, and also they have a better freedom of movement, as they do not have a tether limiting their range. AUVs and ROVs can in most circumstances be equipped with the same tools and instruments [11].

1.2 The Use and Necessities of AUVs and ROVs

1.2.1 Military and Law Enforcement Use

In 2008 the US Navy started to replace its manned submarine rescue systems with unmanned ROVs which were capable of rescuing stranded submarine crew from a depth of up to 610 meters [51]. Law enforcement agencies also employ UUVs for examination of ship hulls, when looking for contraband as it can be hidden underwater. They used to deploy a diving team, but now it is usually easier to use a ROV to scan the hull for any illegal items [44].

Militaries are currently using AUVs for search and rescue missions and mine countermeasure operations. The mine countermeasure operation has three steps: finding the mine, classifying it and destroying it. The Centre for Maritime Research and Experimentation (CMRE) is working on an improvement from a post-Cold war approach using surface ships towards a quickly deployed, scalable, cost effective autonomous system that minimize the risk of injuries and death to divers [19]. At the moment, only the first step can be accomplished by an AUV, while the last two rely on divers who are risking their lives and sometimes on ROVs, which are expensive and therefore not considered an expendable device. CMRE has concluded that an AUV has the potential to be a faster, safer and cheaper alternative for the classification and disposal of a mine. CMRE is currently working on making AUVs capable of all of the three steps and not just the first [19]. In the future, the AUVs will be taking on a more critical role in intelligence, surveillance, and reconnaissance, and anti-submarine warfare [31].

1.2.2 Commercial Use

Besides military use, AUVs and ROVs are used in many industries relying on underwater infrastructure such as deepwater drilling. Deep sea diving is a dangerous job [47] and companies like Petrobas [68] are looking for solutions that do not risk the lives of employees and save money. Companies began using the ROVs for maintenance in the oil and gas industry in the 1980's [79]. Nowadays, ROVs and AUVs are used in every possible way from aquaculture to underwater exploration but the majority is still used in the oil and gas industry [55]. Critical infrastructure like pipelines needs to be maintained and inspected regularly. If a pipe breaks, it can have a huge environmental impact on marine life, leaving companies with a huge fine and the cost of the cleanup [3]. AUVs can also be used to inspect dams, bridges, wastewater tanks, ship hulls and much more. Some underwater vehicles are even equipped with a robot arm and can perform simple repairs without human intervention [30].

1.2.3 Scientific Use

ROVs and AUVs are necessary for exploring the seafloor because it generally is unavailable for humans. The inaccessibility of the seafloor is the reason why more than 80 percent of the ocean has never been mapped or explored by any humans. The human kind know more about the surface of the moon and mars than we do of our own seafloor [59], which is why scientists instead use high tech robots to make the information of the seafloor available. ROVs and AUVs are two kinds of robots used in the deep-sea research [20]. The AUV Bluefin-21 saw use in 2014 when Malaysia Airlines Flight 370 disappeared over the

South China Sea. It was used to look for debris on the seafloor using sonar, mapping up to 90 square kilometers per day [22]. ROVs are usually more suited for complex tasks such as retrieving samples from the seafloor. ROVs are seen in use by marine scientists, who are using the ROVs to collect data about the ecosystem in the arctic to better understand climate change such as rising sea temperatures. ROVs were also on the front line of searching for Amelia Earhart´s plane and exploring the sunken Titanic wreck [91].

1.3 Challenges to the Use of AUVs and ROVs

In order to conduct remote-controlled underwater operations, it is necessary for the operator, located above the surface, to receive certain information while the operation is still on-going, e.g., video feedback and/or other information about the surroundings and the status of the operation [18].

One of the currently implemented solutions for remote control of ROVs is, as mentioned above, to use a cable to transmit data between the ROV and the command and control unit. This, however, is not always an optimal solution, given that the cable attached to the ROV might limit its range of maneuvers [17], e.g., by limiting how far or deep the ROV can go depending on the cable length [5]. Furthermore, the drag on the cable remarkably effects the stability of the ROV, especially in areas with strong water currents [5]. It also adds an extra dimension to the control of the ROV, since it is necessary to take the cable's effect on the ROV's movements into account [35]. According to [29], cable inertia is the most dominant source of error to the positioning of underwater vehicles.

Regarding the use of AUVs, different models have been implemented to avoid the need for resurfacing while an operation is undergoing. A common version is to use docking stations, as in [52, 54]. The AUV will return to the docking station according to parameters in the preprogrammed schedule for re-charging and transmission of data [52]. The docking station can have a cabled connection to a station above the surface, meaning that collected data from the AUV can be transmitted via the docking station at a high data rate [52]. This method seems practically applicable, though it is assumed that the AUV is required to cut off its mission in order to return to a docking station, hence taking away time from the data collection which might increase the number of AUVs required for a certain task.

Altogether, this calls for a wireless data transmission solution that can be employed under water. This will be examined in the present report.

Chapter 2

Video Transmission Requirements

In order to get a better understanding of the requirements to the performance of the UUVs, two of the dominant user groups presented in section 1.2, namely the marine biologists and oil companies, will be further examined in this chapter with a focus on their specific needs, use-cases and issues with current solutions. This will allow for a set of more specific requirements that can be used in the development of a solution.

2.1 Marine Biologists

As mentioned in section 1.2.3, AUVs and ROVs are widely used for scientific purposes, this including data gathering in the field of marine biology. The marine biologists employ both AUVs and ROVs as they come with their own sets of benefits and limitations.

The AUVs, on one hand, are well suited for scans of large areas that they can run over with a speed of around 1-2 m/s [94]. In contrast to this, the ROVs used in marine biology are tethered and therefore only capable of investigating a few hundreds of square meters of the seabed at a time [95], as cable management, as mentioned in section 1.3, continues to be an issue. Though the AUVs do not have a tethered connection to an operator above the surface, it is still possible to send simple instructions or receive a limited amount of data through acoustic channels, as the AUVs are equipped with the necessary technology for receiving and transmitting acoustic data [64, 94]. This channel, however, does not directly allow for a live video feed etc., unlike what is possible with tethered vehicles.

The ROVs used in marine biology often carry multiple cameras for different purposes [95].

One of these cameras will usually be for navigation, as it allows the operator of the ROV to get an overview of the surroundings. Usually, the operator will be in a boat on the surface, and the ROV will be deployed from the surface vehicle, meaning that it will operate in a relatively short distance of the surface vehicle [95]. One of the ROVs used in marine biology is for example the Saab Seaeye Falcon [64]. According to its manufacturer, Saab Seaeye, this vehicle can go down to 300 m, and can travel with a speed of approximately 1.5 m/s. It comes with a tether with a length of 1,100 m [75], meaning that it can never be more than 1,100 m away from the vehicle, it has been submerged from. According to [95], however, this is not practically possible, so it is assumed that in reality the ROV is operated over shorter distances than that.

In conclusion, it seems that the AUVs are limited by only being able to transmit simple messages while they are submerged, whereas the ROVs are limited to small geographical areas due to issues with cable management. This means that a hybrid solution, where an AUV, that already has the acoustic modem mounted to it, could provide a live video feed, allowing scientists to take advantage of the best from both worlds. The live video feed should provide the operators above the surface with enough knowledge to enable them to navigate the UUV within a relatively short distance.

2.2 Oil Companies

The oil industry is another sector that employs UUVs for a variety of tasks. It mainly uses ROVs to get access to areas that divers cannot reach, or where it is too dangerous for the divers to operate. Furthermore, these ROVs can be used for repairs of different underwater equipment, which earlier was a dangerous task for the divers. The ROVs have been very useful for the oil industry and they have different types of ROVs for different purposes. Mainly, the oil industry uses two types: work class ROVs and observational class ROVs. The work class ROVs can be used for different tasks as they have mechanical arms. The observational ROVs have the capability to record and transmit real-time video underwater and are used for both surveying and inspections. These can be found in different sizes, some smaller with a range of up to 300 m underwater and some bigger with a range of up to 6,096 m underwater [13]. The long tethers on the ROVs make the deep-water surveys less productive and it comes with costs depending on the depth of the water. This is why it can be more efficient to use AUVs instead [28].

A report from 2001 states that: "[an] AUV that can operate in water depths of up to 3,000 m should be adequate for most oil industry applications in the medium term" [28], but much has changed since then. Between 2001 and 2009 only 15% of the US oil production came

from ultra-deep[1] oil platforms, but in 2017 that number grew to 52% [16]. As oilfields at shallow water gets depleted, the offshore drilling companies will be searching for oil at even greater depths, which will require a longer range for communication systems [16].

It can therefore be concluded that because of ROVs being less productive due to their tethers, it would be more efficient to use AUVs for underwater video inspections. Considering live video feed is only intended to be used as a complement to other sensors, it would be reasonable to assume that a low resolution and frame rate is acceptable. The reach of the AUV should be at least 1,500 m.

2.3 Discussion on Video Transmission Requirements

When it comes to both the marine biologists and the oil companies it can be assumed that the requirements to the video feed would be that it could be transmitted over around 1,500 m, and that the quality of the picture should be good enough for the operator to identify objects on the seabed, e.g., corals, meaning that the video feed probably could be in black and white and still fulfill its purpose.

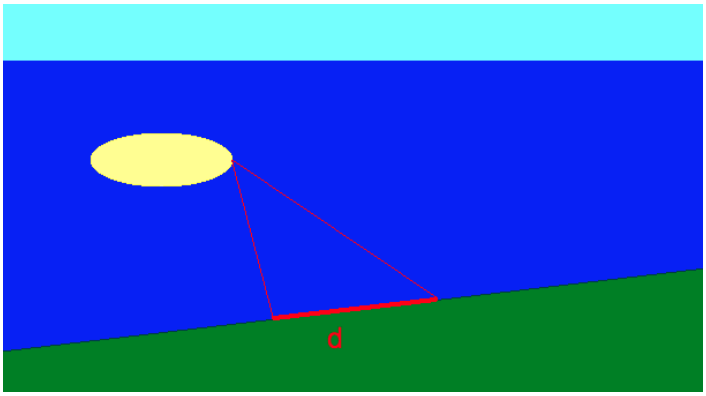


Figure 2.1: The illustration shows a yellow circle, representing an UUV moving from left to right with the velocity v. The two red lines between the vehicle and the green seabed show the scope of the camera, thus the thick red line, d, represents the distance on the seabed covered by the camera on the UUV.

Additionally, the frame rate of the video feed should be high enough to provide an overlap between the transmitted frames, meaning that the relation between the distance on the seabed ahead of the vehicle shown in one frame, d (as shown on fig. 2.1), the frame rate, FPS, and the velocity, v, of the vehicle – which in the case of the Falcon ROV is 1.5 m/s – must satisfy equation 2.1 in order to ensure that the operators will see the entire seabed, and that they at all times know, what is ahead of the vehicle.

[1] More than 1,500 m.

$$d \times FPS > v \tag{2.1}$$

Under the above mentioned assumptions, it seems like a low resolution camera (e.g., 360x480) in black and white with a color depth[2] of 5 or 6 bits would be sufficient in order to fulfill its purpose. If these numbers are multiplied together, they add up to a total of $360 * 480 * 5 = 864kb$ or $360 * 480 * 6 \approx 1,037kb$ per image, meaning that the solution should be able to handle this payload within a reasonable time.

[2]The number of bits required for each pixel [77], essentially determining the number of color nuances, or the number of shades of gray, when working on gray scale images

Chapter 3

Underwater Communication

In this section some of the key concepts used for assessing the usability of wireless signals will be introduced in order to facilitate the selection of the most suitable signal type – optic, acoustic or via radio waves – for this project. These three signal types will be evaluated in the end of the section.

3.1 General Signal Properties

When working with wireless signals, it is useful to be able to predict the capacity of a given signal type with a certain bandwidth for signal transmission between a transmitter and a receiving unit. In this section, the noisy-channel coding theorem, the Shannon-Hartley theorem, and the concept of attenuation will be explored as they provide a solid foundation for performing these predictions. Furthermore, certain aspects of the physical transmission of a signal will be discussed.

3.1.1 Channel Capacity

As mentioned, the noisy-channel coding theorem, the Shannon-Hartley theorem, and the concept of attenuation can be used together to understand the behaviour of a signal. In short, the noisy-channel coding theorem states that there exists an upper limit to the capacity of a communication channel. The Shannon-Hartley theorem provides us with a formula for this limit depending on several factors, including the signal strength at the re-

ceiver, which in turn can be evaluated with an adequate understanding of the attenuation of the signal through a given medium.

Going into greater detail, the noisy-channel coding theorem describes the maximum rate at which data can be transmitted relatively error-free through a communication channel. It states that given a noisy channel with a channel capacity C, then the data rate R is achievable if $R < C$. Conversely, if $R > C$ then R is unachievable [56, 81]. In order to obtain a data rate close to the channel capacity, it is necessary to implement highly efficient signal modulation schemes. This could for example be the 256 Quadrature Amplitude Modulation scheme that is currently one of the most efficient modulation schemes available [100]. The noisy-channel coding theorem does not provide any suggestion as to how big the maximum capacity is – but here the Shannon-Hartley theorem provides us with a solution. How to calculate the maximum capacity, C, measured in bits per second, is described in equation 3.1 [56].

$$C = B \cdot log_2(1 + \frac{S}{N}) \tag{3.1}$$

Here, B represents the bandwidth of the channel measured in hertz. The bigger the bandwidth, the larger amount of data can be transmitted and received at a signal over time. S is the power of the signal carrying the demanded data. This is measured at the receiver. N, on the other hand, is the power of the noise, also measured at the receiver. The noise can be described as additive white Gaussian noise[1] with a strength of N. The relation of S and N is S/N, also known as the signal to noise ratio (SNR). This means that SNR is a relation between two signals, where one of the signals carry the demanded data, and therefore is desired, whereas the other signal is an undesired signal, as it is nothing but disturbing noise [56]. From equation 3.1 it is clear that in order to achieve a reliable connection, it is favourable that the power of the desired signal, S, is greater than the power of the noise, N. In practice, the receiver sensitivity determines the smallest value of S at which the signal can be received successfully at a given noise level [90].

Especially in underwater environments, S is highly affected by the medium, the waves travels through, as the properties of water compared to air severely limits the propagation range of the transmitted signals. This is described by the phenomenon of attenuation, which means "lessening" or "weakening". When it comes to signals, attenuation is a fundamental property, which refers to a loss of signal strength over a distance [73]. This loss can e.g., be due to the friction between molecules or to the conductivity of the medium that the signal travels through, depending on the signal type and the medium [56].

[1] Additive white Gaussian noise is a model used to imitate the random noise from natural sources where all the frequencies have the same amplitude [23, 80].

3.1.2 Bit Flips and Errors

Bit flips or bit errors are a phenomenon and a common problem in all of signal transmission. It is the process of switching the values in binary code from a 1 to a 0 and vice versa. Bit flips can have many causes, from improper scientific design, miscalibrated sensors, and bugs in both software and hardware, to electrical interference. Bit errors are categorized into three types: single errors due to interference, mechanical failure, and malicious tampering, with the most common being interference in the shape of poor transmission and/or receiving conditions [38].

In order to combat bit flips and errors, one must resort to error detection and correction code, to ensure that a transmitted message remains identical from sender to receiver. Detecting errors can be done in various ways, but typically involves attaching "parity bits" which is a simple system of attaching certain bits to the intended message in a mathematical way, to give the receiver the ability to detect an error, although it doesn't offer a change to rectify it. Error correction codes however allow for the transmitted message to be recovered on the receiving end, should the code in transmission be corrupted or altered in any way [74]. Hamming code would be an example of such a correction code. If no error detection or bit correction code is in place, it is assumed that the receiving end has no way of validating the correctness of the code received, which can lead to faulty data, errors and other problems.

There are different error correction codes used for different purposes, one being the already mentioned Hamming code. The Hamming code is mostly used for single error correction meaning that it is used for correction of errors where only a single bit has changed by mistake. When using an error correction code, the number of bits, that need to be transmitted, will increase. The number of extra bits after using the Hamming code can be expressed by equation 3.2 [57], where the number of extra bits is described by r and the number of data bits are described by m. $m + r + 1$ describes the total number of bits after the message has been encoded using the Hamming code.

$$2^r \geq m + r + 1 \tag{3.2}$$

Then the r that will satisfy the equation should be found, and then the position of the extra bits are found by calculating 2^r for $r_n = 1, 2, 3, ..n$ where n is the r that satisfies the equation [92]. Overall, the increase of bits depends on the size of the data that is to be transmitted, and in general r will be relatively small compared to m, meaning that it will have a less significant influence on the total amount of bits.

3.2 Signal Performance in Underwater Environments

There are many ways of transferring data and communicating underwater. Several systems rely on wired connections that do not differ much from above surface communications, but it is difficult to establish the fast and reliable underwater wireless connections that this report is aiming to achieve. This section describes some of the different methods including their benefits and drawbacks.

Overall, there are two categories of waves, one being electromagnetic waves (EM), which include radio waves and light waves, and the other being mechanical waves, like acoustic waves. Electromagnetic waves do not require a medium to travel through, and thus they can travel through a vacuum. Light travelling through space is an example of this phenomenon. Mechanical waves, on the other hand, require a medium present in order to travel, and rely on the compression and rarefaction of the medium in which they operate, in order to propagate. An example of the absence of a medium for a mechanical wave, would be how there is no sound in the vacuum of space [6].

3.2.1 Radio Communication

The two main factors affecting the range of radio waves under water, are the frequency of the waves, and the conductivity of the saltwater it propagates through [7]. As saltwater has a relatively high conductivity, the attenuation will be quite high, resulting in a quick loss of signal strength (see fig. 3.1).

Figure 3.1 shows the relation of the frequency and the attenuation, in dB per metre. As the frequency is increasing, the attenuation increases as well. The attenuation also depends on the water type, and from figure 3.1 a conclusion can be that the attenuation is smaller in fresh water compared to sea water. One of the ways of calculating the absorption through a medium is with the absorption coefficient. It can be described by:

$$\alpha = \sqrt{\pi \cdot f \cdot \mu \cdot \sigma}, \tag{3.3}$$

where α is the absorption coefficient, f the frequency, μ the permeability and σ the conductivity [53]. Note that the absorption coefficient depends on the frequency, which means that while high frequency radio waves will have great attenuation and is only usable for very short range, low frequency waves will be better able to penetrate deeper into the ocean. Consequently, radio waves, except very low frequency waves (3 kHz to 30 kHz) and extremely low frequency (ELF) waves (3 Hz to 300 Hz), are unusable in seawater, and to transmit radio waves with such low frequencies, extremely long antennas might be

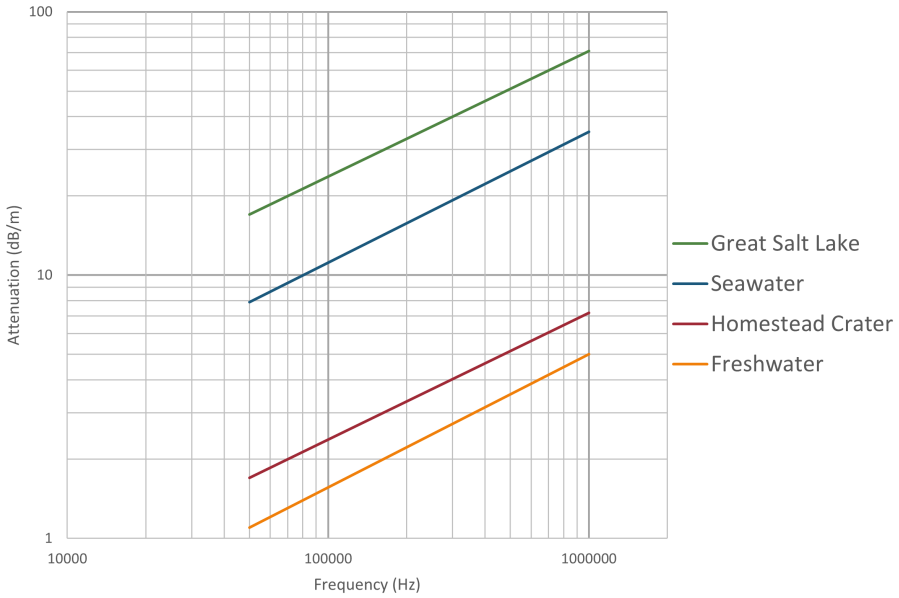


Figure 3.1: The correlation between attenuation and transmission frequency [43].

needed [93], because the length of an antenna should generally be half the length of the generated wave to achieve a high efficiency [89]. ELF waves are well outside the normal radio spectrum and are used for communication exclusively by militaries. US Navy project ELF was an earth-to-submarine communication system using extremely low frequency radio waves. This system operated at 76 Hz and assuming it travels at the speed of light, it had a wavelength of:

$$\lambda = \frac{v}{f} = \frac{c}{76Hz} = 3944638m, \tag{3.4}$$

where λ is the wavelength, v is velocity and f is the frequency. This means that the optimal length of the antenna would be:

$$\frac{(3944638m)}{2} = 1972319m \approx 1,972km \tag{3.5}$$

So, to achieve maximum efficiency the antenna should be 1,972 km long. That is why Project ELF used the bedrock of the earth as part of the antenna along with 28 miles[2] of antenna lines [36]. This does not add up to 1,972 km, but to counteract the inefficiency, the Project ELF facility consumed about 3 MW when transmitting. This resulted in a reliable but slow transmission with a transmitting power output of about 8 W. ELF wave attenuation is exceptionally low and can penetrate seawater up to hundreds of meters, but it comes at a cost of data rate and sizable transmitting antennas, making it impossible to mount a transmitter to an AUV, effectively making it one-way communication.

[2]Approx. 45 km

Altogether, it seems that as high frequency radio waves would be subject to a very high attenuation rate, the received signal would have a very low strength, thus, according to the Shannon-Hartley theorem presented in section 3.1, the capacity of the channel would be close to 0 for high frequency radio waves on any distances greater than a few tenths of meters [17]. Considering low frequency waves, Shannon-Hartley also explains that low frequency radio waves only provide a low channel capacity due to broadband limitations, and, as described in this section, it would not be possible to design an AUV that could meet the requirements to transmission equipment and power supply for a low frequency radio transmitter. This leads to the conclusion that radio waves are only well suited for underwater transmission over a few tenths of meters.

3.2.2 Optical Communications

Green and blue lasers and LEDs can be used to transfer information. They can be used on distances of up to 200 m [17] and work by very quickly turning on and off light which can then be converted to a digital signal by the receiver. Compared to radio waves, optical communication offers superior data rates (in best case up to 2.7 Gbps at 34 meter and 200 Mbps at 200 meters [17]), but it comes at the expense of only working when there is a line of sight between the transmitter and receiver. For short range, a LED light may be used because it is cheap and lightweight, but at longer distances a laser might be preferred because it can carry data further, but it requires a more powerful power supply and expensive setup. LEDs have the advantage of diffusing the light and is able to transmit to multiple receivers at the same time. Lasers on the other hand, have a very narrow beam of light and must be aimed accurately at the receiver which can be challenging at long distances if the water is turbulent, and the targets are moving. Both methods are subject to scattering and attenuation so to counteract this, most modems use either a blue or green light source since attenuation is lower at those wavelengths [97].

As light consists of electromagnetic waves, attenuation happens when the photons, which are both waves and particles simultaneously, get scattered or absorbed by particles and molecules. This results in light being reflected, scattered or absorbed between the sender and the receiver, meaning that there is a loss of light between the two. This is what is known as attenuation of light [86].

This scattering and absorption of light in water, can be calculated using the Beer-Lambert Law, which is a relationship between the attenuation of light through a substance and the properties of the given substance [48]. Water is over 800 times denser than air at 20° C and 1 ATM[3] [87], which is one of the properties that cause the heavy absorption. In water,

[3]Standard atmospheric pressure at sea level.

different wavelengths of light get absorbed at different depths, due to a phenomenon called "selective absorption". This is caused by vibration and deformations of water molecules, that get excited at different wavelengths. Absorption is stronger at longer wavelengths, while the exact values are determined by the water's transparency (plankton, sediment, etc.). Red is for example one of the first colors to be absorbed, making red light useless as a signal, due to its high absorption rate in water [72]. In a report [2], Mazin Ali shows that the SNR of light travelling through water even over short distances of less than 100 m would be so small, that C quickly approaches zero, meaning that we do not have a usable signal at distances greater than this, though using lasers, as mentioned, can improve the signal performance. This explains why optical solutions are only effective on distances up to a few 100 meters under water.

3.2.3 Acoustic Waves

Acoustic waves are mechanical waves, that function by vibrating a medium, in this case water. Acoustic waves travel much slower than radio waves, and even though acoustic waves travel about 4.3 times faster underwater than through the air [12], they still do not come anywhere near the speed of radio waves. Therefore acoustic signals also have a significantly lower data rate [62]. Data rates range from 50 kbps at 1 km [46] to 145 bps at 30 km [32]. Another drawback for acoustic signals is that they do not have the ability to cross the ocean surface meaning underwater-to-surface communication is a big challenge.

The attenuation largely comes down to two factors: viscosity and thermal conduction. Sound waves work by moving a medium in a back-and-forth motion. In water, it is this repeated change of pressure, we perceive as sound [60]. The initial energy from the signal source is spent on pushing the water molecules, and the friction between the molecules as they contract and relax in waves, results in the loss of energy and is part of the reason why the signal strength is lost over time. The friction is proportional to the viscosity of the medium [84], and thus the higher the viscosity, the more energy is lost from the mechanical waves passing through it.

It is difficult to assess the level of sound attenuation in water, as the actual data on the subject is with varying quality [61]. At low frequencies, the distance and amount of water required to calculate the conversion of signal strength to thermal energy is too large. For frequencies above 100 kHz, the heat loss to viscous drag from the molecules moving against each other in a signal is measurable. There is an exponential decay present in acoustic waves underwater, although it varies from sea to sea, due to the variety in chemical composition. Roughly, the loss of energy equates to the loss of 1 dB/km for acoustic signals with a frequency of 10 kHz [56], meaning that the energy of the signal decreases

by 21% for each successive kilometer [61].

An attenuation of 1 dB/km is relatively small, thus allowing for long distance transmissions. This, of course, comes at the cost of data rate, as the bandwidth is limited at low frequencies. Nonetheless, it can provide a communication channel with a data rate of around 15 kbps at a distance of 1,500 m [17].

3.3 Evaluation of Signal Types

Bearing the discussion from section 2.3 in mind, a solution for underwater wireless transmission should be able to handle a payload of approx. 1,000 kb per image within a reasonable time frame, over a distance of around 1,500 m. Radio and optic waves both suffer from high attenuation, meaning that neither would be able to provide a solid connection over distances of that order. This leaves only one alternative: acoustic waves. The data rate for acoustic waves with a range of 1,500 m is around 15 kbps, meaning that in a naive model for data transmission, the transmission of one image would take around one minute.

Chapter 4

Current Solutions

There are no commercial models for wirelessly transmitting video and images under water, but several experiments have been carried out. The distance and the quality of the video depends on the chosen method. High quality video equals short transmitting distance, e.g., in cases where optical lasers are used. An experiment was made where a transmission of a UHD real-time video under water was achieved, but only over a distance of up to 4.5 m [71].

There has also been an experimental attempt on optic wireless water-to-air communication, between an underwater platform and an airborne terminal. In this attempt it was achieved to have wireless communication in the range of a 26 m air/water channel, 21 meters in water and 5 in air. This model implemented 32-QAM (Quadrature Amplitude Modulation) Orthogonal Frequency Division Multiplexing (OFDM) signals with 218 sub-carrier waves where the data rate was increased in the four attempts with four different data rates, 4.8, 5.3, 5.5 and 6 Gbps [21].

The company Sonardyne is the manufacturer behind the BlueComm product line. In the product line there is an optical modem, BlueComm 200, which can be attached on submersible vehicles and is capable of transmitting data wirelessly in a range of 150 m. The modem has a bit rate of 2-10 Mbps making it possible to transmit a live video feed. There are not many solutions when transmitting videos wirelessly underwater and this appears to be one of the longest distances available for video transmission currently in use [82, 83].

As far as the research for this project has shown, it is indeed possible to transmit video underwater, using some of the already existing modems available on the market. However,

as of the time of writing this paper, there are currently no AUVs widely available on the market, capable of maintaining a video feed, or even transmitting video offline. The solution to gathering data with a camera is still done by storing the data aboard the vessel and then retrieving it later.

Chapter 5

Problem Statement

Through the previous chapters, it has become clear that the interest for a wireless underwater data transmission system is present. It seems that a wireless solution for almost-spontaneous image transmission on distances of around 1,500 m would be useful for marine biologists and oil companies, amongst others. Unfortunately, it is not possible to implement the above-surface signal schemes directly in underwater environments. It appears that both radio waves and optic waves are subjects to attenuation to the degree where they cannot be implemented in an UUV and effectively send out signals, unless the receiving unit is located within a few hundred meters. Acoustic waves, on the other hand, enable long-distance transmission, though it comes at the cost of data rate. At 1,500 m, it seems that a data rate of 15 kbps would be possible to achieve with a good modem. This is, however, very low, when the aim is to transmit images, and roughly, the transmission of a very simple, gray scale image would take around one minute with a naive coding scheme. As a result of this, the solution will only be able to handle offline transmission of images, i.e. with a time delay. As there are currently no commercial solutions to this problem, the following problem statement has been put forward:

> *How can an underwater transmission scheme based on acoustic signals be designed, and is it possible to perform an offline transmission of images of approx. 1,000 kb over a distance of 1,500 m within a reasonable amount of time so that an operator can use them for navigation?*

In the following chapter, the different methods for improving the transmission scheme will be examined in order to obtain a solid solution for the present problem.

Chapter 6

Solution Analysis

Strictly speaking, the problem stated in chapter 5 is all about minimizing the ratio between the payload and data rate, hence allowing for a lower transmission time. Different strategies can be used to achieve this. In this section, three approaches will be analyzed, these being the use of multiple antennas, an adaptive system that varies with the distance, and image compression.

6.1 Multiple Transmitters and Receivers

It is possible to affect the channel capacity C, by using a technique called multiple input and multiple output (MIMO). There are two ways to use MIMO to improve the signal; beamforming and spatial multiplexing.

Beamforming is a technique to increase the robustness and range of a system by using multiple transmitters to transmit the same data [58]. By utilizing constructive interference and phase shifting, the signal will become significantly more powerful in one direction, increasing the SNR in this direction, but it will be weak in other directions [40]. More transmitters lead to a more directional signal with less side lobes in other directions [88], as shown on fig. 6.1. This works great when low power consumption is required since you can focus the signal on one specific target, so no energy is wasted by beaming signals in all directions [49].

The position and phase shift can be determined by measuring the time delay from each transmitter to the receiver and then adjusting the phase accordingly to aim the main beam

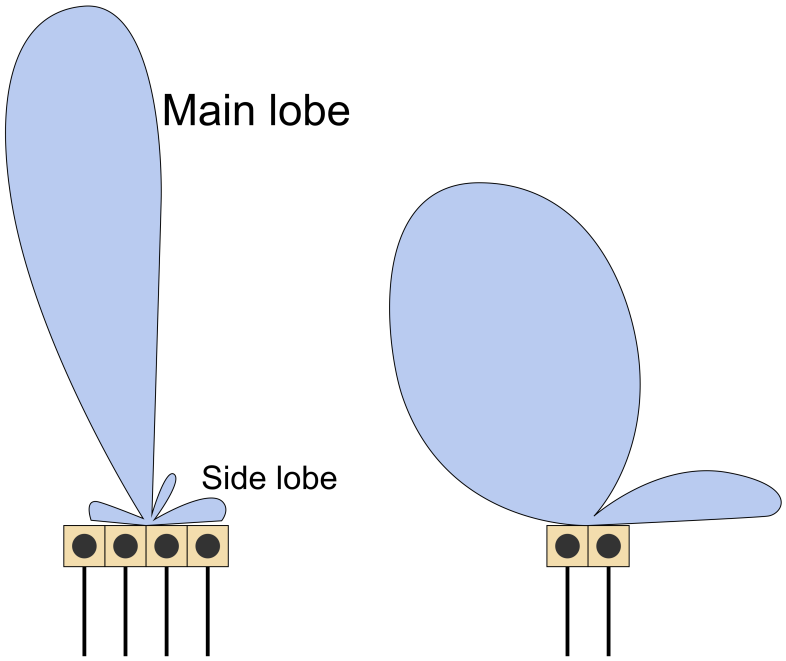


Figure 6.1: Beamforming: by utilizing constructive interference and multiple transmitters it is possible to generate a more directional beam [10, 88].

directly at the receiver. The time delay is difficult to estimate under water considering the speed of sound varies depending on temperature, depth and salinity. Furthermore, it can be difficult to estimate the direction and distance to the receiver because GPS is unusable under water.

Spatial multiplexing works by splitting the data stream up and using multiple transmitters and receivers. This way, the bit rate can improve by a factor of N, where N is the number of transmitters [39]. Then the total capacity can be expressed as:

$$C_N = N \cdot B \cdot log_2(1 + SNR) \tag{6.1}$$

In equation 6.1, N describes the number of pairs of transmitters and receivers, and it can be seen that C_N is equal to $N \cdot C$, where C is computed as in eq. 3.1, meaning that the channel capacity using multiple transmitters and receivers is proportional to the channel capacity, when only using one pair of transmitters and receivers. The bigger N, the larger the total channel capacity.

Both methods require more space than a single transmitter because the transmitter elements needs to be spaced with a distance greater than half the wavelength. As previously

mentioned, the wavelength is very long in the low frequency spectrum, where the only usable frequencies for underwater communications are found, meaning that both methods would require a quite broad transmitter station on the UUV.

6.2 Adaptive Data Rate Transmission

Another strategy could be to adapt the transmission frequency to the distance to the receiver. According to [17], it is possible to achieve higher data rates over shorter distances than over longer distances, so a system should be designed, that could take advantage of these different optimal data rates by using different frequencies for transmission, depending on the distance to the receiver and the receiver's sensitivity.

When data rates are high enough, it would of course be ideal to transmit multiple frames a second, but when data rates are limited, it could be possible to adjust the frame rate and the velocity of the vehicle, in accordance with eq. 2.1, to counter the low data rate that can be achieved over longer distances, as the data rate varies a lot depending on the transmission distance [17].

6.3 Image Compression

Data compression is a technique used to reduce the amount of data needed to represent an object by reducing the amount of redundancy [8], hence allowing for better use of the data rate through a given transmission channel. This is essential in an underwater acoustic communication system since the data rate is very low and compression greatly reduces the transmission time. A compression system typically consists of an encoder and a decoder. First, the encoding algorithm produces a shorter binary representation of the data that is to be transmitted. The short string is then transmitted to a receiving unit, where the decoding algorithm transforms the encoded data string back into a satisfactory representation of the original data [8]. See fig. 6.2 for a schematic overview of this process.

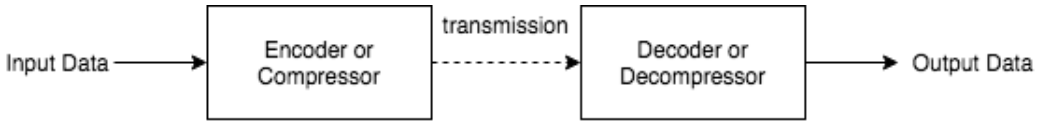


Figure 6.2: A simple model for the process of transmitting data through a communication channel by the use of compression and decompression algorithms [8].

Depending on the data input, a certain error rate can be accepted, thus allowing for the use

of lossy data compression that, in contrast to lossless compression, is not fully reversible [76]. Lossy and lossless compression types are also presented as a way to compare compression techniques based on the quality of the output data. Most of the time, it is not necessary to use lossless compression of pictures and video, though transmission of text should always be lossless, since errors in the transmission could result in missing or misplaced letters etc. When dealing with pictures though, minor variations in colors and shades might not be noticed, nor be of any importance for the full picture. Only when the picture or video is up for detailed analysis (e.g., in the case of medical scans) it might be necessary to use lossless compression [76, 77]. The benefits of using lossy compression, when possible, is that the compression effect will typically be greater than that of lossless compression [8]. The compression effect can be evaluated by computing the compression ratio that describes the ratio between the size of the compressed bit string and the size of the original data [77]. This means that any compression ratio strictly below 1 will indicate that the compression has had the desired effect on the original data, resulting in a shorter bit sequence for transmission. The smaller the compression ratio is, the bigger the impact of the compression code has been. Another useful parameter for evaluating the compression effect when working with pictures is the number of bits used to represent each pixel (bits per pixel, bpp). The average bpp of the compressed image should be compared to that of the original data [77].

In the case of this project, a lossy compression technique is most beneficial, as this will make a bigger impact on the amount of data that has to be transmitted, by reducing the compression ratio as much as possible. This is necessary given the low data rate that has to be overcome. Additionally, images can usually afford to lose a lot of data, while still maintaining a recognizable image where the most important details are preserved.

6.4 Summary

As described above, there are possible ways to improve underwater communications. The obvious solution would be to use more channels for data transmission than what is currently being used. The challenge that using multiple channels will present, is that taking up more bandwidth has a larger risk of overlapping with other acoustic wave communication systems. Further improvement can be made by reducing the amount of data being transmitted with compression, meaning that the limited data rate is utilized more efficiently. Based on this, it has been decided that the implementation will not be using multiple channels, but instead focusing on image compression.

Chapter 7

Implementation

In this section the concept of the solution and its requirements will be described. Furthermore, a brief description of the functionality of the transmitter and the receiver part of the solution will be presented.

7.1 Concept

In order to design a solution for the problem stated in chapter 5, an analysis has been made using the MoSCoW method. The MoSCoW method is a method for prioritizing the different requirements for a project, which can be helpful in order to point out which are the most important. The letters M, S, C, and W stand for: Must have, Should have, Could have and Will not have. These are the different groups that the requirements will be classified into [42]. The result of the analysis is shown on a flowchart, and finally a modulation scheme has been chosen for the audio transmission of the binary representation of the compressed image.

The requirements for this project are:

Must have The solution must be able to perform compression of images, a conversion of the compressed image to a bit string, and a conversion of the bit string to an audio signal followed by decoding.

Should have The solution should have a very simple user interface, where it is possible to record the sound as well as stopping the sound recording.

Could have The solution could have an error correction code to prevent bit flips. Also an evaluation of the compression effect could be present in the solution. The solution could also be adaptive in the form of a dynamic change/modification of the solution or that the bit rate should be dependent on distance or noise. This would mean that the transmitter has to receive a signal telling it how far it is from the mother ship.

Will not have The solution will not have the possibility of video transmission, nor the possibility of live transmission, and this is mainly due to transmission time. Also the solution will not be tested in an underwater environment.

Due to the time limit of the project, only the must have requirements and the should have requirements will be implemented in the solution.

7.1.1 Overall Idea for Implementation

The general idea of how these requirements will be implemented is presented in fig. 7.1. Here, it is shown, how an image will first be compressed and converted to an audio signal. At the receiver end, the registered input must be analyzed and the information extracted, before it can be converted back into an image.

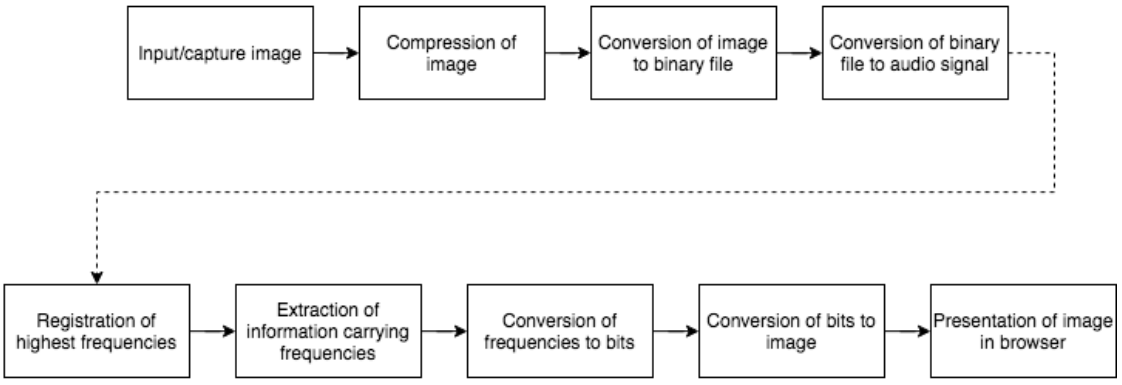


Figure 7.1: Schematic overview of the main flow in the solution. The dotted line represents the actual transmission of the audio signal. The top row is the transmitter part (Tx), and the bottom row is the receiving part (Rx).

The image transmission using acoustic waves will be done as an air transmission for proof-of-concept as we do not have the equipment to test under water. The transmission setup will be using a speaker to play the sound and using a microphone to record it. This means that the entire process can be simulated where the only difference is that the underwater channel is replaced by an air transmission.

The analysis of the microphone input, which will be required for registration of the highest frequencies, can be made using the P5 JavaScript library described in 8.2.2. As the update rate of P5 is bound to the framerate of the receiving unit, in this case a computer, this will be limiting the data rate, as the receiver can not detect the quick change of frequency. The audio file produced in the transmitting part (Tx) will have to take this into account when determining the bit duration[1].

Signal Modulation

Before the signal can be sent, it needs to be encoded with a modulation scheme. The most common modulation schemes are AM and FM.

AM stands for Amplitude Modulation, and it works by modulating the amplitude of a signal with a constant carrier frequency. If a bit string is modulated with AM, one way to modulate a bit string is where a high amplitude is equal to a 1 while a low amplitude is equal to a 0. AM is more susceptible to errors and bit flips because noise can affect the amplitude of the signal which is where the information is stored [34].

FM stands for Frequency Modulation, and instead of modulating the noise-prone amplitude, the frequency gets modulated instead. FM has a different frequency spectrum, which lowers the range and allows for a much higher bandwidth. A bit string modulated in FM uses different frequencies for 0s and 1s [33]. If the modulation scheme uses only two frequencies for 0s and 1s, then it is called Binary Frequency Shift Keying (BFSK), which is a simple modulation scheme and the most commonly used form of FSK [4]. If multiple frequencies are used for a set of bit sequences (000 111 010 etc.), it is called Multiple Frequency Shift Keying (MFSK). The advantage of MFSK over BFSK is that the system is more robust and more data can be sent over time [85].

For this project, it has been chosen to go with MFSK as the receiver's data rate is very low already because of technical limitations, and by using a faster modulation scheme, the data rate will be improved.

7.2 Implementation Focus

For the project, a complete prototype has been designed, in accordance with the requirements identified using the MoSCoW method (see section 7.1). This has required a design

[1]The amount of time each tone, representing a bit or a combination of bits, will last.

for both the encoding and transmitting part, as well as the receiving part, as shown on fig. 7.1. The details of the implementation will be described in sections 8.1 and 8.2, but in general, the focus has been on implementing a solid code for the conversion of images to audio signals in the encoding part, as well as the selection of information carrying frequencies and the conversion of them into an image again in the receiver part. Apart from this, different libraries have been utilized to capture an image, and to perform image compression and conversion of sound samples to a sound file in the encoding part, as well as signal analysis to extract the loudest frequencies at the receiver end. The libraries have been used, as it is not part of this student project to implement their functions from scratch. Furthermore, the use of these library functions have made room for a more thorough process in the development of the framework within which the library functions are implemented, as well as the development of the central parts of the program, mentioned earlier in this section.

7.3 Process demonstration

This section will describe and demonstrate the setup, as shown on fig. 7.2, as well as the process in order to achieve the compression, encoding, transmission and decoding that is sufficient for the requirements set for the project. The section will detail the process of compressing, encoding, transmitting, receiving and decoding the signal, in order to send and reassemble an image file. See also attachment 2 (demonstration video).

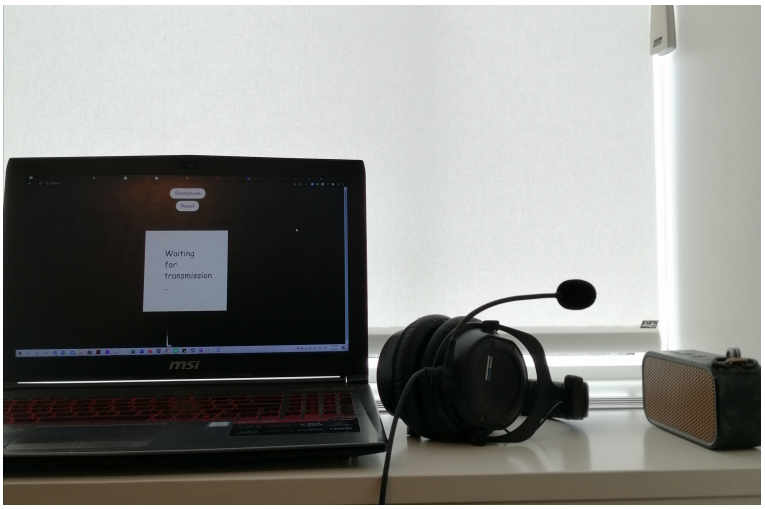


Figure 7.2: The picture shows the equipment that has been used when testing the program. In the left the laptop where the encoding part happens can be seen, and in the middle there is a headset where the microphone has been used to capture the audio from the speaker seen on the right side of the picture.

7.3.1 Program utilization

When the software has been compiled, it is then initialized and started in the terminal with the argument `-webcam 0` if the user wishes to use an existing image for transmission, or `-webcam 1`, if the user wishes to take a picture using a webcam to use for the compression and transmission. An example of this is shown in fig. 7.3. In this instance, the argument `-webcam 0` is used, which means that an image will not be captured, and a placeholder image will be used instead.

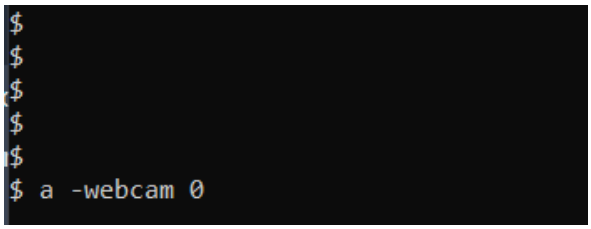


Figure 7.3: Screenshot of the program being initialized with parameters in compiler. When the argument `-webcam 0` is used, an image saved on the computer will be used for the transmission (i.e. no image will be captured with the webcam).

Next, the program will compress the image into a smaller resolution and file size using the JPEG format. After this step, the software will begin to convert the image file into a bit sequence. This bit sequence is then stored in a file called `tempBinary.txt`. This file contains the bit sequence of the entire image file.

When this step is completed, the software will draw upon the data in the `tempBinary.txt` file. The software will then encode sequences of 3 bits into certain frequency values, according to the `bitDuration` value in the code. If the `tempBinary.txt` data is not divisible by 3, the software will encode combinations of 2 bits and single bits at the tail of the data into separate frequencies.

Next, the software outputs the bit duration, binary file size, input length, number of samples, length of audio file in seconds and the bit rate in the console, as well as outputting the buffer in a .csv file for analysis. Lastly, the software confirms that it has outputted and saved an audio file.

This audio file now contains the entire audio signal with all of the bits from the `tempBinary.txt` file encoded into frequencies. This audio file can now be transmitted using any speaker connected to the computer.

7.3.2 Audio transmission

The audio transmission can be played from any device that has the `imageAudio.wav` file stored, and has a functional speaker. The only requirement in this step is that the speaker must exceed the amplitude of the background noise of the microphone used to record. When the receiver is primed to record, the audio transmission can begin. This will send out the frequency combination required to reconstruct the image sent.

While the transmission happens, another device is listening to the audio signal using the microphone. The receiver is running in a web browser on a website run on a local web server. The receiver website consists of three buttons that enable the user to operate the script. Due to security restrictions of the web browser, it is not possible to automatically run the receiver, since user confirmation is required in order for the browser to utilize a microphone. Once the server has been launched and permission to use the microphone has been given, the web application is ready to start receiving using the start/stop button. Additionally, it is possible to reset the receiver using the reset button if needed. When a recording has been initialized, the software registers all valid frequencies and separator tones. Once the transmission is over, the user clicks the stop button.

7.3.3 Audio Decoding

Once the stop button has been clicked, the audio is being processed. While the receiver was recording, the receiver knew what frequencies to listen for. As can be seen in table 8.3, certain frequencies corresponds to certain combinations of bits. In the console of the browser, the frequencies registered, as well as the bit sequence that has been decoded is printed.

Once the bit string has been received, it is then converted into a base64 string. The base64 string is then read by the web browser to construct the received image. The reconstructed image replaces the placeholder image afterwards.

Chapter 8

Documentation

In this chapter, a thorough description of the program will be made in order to document the functionality described in section 7.3. The chapter will be split into two sections; one for the transmitter part and one for the receiver part.

8.1 Encoding and Transmission (Tx)

The purpose of the following section is to exemplify and describe the code blocks of the software and to provide context to the different functions, the internal communication of the program, and the effect of each major code block. Each subsection of the Tx section will be introduced with a summary of the code block, followed by coding examples as well as any calculations that have been made to get the used variables. Lastly each subsection will, if applicable, conclude with a description of the output of the code block, as well as what the next step of the process is.

8.1.1 Overall Structure

The transmitter or Tx part of the solution has been encoded using the programming language C. It consists of four main steps; first capturing an image, then compressing the image, and then converting the image to a binary file (a .txt file of 1's and 0's) and at last converting this binary .txt file into audio. The overall structure of the transmitter and receiver can be seen as the top row in fig. 7.1.

In the transmitter part of the solution, the free open-source multimedia framework FFmpeg [1] has been used to capture an image, compress it afterwards and also to convert the image into sound. In order to make it work, the `ffmpeg.exe` must be available in the correct folder. The program is controlled primarily by command lines, which makes it easy to access its many libraries and functions. While FFmpeg is not the most lightweight library, it does offer a lot of features that is useful in our program. Another benefit, is that every functionality can be accessed just by executing a command using the C library function `system()` with the appropriate flags [1].

`CreateSoundFromImage.c` is the main file in the Tx part of the solution. In this, the functions for both capturing and compressing an image will be called, and the files for converting the image to binary data and the binary data to sound will be compiled and executed automatically, using the C library function `system()`, which takes a command as input parameter [15].

The first thing that happens in `CreateSoundFromImage.c` is that the two files, that are needed, will be compiled so that they are ready to be used when necessary. At first the `CaptureImage()` function will be called with the `argv[]` as an input parameter. In this case, the actual parameter will determine whether the webcam on the computer should take an image or not. The actual parameter can either be `-webcam 0` (don't capture an image) or `-webcam 1` (capture an image). `CaptureImage()` is a simple function that checks the actual input parameter. First some if statement check if the parameters has been entered correctly and then if it should capture an image or not. To capture an image, FFmpeg has been used, and it can be used as follows:

```
1 system("ffmpeg.exe -f dshow -y -i \"video=Lenovo EasyCamera\" -frames:v 1
      filename.png");
```

This command captures a single frame from the computer's webcam. In this case the webcam is named "Lenovo EasyCamera". It then saves the image as a .png-file called `filename.png`. An explanation of the different parameters inside the `system()` has been listed in table 8.1 [37].

FFmpeg parameter	**Explanation**
-f dshow	Force input to use Microsoft DirectShow API
-y	Overwrite output files
-i \"video=Lenovo EasyCamera\"	Input source
-frames:v 1	Number of frames to capture
filename.png	Output file name and format

Table 8.1: Schematic overview of the flags used for caption of image via webcam when using the FFmpeg library.

If webcam is disabled, it will be necessary that an image with the same filename is present in the correct location, else the program will return 1, meaning that it failed. Also, if webcam is enabled, but an error occurs, the program will try to capture an image 5 times, before returning 1 and printing "capture image failed".

After an image is either captured or another image is present, the function `compressImage()` for compressing the image will be called. The compression is done by again using the FFmpeg library, which can be used as follows, where -q:v and a number describes the quality of the compressed image. This will affect the size of the file and thereby also the transmission time. The command is shown below:

```
1 system("ffmpeg.exe -i filename.png -q:v 2 -vf scale=10:-1 compressed.jpeg");
```

Here `filename.png` will be compressed into the `compressed.jpeg`. In this case the image is being compressed using JPEG, but it is also possible to use another compression type when using FFmpeg. An explanation of the different parameters inside the `system()` has been listed in table 8.2 [37].

FFmpeg parameter	**Explanation**
-i filename.png	Input file
-q:v 2	Specify quality on a scale from 1 to 31. Affects file size.
-vf scale=10:-1	Add a scaling filter. Width = 10 px, height = auto
compressed.jpeg	Output file name and format

Table 8.2: Schematic overview of flags used for compression of image.

Earlier in section 6.3 the general idea of compression was described and in order to reduce the size of the image, which means also reducing the number of bits to be converted to audio, a type of compression was needed. Here the JPEG compression has been chosen and it will therefore be briefly described in the following section.

JPEG Compression

As mentioned in section 8.1.1, JPEG has been chosen as the type of compression when compressing an image in the Tx part of the solution. The parts of JPEG compression will only be briefly described as the FFmpeg library does all of the compression in the solution. The overall concept of compression has been described in 6.3, this including a description of lossy and lossless compression. JPEG uses a lossy type of compression and it can be done following different steps; first the color space will be converted from RGB (channels for red, blue and green) to YCbCr (channels for brightness Y and the color information in Cb and Cr). The brightness Y is kept as it is, but the color information Cb and Cr is

usually scaled down (it can either be kept, keep half of it or keep a quarter of it). This part is called downsampling [50].

The Y, Cb and Cr channels will then be split into 8x8 blocks of pixels, which will then be transformed from the so-called spatial domain, which is where the image is now, to the frequency domain. The spatial domain means the domain where a position in a block is also representing a position in the image. When transforming from the spatial to the frequency domain the position in a block will represent a frequency band in the block of the image. This transform from the spatial domain to the frequency domain happens by the use of the discrete cosine transform (DCT). DCT is a mathematical expression which is the most widely used linear transform in data compression. The DCT is a lossy form of compression algorithm and due to it being a lossy form, it means that the compression ratio will typically be higher than that of a lossless compression form. Compression ratio has also been explained in section 6.3. Overall the DCT expresses a sequence of data in terms of a sum of cosine functions at different frequencies [50].

The next step is quantization, where some information of the frequency will be discarded using quantization tables (one 8x8 table for brightness information and one for color information). A quantization table will be used to divide each frequency value of the image's 8x8 blocks by a corresponding number in the corresponding quantization table - the result is then rounded to the nearest integer. The numbers in the quantization tables depend on the desired quality (low number when high quality and reverse), also the numbers for the high frequency information will be greater than for the low frequency information. The numbers in the brightness table will overall be smaller than in the color table. The values in the 8x8 blocks will be listed using a zig-zag pattern that sorts the numbers from the lowest frequencies to the highest. After this Huffman encoding will be used to compress the quantised blocks [50, 70].

Huffman coding can be described as one of the most basic, lossless compression methods that can be used in text compressing or image compression. Huffman coding is based on the frequency of a data item, where the data that occur more frequently will have a lower number of bits to encode the data. The basis of this coding is a code tree (Huffman tree) which assigns short bit strings to symbols that are most frequently used and long bit strings to the symbols that are rarely used. Each symbol is encoded with a variable length from the Huffman table. Typically, Huffman codes compress data very effectively and data savings between 20% to 90% are expected depending on the data type [27].

As mentioned in 8.1.1 it is possible to choose which quality the compressed image should be, and this is done with the parameter -q:v and a number, when executing FFmpeg. The lower the number, the higher the quality of the image, which therefore means a bigger file size [37]. If an image (not compressed) has a file size of 1.45 MB and it then gets

compressed with -q:v 1 and scale 100, then the size of the file will be 2.95 KB. When quality is set to 1 it means that the quality is high. If the image is compressed with -q:v 20 and scale 100 the file size will be only 840 bytes, this is due to the quality being set to 20, meaning that the quality will be quite low, therefore resulting in a smaller file size with less details.

8.1.2 Conversion of Image to Binary Text File

The next step is to convert the compressed image to binary data. The code in the file `image2binary.c` works by reading the image bit by bit and copying the bits into a text file representing the image. The image is opened in read binary (`rb`) mode, which is used when reading non-text files, and the txt file is opened in write (`w`) mode. First, two pointers will be initialized to the beginning of the two files. Then the C library function `fseek()` will be used to find the length of the image file and `fread()` will be used to read through it and then saving the bytes in an array, making it possible to loop through all of the bytes.

```
for(imageIndex = 0; imageIndex < imageFileSize; imageIndex++){
  for(i = 0; i <= 7; i++){
    if(imageBytes[imageIndex] & (1 << (7 - i))){
      fputc('1', txtPtr);
    }
    else{
      fputc('0', txtPtr);
    }
  }
}
```

In the for loops shown above, we loop through all of the bits in each byte. By using bitwise operators we can get each single bit of the image and write it to the txt file using the C function `fputc()`. The outer for loop loops through each byte, and the inner for loop loops through each of the 8 bits that are in every byte. The if statement's condition is: `imageBytes[imageIndex] & ((1 « (7 - i))`. This means that for every iteration, 1 will be left bit shifted and then compared to the byte of `imageBytes[imageIndex]`. If the value of the i'th bit in `imageBytes[imageIndex]` is equal to 1, a 1 will be added to the text document. Else if the value is not equal to 1 (i.e. it is equal to 0), a 0 will be added to the document. When both loops terminate, the two files will be closed and the bits in the text file, that has been produced, are now ready to be converted to audio. Inspiration for this part of the code has been found on [25].

8.1.3 Conversion of Binary Text File to Audio File

The file `binary2sound.c` contains the part of the program where the conversion from a binary file to an audio file happens. The bits will be converted to different audio tones based on a MFSK modulation scheme, where combinations of three consecutive bits are converted at a time according to table 8.3. This part takes a .txt-file containing the binary representation of a compressed image as an input and returns a sound file ready for transmission.

Bit combination	Frequency, Hz
separation tone	440
000	880
001	1320
010	1760
100	2200
011	2640
101	3080
110	3520
111	3960
00	4400
01	4840
10	5280
11	5720
0	6160
1	6600

Table 8.3: Schematic overview of the corresponding bit combination and transmission frequencies. The first frequency of 440 Hz is reserved for the separation tone, and the last six frequencies are only used in the case, where the total number of bits for transmission is not divisible by 3.

In order to perform the conversion, the program first computes sets of samples containing the amplitude at a given time. This is done for each frequency over the temporal span of one bit duration. The bit duration is given as a number of samples and hence depends on the sampling frequency. The sampling frequency is equal to the number of samples per second. As most acoustic hardware use a sampling frequency of 44,100 Hz [77], a sampling frequency close to this will be preferred. This also complies with Harry Nyquist's so called Nyquist rate, stating that the sampling frequency must be slightly higher than two times the highest frequency in the signal [77]. In this case, according to table 8.3, the highest frequency is 6600 Hz, and as $2 * 6,600Hz = 13,200Hz \leq 44,000Hz$, a sampling frequency of 44,000 Hz will be proficient.

In order for the bit duration to be aligned with the framerate (which is relevant for the

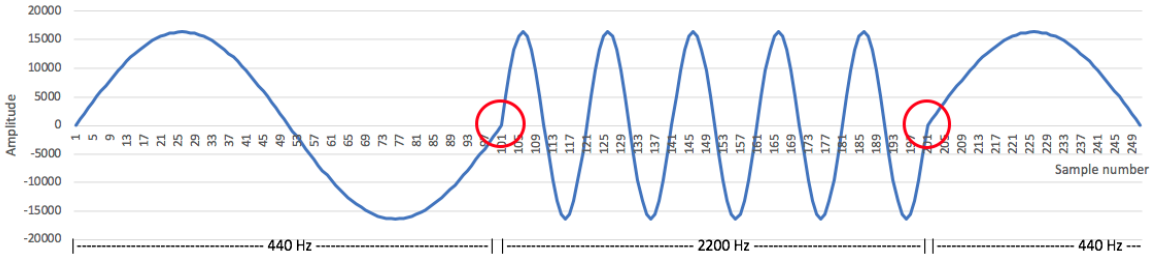
receiver part described in section 8.2), it is computed as a product of the ratio between the sampling frequency and the framerate, e.g., $\lfloor \frac{44{,}000Hz}{144Hz} \rfloor = 306$, multiplied by a scalar that indicates for how many frames, each tone should last.

The samples are stored in an array and will be copied into a buffer in the order that the bit combinations, that they represent, appear in the text file. Apart from the samples representing bit combinations, a sample is also computed for the separation tone. This tone is added to the buffer as the first tone, so on indices 0 to bit duration `bd` minus one, and is then added between every tone carrying information. This is done to ease the decoding process. In conclusion, this means that on indices in the interval $[(2k)bd; (2k+1)bd-1]$ only samples representing the separation tone will be found, whereas the information carrying samples in the buffer will be found on indices $[(2k+1)bd; (2(k+1))bd-1]$, where k is a non-negative integer.

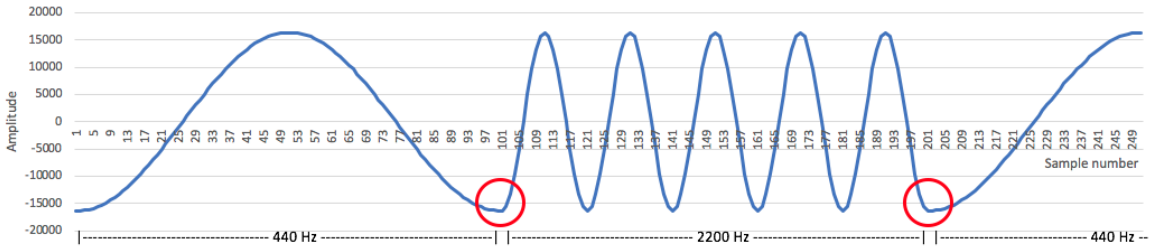
The sets of samples for each frequency consist of a number of samples representing the amplitude of a given wave at a given time. The number of samples per set is equal to the bit duration, and the samples provide a discrete representation of the sine wave corresponding to the frequency of each set. The amplitudes are calculated in accordance with the continuous sine wave function, as shown in eq. 8.1 [77]:

$$g(t) = A\sin(2\pi ft + \theta) \tag{8.1}$$

Eq. 8.1 provides the amplitude of a sine wave at any given time. Here, f represents the frequency and t time. θ is the phase, which in this case should be equal to $-\frac{\pi}{2}$, as this will ensure, that the transition between two tones happens when the wave is horizontal, i.e., when the amplitude has the highest absolute value. With a phase shift of $-\frac{\pi}{2}$, the transition between two tones will happen in the bottom of a trough. See fig. 8.1 for a comparison of the sine waves with phases equal to 0 and $-\frac{\pi}{2}$, respectively. The effect of the phase shift on the produced sound can also be seen on fig. 8.2, that compares the spectra of the non-shifted and the shifted signals.

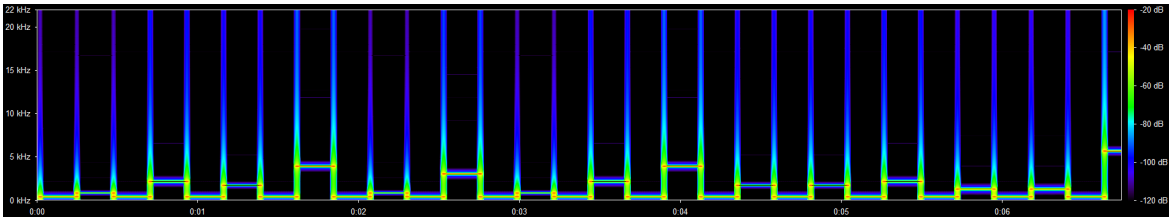


(a) Combination of different sine waves, all with $\theta = 0$.

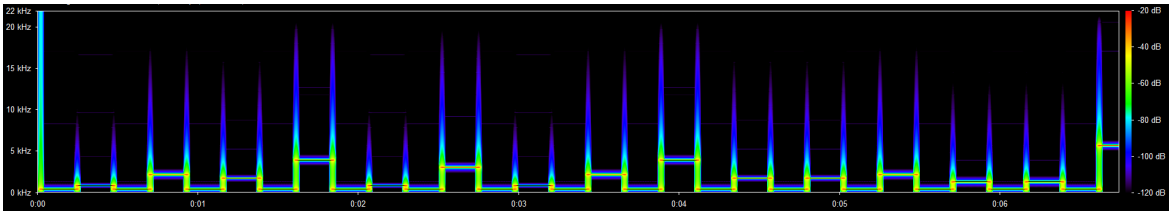


(b) Combination of different sine waves, all with $\theta = -\frac{\pi}{2}$.

Figure 8.1: Graphs of samples created for two different tones (440 Hz and 2200 Hz) with a bit duration of 100 samples, and a sampling frequency of 44.000 Hz. The red circles point out the transitions between two tones. From this, it can be seen that with a phase of 0, the transition causes a sudden break on the curve, as it happens when the amplitude is also equal to 0. On the other hand, when the phase is shifted, the transition can happen when the curve is in a horizontal position, which allows for a cleaner transition.



(a) Combination of different sine waves, all with $\theta = 0$.



(b) Combination of different sine waves, all with $\theta = -\frac{\pi}{2}$.

Figure 8.2: Spectra of samples created for multiple tones with a separator tone at 440 Hz. The bit duration is 1,000 samples, and the sampling frequency is 44,000 Hz. From these, it can be seen that with a phase of 0, the transition causes a quite a lot of noise, represented by the vertical peak at every transition. When the phase is shifted, the transition is obviously happening in a cleaner way, as less noise is created as a result of the transition. The representation has been made in a program named Spek.

A, in eq. 8.1, represents the amplitude, which should be high enough for the signal to be well above the noise level. In this project, an amplitude of 16,383 was chosen. Regarding the frequency, f, a set of samples has been calculated for each of the frequencies shown in table 8.3. The time t has been calculated by diving the sample number (values running from 0 to bit duration) by the sampling frequency, f_s. This leads to the following rewriting of eq. 8.1:

$$g(t) = A\sin(2\pi f\frac{j}{f_s} - \frac{\pi}{2}), \tag{8.2}$$

where j is the current sample number, meaning that j is an integer, and $0 \leq j <$ bit duration.

When these calculations have been made, the samples can be copied into the audio buffer, that must be an array large enough to contain two full sets of samples per three bits in the input file, additional to the initial set of separation tone samples. This is done with the following set of code[1], where the while-loop continues until all characters in the binary string have been converted to an audio sample:

```
addSeparatorTone();
while (j < binaryFileLen){
  addBitstringTone();
  addSeparatorTone();
  j += 3;
}
```

In order to know what set of samples should be added to the buffer, the next three characters in the binary file are compared to the possible bit combinations shown in table 8.3. This is done with the `strcmp()` function from the C library `string.h` that returns 0 if two strings are equal to one another. Notice again, that the first set of samples added to the buffer is the set corresponding to the separation tone, that is also added between every information-carrying tone.

Finally, the buffer is converted to an audio file, e.g., in the .wav or .flac format. This is done using the FFmpeg library as shown in the code section below. These lines of code have been written on basis of the code described in [14].

```
FILE *audioPtr = popen("ffmpeg.exe -hide_banner -loglevel error -y -f s16le -
    acodec pcm_s16le -vn -ar 44000 -ac 1 -i - ../tempFiles/imageAudio.wav", "
    w");
fwrite(buffer, 2, N, audioPtr);
pclose(audioPtr);
```

[1]For readability, the code section has been slightly modified, e.g., input parameters have not been included

First, a pipe to FFmpeg has been created using the function `popen()` with the flags listed in table 8.4. After this, the function `fwrite()` has been used to send the contents, that are to be converted to audio, into the pipe [14]. The input to `fwrite()` consists of the buffer-array, the size of each element in the buffer measured in bytes, the number of elements in the buffer and finally the pointer to the pipe.

FFmpeg parameter	**Explanation**
-y	Overwrite output files
-f s16le	Force input to be signed 16-bit integers in little endian format
-acodec pcm_s16le	Specify audio codec
-vn	Tells FFmpeg that it is only audio - no video
-ar 88000	Specify sample rate
-ac 1	Number of audio channels - 1 corresponds to mono
-i -	Tells FFmpeg to read from standard input - in this case it reads from the pipe.
out.flac	Output file name and format

Table 8.4: Schematic overview of flags used for pipe.

8.2 Reception and Decoding (Rx)

The receiving or Rx part of the solution has been encoded using the programming language JavaScript. For analyzing the recorded sound in the program, the p5.js open-source library [66] in JavaScript has been used. The p5 library makes it possible to easily record audio from the computer's microphone, and with a built-in function for making a Fast Fourier Transform (FFT), it is possible to analyze the audio as well as drawing a spectrogram. The Rx part consists of a server and a HTML page in which the JavaScript file is embedded.

8.2.1 Server

A local website stored as a file on your computer cannot utilize a microphone input. In order to receive a microphone input on a website, a web server needs to be utilized. The web server that hosts the website, has the ability to ask for permission to gain access to a microphone input of a computer. This is essential for the receiver as it is being run and executed in the web browser, where the p5 library used for handling the microphone input requires the website to be hosted on a web server in order to function. The web server used for the receiver is an HTTP server which is being run with Node.js. Node.js was chosen as

the authors of the report were provided with code for a simple server, previously in their semester, which has been modified and utilized in this project.

In order for the server to run locally, the hostname has been initialized to 127.0.0.1, and the port set to 3000. 127.0.0.1 is the local host IPv4 address for loop back traffic. 3000 is an arbitrary port chosen due to it not having elevated privilege, and being relatively safe to experiment with.

```
server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname}:${port}/`);
```

Here, the method `server.listen` is being used, to create a server that listens to the specified port variable (3000) and IP address using the hostname variable (127.0.0.1). Once this is done, the server prints a message stating that the server is running at the specified values.

In the `processRequest()` function, the server then responds to a GET request for the HTML file `Rx_Window.html`, in the following code excerpt:

```
case "GET":
    console.log(req.url);
    switch (req.url) {
        case "/":
            fileResponse(res, "Rx/Rx_window.html");
            break;
        default:
            fileResponse(res, `${req.url}`)
            break;
    }
    break;
```

The HTML file contains the reference to the p5 library in the head of the HTML file, while the `script.js` is referenced in the body of the document. The `script.js` contains the bulk functionality of the web application.

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
       initial-scale=1.0">
    <title>rx</title>
    <script src="Rx/p5/p5.js"></script>
    <script src="Rx/p5/p5.sound.js"></script>
</head>
<body>
    <button id="startpause">start/pause</button>
```

```
11      <button id="reset">reset</button>
12      <img src="" id="img" style="width: 50%;">
13      <script src="Rx/script.js"></script>
14  </body>
```

At this point in the process, the server has been created, the HTML file has been initialized, and the script for the recording and analysis of the transmission has been initialized, ready to begin recording and decoding.

8.2.2 Conversion of Recorded Sound to Binary File

The p5 library makes it possible to record and analyze input coming from the computer's microphone (or a different input source). This happens when calling the method `fft.analyze()`. FFT stands for The Fast Fourier Transform, which is an analysis algorithm that isolates audio frequencies, and it is being used when working with discrete values [67]. `fft.analyze()` computes the amplitude values in a frequency domain and returns an array of the amplitudes referred to as bins (2048 bins in total). The array stores the amplitudes of the frequencies ranging from the lowest frequencies that humans can hear to the highest frequencies that humans can hear (i.e. from 0 to 22,000 Hz). This makes it possible to find the different transmitted frequencies [67]. The registration of the transmitted frequencies is performed in `draw()`, that is a function from the p5-library, that runs once every frame. In `draw()`, the `fft.analyze()` is used to compute the amplitude of the frequencies in the microphone input. The amplitudes are sorted into an array (in the code below, the array is called `spectrum`), where each position represents a small interval of frequencies, e.g., from 0 Hz to 11 Hz. When this has been done, the interval with the highest total amplitude is found, using the for loop, showed in the following code section, to run through all positions of the array.

```
for (let i = 0; i < numberOfBins; i++) {
  let thisAmp = spectrum[i];
  if (thisAmp > maxAmp) {
    maxAmp = thisAmp;
    largestBin = i;
  }
}
```

The index of the position of the highest amplitude will be used to calculate the lowest frequency in the corresponding frequency interval by multiplying the index, `i`, with half the sample rate divided by the number of bins, as described in eq. 8.3.

$$\text{loudest frequency} = i \cdot \frac{\frac{\text{sample rate}}{2}}{\text{number of bins}} \tag{8.3}$$

This is the value that will be returned, but of course the actual value of the single frequency with highest amplitude can vary from the computed value.

If the recording of the signal has begun (i.e. if the start/stop-button has been pressed to start the recording) and if the calculated "loudest frequency" is either a valid (information carrying) tone or a separation tone, then it will be added to the array `spec` described in this section. The `spec[]` array now contains the loudest frequencies that have been detected. When the recording has been stopped, different while loops will start comparing the frequencies in `spec[]` to the constant array containing valid frequencies (`validFreqs[]`, see below). When a frequency has been identified, it will be returned and passed on to another function, that will add the corresponding bit string to the string that will in the end contain all the transmitted bits.

```
1 const validFreqs = [880, 1320, 1760, 2200, 2640, 3080, 3520, 3960, 4400,
      4840, 5280, 5720, 6160, 6600];
2 const bitCombinations = [ "000","001","010","100","011","101","110","111",
3                           "00","01","10","11",
4                           "0","1"];
```

The while loops mentioned above are structured as shown in the code excerpt below, and they all serve different purposes. When the recording has been stopped, they start by sorting out possible background noise before the signal has begun. This is done by comparing the frequency `spec[x]` that we are looking at, and the next frequency, `spec[x+1]`, to the frequency of 440 Hz. This is seen in line 2 to 3 in the code below. If these frequencies aren't 440 Hz, which is the separation tone, it is just noise, and it will therefore be skipped. This is continued until there has been two registrations of a separation tone in a row, indicating that we have reached the beginning of the information signal.

After this, an outer while loop (line 5 to 24) will run as long as the index `x` is still within the length of the `spec[]` array. The remaining of the code, consisting of other while loops and if statements, seen below, happens inside of this outer while loop.

From line 6 to 10 a while-loop and an if-statement checks if the separation tone has been identified multiple times in a row. If this is the case, then the separator tone has been identified and the boolean named `separator` is assigned to true.

On line 11 to 12 if !separator is true then the frequency is skipped - this is in case the separation tone has not been identified yet. After this the variable `current` is assigned to the frequency of `spec[x]`.

On line 16, the scope of an if statement is entered if the frequency is valid and if the separator tone has been detected. Inside the body, the separator will be assigned to false. After this, the valid frequency, `spec[x]`, will be pushed to the `identifiedFreqs` array, and the identified frequencies will be converted to bits by calling the `freqToBits()` function, that will be described later in this section, and adding the output of the function to the total bit string. The last while loop on line 20 skips all of the frequencies that are not 440 Hz, namely the separation tone, meaning that we are ready for the next separation tone. All of this continues until all of the audio has been identified.

```
1  if (!recording) {
2    while (!((compare(spec[x], 440) && compare(spec[x + 1], 440))) && x < spec.
       length) {
3      x++;
4    }
5    while (x < spec.length) {
6      while (compare(spec[x], 440) && x < spec.length) {
7        if (compare(spec[x+1], 440))
8          separator = true;
9        x++;
10     }
11     if (!separator)
12       x++;
13
14     current = spec[x];
15
16     if (isValidFreq(spec[x]) && separator) {
17       separator = false;
18       identifiedFreqs.push(isValidFreq(spec[x]));
19       bitstring += freqToBits(identifiedFreqs[identifiedFreqs.length - 1]);
20       while (!compare(spec[x], 440) && x < spec.length) {
21         x++;
22       }
23     }
24   }
25 }
```

To perform the operations described above, several subfunctions have been used. One of them is the function `compare()`, that compares a frequency to another frequency with a margin of $\pm$40Hz. If the frequency is in this spectrum, true will be returned, else false.

```
1  function isValidFreq(freq) {
2    for (let i = 0; i < validFreqs.length; i++) {
3      if (compare(freq, validFreqs[i])) {
4        return validFreqs[i];
5      }
6    }
7    return false;
```

Another subfunction is the `isValidFreq()` function, which can be seen above, that takes a frequency as its input parameter. A for loop iterates through the `validFreqs` array and it compares the inputted frequency to all of the valid frequencies with a margin of $\pm$40Hz. The reason why it checks with a margin is due to the way that the `fft.analyze()` is analyzing the audio.

```
function freqToBits(freq) {
  for (let i = 0; i < validFreqs.length; i++) {
    if (validFreqs[i] === freq) {
      return bitCombinations[i];
    }
  }
  return 'XXX';
}
```

The last subfunction that will be described here, is the function `freqToBits()`, as seen above. Each identified frequency is compared to the frequencies in the `validFreqs` array, and the index of its position in the `validFreqs` array is used to identify the corresponding bit combination in the `bitCombinations` array - this is how the frequency is converted to bits.

8.2.3 Conversion of Binary String to Image

When using JavaScript, it is possible to directly add image data as a base64 string to the HTML `src` attribute, which will then be displayed directly on the HTML page. The following code snippet converts the bits extracted from the microphone to base64:

```
function showImage(bitstring) {
  let encodedData = btoa(binaryToString(bitstring.replace(/(.{8})/g, "$1 ")))
    ;

  document.getElementById("img").src = "" +
    encodedData;

  function binaryToString(str) {
    let newBin = str.split(" ");

    for (i = 0; i < newBin.length; i++) {
      binCode.push(String.fromCharCode(parseInt(newBin[i], 2)));
    }
    return binCode.join("");
  }
}
```

Inspiration for the code has been found on: [26, 45].

On line 2 the bits are converted into bytes with the `replace()` method. Using a regular expression (/(.8)/g,"$1"), a space is inserted after every 8th bit. Then `binaryToString()` is called with the input parameter being the string of bytes. The function `binaryToString()` converts the bytes to a string of unicode characters, which is necessary because the method `btoa()` creates a base64 encoded string from a binary string (a string in which each character in the string is treated as a byte of binary data) [99].

The conversion from bytes to a binary string happens as mentioned before in the function `binaryToString()`, which can be seen from line 6 to line 13 in the code shown above. First, the byte string will be split at each space and stored in an array. After this, the for loop iterates from 0 to the length of the `newBin` array. In every iteration the value of `String.fromCharCode(parseInt(newBin[i], 2))` is pushed to the `binCode` array. The function `parseInt` takes the parameter newBin[i], that is the value to be parsed, and the parameter 2 which is the integer that represents the radix[2], and converts the byte into an integer. The integer is then converted to a unicode character using `String.fromCharCode`. Finally, at line 10 the array will be stitched together to a string with no spaces and returned.

When this is done the `btoa` method will be called with this string as its parameter. `btoa` encodes a binary string in base64. Base64 is an encoding scheme that makes it possible to convert different characters into a more readable string by replacing every six characters with one letter or character [96]. An example of how the conversion of one byte could happen is shown in table 8.5.

Byte	01010101
parseInt	85
String.fromCharCode	U
btoa	VQ==

Table 8.5: An example of how the conversion from a byte to base64 works. Starting with the byte "01010101", the right column is the result of the function in the left column.

In the end, the image is presented on the client as described earlier, and so the transmission has been completed.

[2]The number of unique symbols used to represent a number [98]

Chapter 9

Test

The purpose of this section has been to run different tests on the developed solution in order to make sure that the different functions work as expected, both separately and together. First, different tests on the Tx part have been performed, these including the compression, conversion from image to bits and the conversion from bits to audio. After this, tests on the Rx part will be performed, to ensure that the program reads the information carrying frequencies as expected, and correctly converts these into a binary string. At last, a test will be performed on the full program.

Unless something different is stated, the program has been tested with a sampling frequency of 44,000 Hz and a bit duration of 9,180 samples. For sound transmission, a Beyerdynamic Custom Game headset has been used. The microphone has been placed inside the ear cups for optimal transmission conditions, during the testing and development of the program. In the final test of the full program, a Kreafunk aSPORT Bluetooth loud speaker with a 3 W driver has been used to transmit the sound.

9.1 Tx

9.1.1 Compression

When compressing an image, FFmpeg is, as mentioned in section 8.1.1, being used. A small test on the compression can be made by calling the following code and then comparing the original image to the compressed image. A .png image with a file size of 104 KB

has been compressed to a JPEG image, with the dimensions 150:automatic and a quality setting of 2, meaning that the quality of the image is relatively high. After the compression the file size of the compressed image was 3.18 KB.

```
compressingImage = system("ffmpeg.exe -hide_banner -loglevel error -y -i ../
    tempFiles/tempImage.png -q:v 2 -vf scale=150:-1 ../tempFiles/
    tempImage_compressed.jpeg")
```

The images were then compared, as seen in fig. 9.1. Here it can be seen that the compressed image is more blurred than the original image, which was also expected. If the quality had been set to more than 2, the difference between the two images would have been bigger, as it would mean a lower quality on the compressed image.

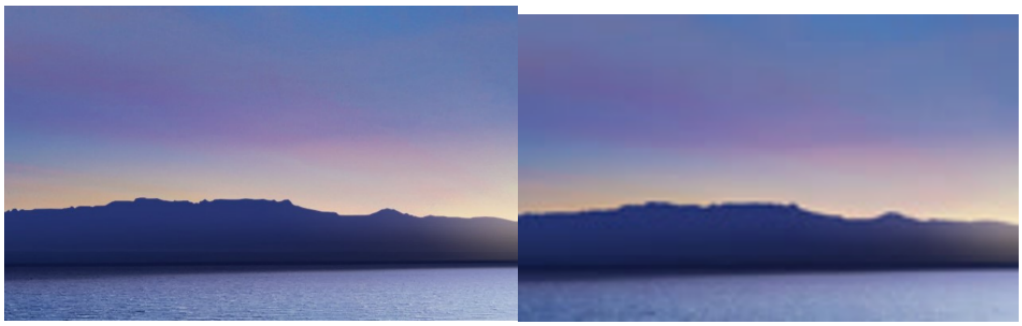


Figure 9.1: The original image on the left and the compressed on the right - the scales on the images have been changed afterwards in order to place them next to each other

9.1.2 Conversion of Image to Binary Text File

A simple test on the function that converts an image to bits has been made. A new .c file was made containing only the for loop in which the conversion happens, as well as a simple char string. The code can be seen below.

```
int main(void){
  char string[6] = "Hello";

  for(index = 0; index < strlen(string); index++)
  {
    for(i = 0; i <= 7; i++)
    {
      if(string[index] & (1 << (7 - i)))
      {
        printf("1");
      }
      else{
```

```
            printf("0");
        }
    }
  }
  return 0;
}
```

After running the code the 1's and 0's were printed directly to the terminal. The printed output was: 01001000011001010110110001101100011011111. This was then inserted into an online binary to text converter [9]. The compiled output can be seen in fig. 9.2 and the conversion from the online converter can be seen in fig. 9.3.

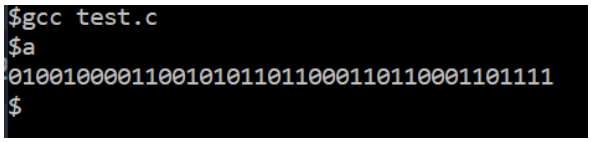


Figure 9.2: Output after compilation

On fig. 9.3, it can be seen that the outputted string containing the binary data is translated to "Hello", which is equal to the inputted char array.

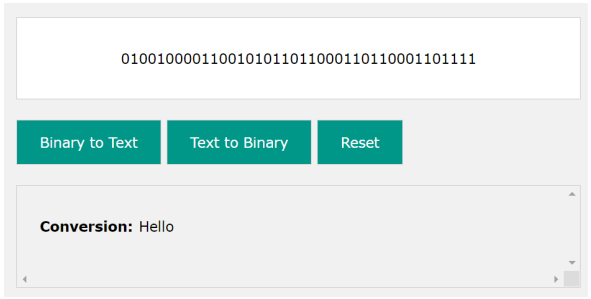


Figure 9.3: Online conversion from bits to letters

It then needs to be shown that the image can be printed as characters instead of bits and if this is the case, the function must work as expected. In order to show this, a small change in the for loop had to be made. Instead of writing 0's and 1's to the .txt file, the image was written directly to the .txt file as characters. See the code below.

```
for(imageIndex = 0; imageIndex < imageFileSize; imageIndex++){
    fputc(imageBytes[imageIndex], txtPtr);
    printf("%c", imageBytes[imageIndex]);
}
```

When opening the .txt file, different characters was written, see fig. 9.4.

Figure 9.4: Image printed as characters instead of bits

As it has also been shown that the function is able to convert characters to bits, it can therefore be concluded that the function `image2binary()` for converting an image into bits works as expected.

9.1.3 Conversion of Binary Text File to Audio File

A method of testing that the software is both encoding the correct frequencies and transitioning correctly between frequencies (i.e. with the specified phase shift), is to write and store the values of the audio buffer as a .csv file. This provides a method of numerically analyzing the audio signal that has been assembled and a way to plot a 2D graph that represents the waveform over the duration of the audio file.

The .csv file has been written using the code seen below:

```
printf("Adding values to .csv file...\n");
FILE *csvfile;

csvfile = fopen("buffer.csv", "w");
for (int i = 0 ; i < N ; ++i){
  fprintf(csvfile, "%d\n", buffer[i]);
}
fclose(csvfile);
printf(".csv file saved\n");
```

The .csv file produces a list of the amplitude of each sample, which ranges from the numerical value 16,383 to -16,838. An excerpt can be seen in fig. 9.5. The amount of time between each peak or trough determines the frequency. If a graph is made of the values of the .csv file, the waveform of the audio signal can be visualized. Each listing of an amplitude value in the .csv file equates to one sample.

Using the listing created in the .csv file, it is possible to plot a 2D graph that outputs the waveform of the audio. Using this visualization, it is possible to determine both the

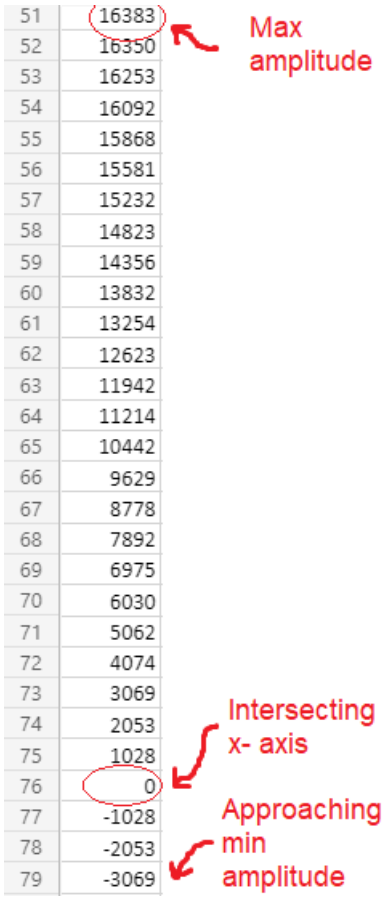


Figure 9.5: Screenshot of the .csv showing the stored amplitude values.

frequency of the signal at any given time, as well as identifying the transition to the next frequency.

For this test, two criteria are set, that must be met to verify this process of the program:

1. The frequencies in the graph must only consist of valid frequencies that are specified by the modulation scheme.
2. The transition between frequencies in the audio signal must happen at the minimal amplitude value of -16383.

In order to validate the frequencies outputted by the program, it is possible to plot a graph from the values that are stored in the .csv file. These frequencies must correspond with the valid frequencies that have been coded into the software, as shown in table 8.3. Each frequency corresponds to the binary section as they are listed.

When plotting a graph using the data from the .csv file, it is possible to make a calculation by counting the number of peaks of each tone and multiplying the peaks with 440. Doing this will equate to the frequency of the tone. This way, it is possible to visually confirm the frequencies of the data before it is even made into an audio file. Using this method, it is also possible to determine where in the audio signal that the transition takes place. The criteria that has been set dictates that the transition between tones must happen at the minimal amplitude of the signal. Fig. 9.6 shows an excerpt of the data of the audio buffer.

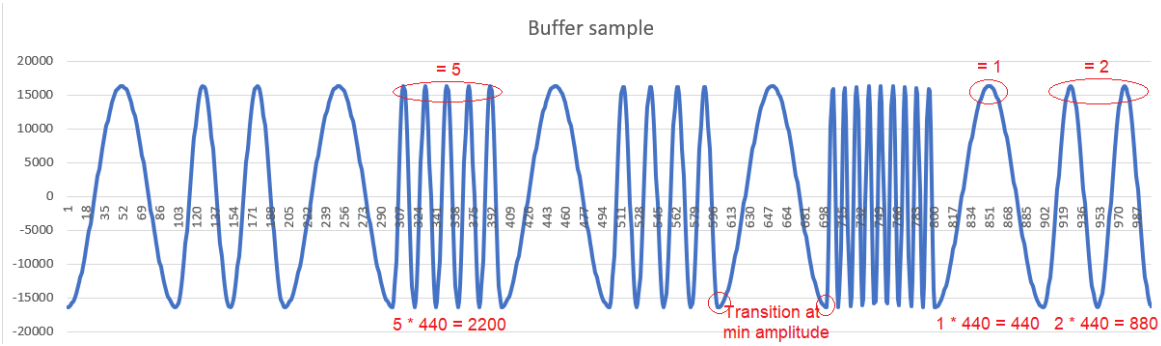


Figure 9.6: Graph using .csv buffer values showing amount of peaks of tones and the appropriate calculations showing frequency values. The bit duration was set to 100 samples, and the sample rate was 44,000 Hz, when this data was produced. Therefore, the number of wave peaks within 100 samples multiplied by 440 Hz equals the frequency of the wave in a given 100 sample interval.

Looking at the peaks of fig. 9.6, along with the calculations, it is possible to determine that all frequencies present in this excerpt meet the criteria of the first condition. It is also

apparent that the transition between tones does in fact take place at the minimal value of the amplitude, as is dictated by the Y axis.

The tones shown on fig. 9.6 are supposed to represent the following bits: `000 100 010 111 000 ....` This means that the frequencies `[440; 880; 440; 2200; 440; 1760; 440; 3960; 440; 880; ...]` should be on the graph. Counting the number of peaks to determine the frequency of each tone reveals that the conversion to sound has been done correctly.

Lastly, by looking at the .csv data, we can determine what the stored amplitude values of the audio buffer would be. In the screenshot from the .csv file shown on fig. 9.5, it is easy to observe the minimal and maximum values of amplitude.

By the method of this process of analyzing the .csv file of the audio buffer, we can determine that both the criteria that were set for testing have been met, and thus it can be confirmed that this section of the program works as intended.

This also concludes the test of the transmitter part, where it has been shown that the main functionality of compressing an image, translating it into a binary file and then computing a set of sound waves for audio transmission, works correctly.

9.2 Rx

9.2.1 Conversion of Logged Frequencies to Bits

In this section we test the functions which convert the sound frequencies into bits. We have six different scenarios to test, each with a different problem. To test this, we modified the receiver script to read the frequencies from six different arrays, one for each scenario.

Case 1

The first case is an example of a perfect reception of the sound, where we expect it to read the wake-up tone, and then translate all the valid frequencies after each separator.

```
1 case1 = [440, 440, 440, 2200, 2200, 2200, 2200, 440, 440, 440, 440, 1320,
      1320, 1320, 1320, 440, 440, 440, 440, 3520, 3520, 3520, 3520, 440, 440,
      440, 440, 4400, 4400, 4400, 4400, 440]
2
```

```
Expected frequencies = [2200, 1320, 3520, 4400]
Expected bitstring = 10000111000

Result frequencies = [2200, 1320, 3520, 4400]
Result bitstring = 10000111000
```

The result of this case was, as expected, a correct translation because the case was an example of an error free transmission.

Case 2

Case 2 is a scenario where the receiver does not identify the wake up tone. So we are expecting it to not translate the first valid frequency (2200 Hz).

```
case2 = [2200, 2200, 2200, 2200, 440, 440, 440, 440, 1320, 1320, 1320, 1320,
    440, 440, 440, 440, 3520, 3520, 3520, 3520, 440, 440, 440, 440, 4400,
    4400, 4400, 4400, 440]

Expected frequencies = [1320, 3520, 4400]
Expected bitstring = 00111000

Result frequencies = [1320, 3520, 4400]
Result bitstring = 00111000
```

As expected, the receiver does not detect the first valid frequency, which is 2,200 Hz, because of the missing wake up tone, and therefore the second valid frequency 1,320 Hz is the first to be translated.

Case 3

This case represents a scenario in which the receiver does not record a separator between two valid frequencies. This could happen due to noise or if the frequencies are transmitted too quickly.

```
case3 = [440, 440, 440, 2200, 2200, 2200, 2200, 1320, 1320, 1320, 1320, 440,
    440, 440, 440, 3520, 3520, 3520, 3520, 440, 440, 440, 440, 4400, 4400,
    4400, 4400, 440]

Expected frequencies = [2200, 3520, 4400]
Expected bitstring = 10011000

Result frequencies = [2200, 3520, 4400]
Result bitstring = 10011000
```

As expected, the receiver does not decode the signal correctly and skips the frequency at 1,320 Hz because it waits for the separator frequency.

Case 4

In this case we have inserted a noise tone which has a frequency that the program considers to be valid. As the noise frequency of 880 Hz is placed just after a separation tone, we expect that the program will translate the noise and therefore not translate the following (intended) valid frequency of 2,200 Hz.

```
case4 = [440, 440, 440, 880, 2200, 2200, 2200, 2200, 440, 440, 440, 440,
    1320, 1320, 1320, 1320, 440, 440, 440, 440, 3520, 3520, 3520, 3520, 440,
    440, 440, 440, 4400, 4400, 4400, 4400, 440]

Expected frequencies = [880, 1320, 3520, 4400]
Expected bitstring = 00000111000

Result frequencies = [880, 1320, 3520, 4400]
Result bitstring = 00000111000
```

As expected it translated the noise at 880 Hz and therefore skipped the translation of the valid frequency on 2,200 Hz.

Case 5

Case 5 is almost identical to case 3, but now it is missing two separators, so we have three valid frequencies in a row. We expect the receiver to only decode the first frequency correctly and skip the next two.

```
case5 = [440, 440, 440, 2200, 2200, 2200, 2200, 1320, 1320, 1320, 1320, 3520,
     3520, 3520, 3520, 440, 440, 440, 440, 4400, 4400, 4400, 4400, 440]

Expected frequencies = [2200, 4400]
Expected bitstring = 10000

Result frequencies = [2200, 4400]
Result bitstring = 10000
```

The result shows that the receiver correctly decodes the first valid frequency but skips the two following frequencies, and ends up with a incorrect bitstring.

Case 6

Case 6 has a noise tone at a valid frequency (880 Hz) before the separator tone but after a valid frequency, so we are expecting that it will not translate the noise at 880 Hz.

```
case6 = [440, 440, 440, 2200, 2200, 2200, 2200, 880, 440, 440, 440, 440,
    1320, 1320, 1320, 1320, 440, 440, 440, 440, 3520, 3520, 3520, 3520, 440,
    440, 440, 440, 4400, 4400, 4400, 4400, 440]

Expected frequencies = [2200, 1320, 3520, 4400]
Expected bitstring = 10000111000

Result frequencies = [2200, 3520, 1320, 4400]
Result bitstring = 10000111000
```

As expected the noise did not have any impact on the translation of all our transmitted valid frequencies.

Conclusion of tests

In conclusion, this part of the code works after our intention. If we take case 3, where a noise tone with a valid frequency was placed just before an information carrying frequency, it will skip the symbol which results in an incorrect output. Therefore it is important that the transmission happens in a silent environment to make sure we get a successful reception.

9.2.2 Reception and Conversion to Binary String

In order to test whether the signal has been received with or without any error, a comparison between the initial binary file and the binary file received can be done. Using an online tool (e.g., [24]), this can easily be achieved.

Fig. 9.7 shows that the sent and received binary strings are similar, except for an extra 0 bit attached to the end of the received signal. This is due to noise from not having stopped the recording when the transmission had finished. However, this is not an issue, since the first part of the transmission contains the meta data of the file, and thus the decoder knows how long the binary string is supposed to be. This means that any bits, that are picked up wrongfully after the audio transmission has ended, will automatically be discarded before reconstructing the image.

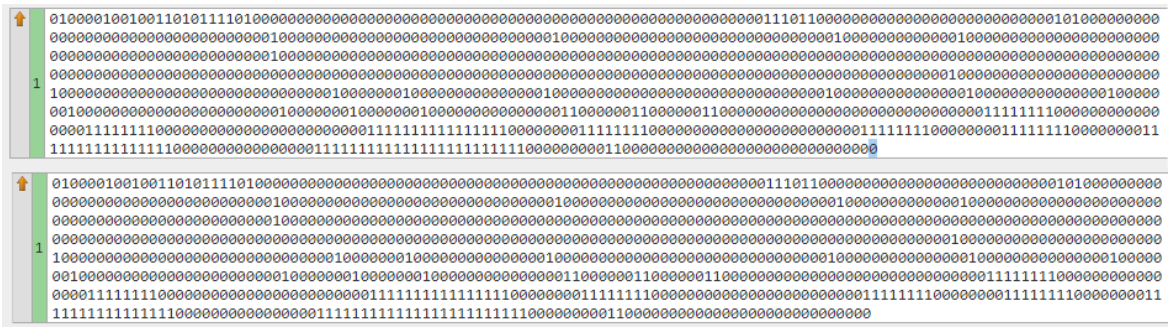


Figure 9.7: Screenshot of the sent and received binary string compared

9.3 Full Program

A test has been made on the full program. First an image was captured on a computer's webcam. The image was a .png type, and the file size was 256 KB before applying JPEG compression. The image was then compressed with the scale 100:automatic and quality 20, meaning that the quality of the compressed image is a lot lower than the original image. After the image has been compressed into a .jpeg file, the file size is 719 bytes. The original image and the compressed image can be seen in fig. 9.8.

Figure 9.8: The original image on the left and the compressed on the right - the scales on the images have been changed afterwards in order to place them next to each other

The compressed image was then converted into bits and these bits were then converted to audio. After this, the audio was transmitted, meaning that it was played, and the receiving unit then identified the valid frequencies and converted them to bits. The whole transmission took about 13 minutes and 20 seconds and the image was received containing

no errors. The image was shown at the client in the Rx part, see fig. 9.9. Here, the compressed image can be seen.

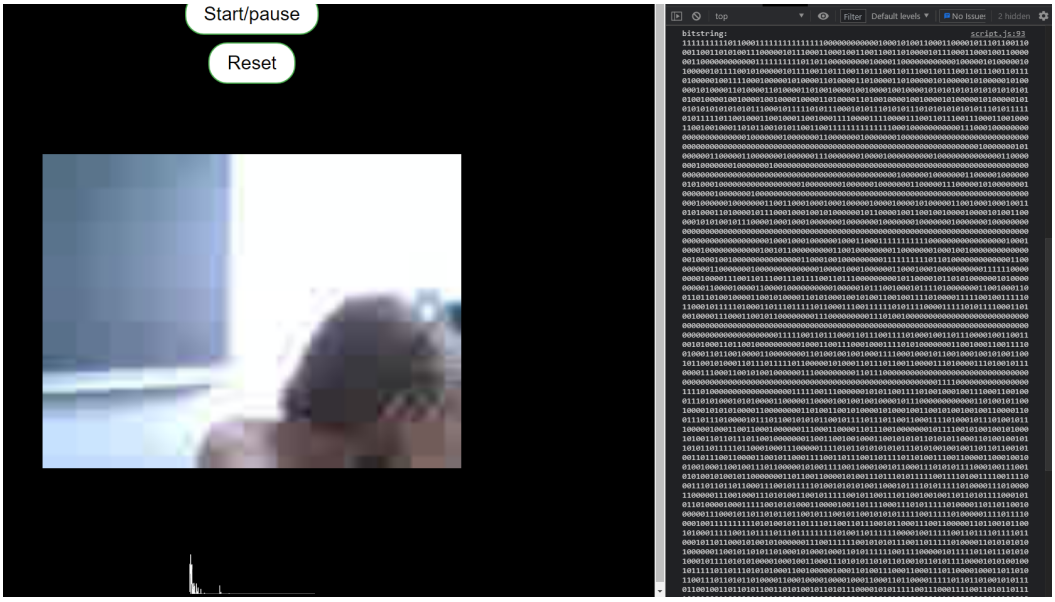


Figure 9.9: The figure shows the HTML page where the image was received and then shown. On the left the console can be seen, where the converted bits were printed.

The converted bits were printed in the console, and they were also compared to the bits of the compressed image from the Tx part. They were compared using [24], and the comparison can be seen in attachment 1. The input and output bit strings are exactly the same with the exception of the last character in the output string that is not present in the input. This is not a problem, however, as only the desired number of bits are read, when the image is displayed.

Overall the test on the full program shows that it works as intended, when there is no significant distance between the transmitter and the receiver.

The speaker used for testing might not be ideal for a long distance transmission of audio, and the quality when capturing the sound depends on the microphone that is being used in testing. The limitations in the equipment for the testing part directly affect the limits in distance of the transmission, in order to get the expected result. Despite these limitations in the equipment, different tests have been made with different distances between the speaker and the microphone to see how distance would affect the reception of the signal. To test this, a simple image with only one, coloured pixel was used (see fig. 9.11). Three tests have been made with different distances between the microphone and the speaker. One with the distance of 35 cm, one with the distance of 100 cm and one with the distance

of 155 cm (see fig. 9.10, where the two first tests are shown). The two first tests were successful however multiple tests at a distance of 155 cm proved unsuccessful. Deductively, the longer the distance the more challenging it becomes for the microphone to capture the audio correctly.

Figure 9.10: The two pictures shows different scenarios for distances between the speaker and the microphone, where the transmission and the encoded part were both successful.

In all of the scenarios, the same image was used, and since the two first tests were successful it means that the image was transmitted correctly. An example of one of the correctly transmitted images can be seen in fig. 9.11. In the last test, the image was not transmitted correctly and therefore not showed on the HTML page.

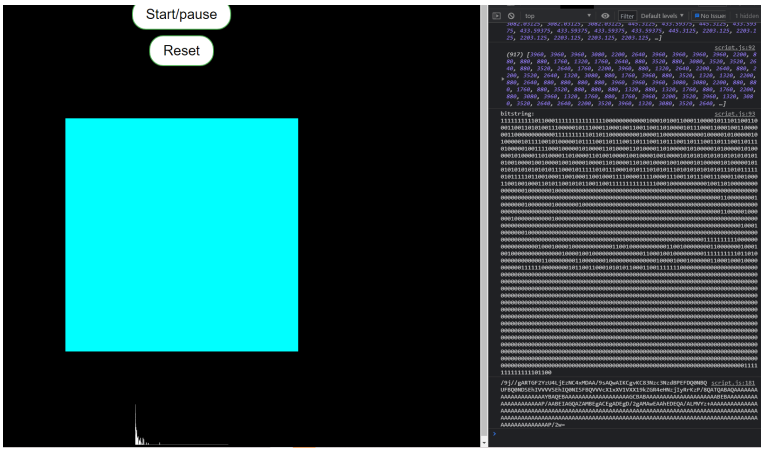


Figure 9.11: The screenshot shows a result for the tests in fig. 9.10.

According to the tests designed to prove the predictability and reliability of the software, we can conclude that the software largely behaves as intended, and serves well enough as a proof-of-concept, and functions within the requirements that have been set by the authors.

Chapter 10

Discussion

Throughout the previous sections of the report, the capabilities of the presented solution have been documented and tested. It appears that a solution has been put forward that is capable of encoding and transmitting an image via an acoustic channel, as well as receiving and decoding the signal. As shown in section 9.3, the proposed coding scheme has been tested above water with a quite simple hardware setup. Therefore, it has also only been tested on limited distances with successful reception on up to 100 cm. As sound propagates well under water, it is assumed that the same coding scheme would work for underwater communication, if it was set up with the required hardware.

Another limitation to the test setup is that both the speaker and the microphone have been static throughout the tests. It is expected that UUVs will be moving, when deployed in the sea, so for now it is unclear whether the coding scheme would be able to handle the reception of a signal transmitted from a moving platform. The effect of the movement on the signal can, however, be assessed using the doppler effect. The doppler effect is the effect of change in frequency of a wave when an observer is moving relative to the source. This poses a problem in wireless communication as it could change the frequency of the waves so that the receiver cannot decode the signal resulting in an increase of the bit error rate.

Based on the oceanographic research vessel R/V Falkor operated by the Schmidt Ocean Institute, it can be calculated how big the uncertainty due to movements could be. The R/V is a surface vessel capable of launching AUVs and ROVs. Using data from the Schmidt Ocean Institute website [78] and assuming a worst case scenario, where the ship and the UUV are moving directly away from or against each other at cruising speed, the observed

frequency can be calculated with the following equation:

$$f_o = \frac{V}{V+v} \cdot f_s \tag{10.1}$$

where f_0 is the observed frequency, V is the speed of the wave, v is the relative speed of the observer and the source, i.e. the receiver and transmitter. f_s is the frequency of the wave at the source. The ship has a cruising speed of 12 knots (6.173 m/s), a typical AUV has a speed of about 4 knots (2.057 m/s) and the speed of sound in water is about 1,481 m/s. The higher the source frequency, the stronger the doppler shift, so it is only necessary to calculate the shift for the highest frequency in use, which is 6,600 Hz:

$$f_o = \frac{1,481\frac{m}{s}}{1,481\frac{m}{s} + (6.173 + 2.057)\frac{m}{s}} \cdot 6,600Hz \approx 6,564Hz \tag{10.2}$$

In this case the doppler shift is 36 Hz, when the vehicles are moving away from each other. If the vehicles were moving closer to each other, the observed frequency would be $f_o \approx 6,637$ Hz, meaning that the doppler shift would be approx. 37 Hz. In conclusion, the doppler shift is significant in both cases, and it is worth taking into consideration. In our solution, we have a margin of error of 40 Hz meaning it would still work correctly in this case, assuming the receiver is precise. For further development, the program could dynamically change the transmitting frequency depending on the relative speed of the AUV to offset the doppler shift.

Another important point from the tests is that the program works well on short distances when the signal strength is well above the noise level. As shown in the six cases in section 9.2.1, the program does not handle noise, that happens to have a frequency that is considered "valid" (within the 80 Hz intervals around the 15 frequencies shown in table 8.3), very well. This might be a remarkable source of error, as these valid intervals cover more than 17 % of the frequency band from 0 to 7,000 Hz. As no error correction code has been implemented, even the smallest error in the transmission is fatal for the output of the program.

Finally, the program only works on computers with Microsoft Windows operating systems, meaning that it does not work on e.g. macOS or Linux, as the `system()` commands in the encoding parts of the program are not compatible with other operating systems.

Did we manage to answer the problem statement in a general sense?

Our problem statement was quite ambitious, and one we knew would be difficult to grasp and implement effectively. On one hand, we set out to cover a gap in the market, or at least make a connection that had not been achieved before. At the same time, we knew that

some of the technology and implementation that was already available, was well above anything we could realistically achieve.

The problem statement asks whether it is possible to base a transmission scheme on acoustic signals over a certain distance with a certain file size, and whether or not this can be achieved within a reasonable amount of time. In this instance "a reasonable amount of time" is entirely dependent on what the use case of the transmission is. If it is for navigation, then our solution does not suffice, given the data rate required to fulfill a navigational need is not met.

If the purpose had been to use the drone for data collection however, then the answer would be yes, since the alternative would be to wait for the drone to complete its mission before data can be collected.

Theoretically, the distance requirement can definitely be met, given the only requirement is a strong enough signal to overcome any noise between the sender and receiver, and additionally, audio propagates better under water. Given the fact that the transmission is possible using air as a medium, only speaks to the fact that transmitting our signal under water would ultimately prove to be less difficult, at least to our current knowledge.

How long does the transmission take, and is it viable for our use case?

We set out on an initial goal of wanting to attempt to transmit a video feed under water using acoustic waves, however this quickly proved to be nearly impossible with the knowledge and tools that we had available. Especially given the fact that we as a group have found no commercially viable implementations of such a system, it was a very ambitious task. Our natural response was to scale down, and we landed on rapid image transmission as our new goal for the project, and this proved, at least in theory, to be somewhat doable. As mentioned in the report, a benefit of rapid image transmission over a constant video feed, is that not every frame on a continuous video stream carries new data that is of use to the operator or data collector, and thus we could argue that sacrificing the smoothness of a video feed would still make our solution a viable one. Now the question becomes "How quickly can we transmit an image?", and that question leaves a lot up for discussion and further development. On one hand, we can transmit highly compressed and small images fairly quickly, but in doing so we lose a lot of useful information in the image, which basically makes it obsolete, at least based on our judgement.

We are very aware that the code that we have written and the solution we have implemented has a lot of undiscovered potential, given the fact that we have chosen to prioritize robustness over speed, and that the encoding and decoding parts of the implementation

are far from optimized. This means that in the project's current state, it leaves a lot to be desired, but further development of the program should yield some better results.

So is it viable in our case? In the case of using rapid image transmission for navigation purposes and giving the operator a feed of images to help steering the drone, it is at this stage not a viable option. We sacrifice too much data for speed in the program's current form with the data rate that we have managed to achieve, and thus, the quality of images sent at a rapid rate are simply too lacking in quality to properly discern crucial details that are needed in order to use it for navigation.

With that being said, it still proves beneficial in other areas, namely data collection. As mentioned in section 1.3, the conventional way of gathering data with an AUV is done by storing the data aboard the AUV, and then retrieving the drone once the mission has been completed. In the case of stumbling upon some data that needs further analysis, the operators of the AUV would be completely unaware until the vehicle is recovered. A continuous data stream, even offline, would ensure that a research team or operator could be kept reasonably up to date with what data is close to and available to the AUV, and as many AUVs carry acoustic modems and are to a limited extent remotely controlled and susceptible to commands, this would mean that a team of researchers could focus their attention on an interesting find or data within a real-time window.

Utilizing offline transmission ends up being less than ideal for the use case that we set out to accomplish, however other benefits of our implementation, such as continuous data transmission still proves to be most beneficial.

How was the frequency spectrum determined and do we utilise our frequency spectrum efficiently?

There are two main reasons for the chosen bandwidth for this project; the first is in relation to the findings of chapter 3 which explains attenuation and that the low frequencies are going to travel the furthest. The attenuation is one of the reasons why we have chosen the bandwidth 440-6,600 Hz. Thus it can be questioned why did we not go all the way down to 20 Hz at the lowest end of the spectrum which would increase our range of transmission. That is because it would be way more difficult to sort the noise out of a transmission.

The second reason why 440 Hz was chosen as a lower bound of the frequency spectrum is because of the chosen sample rate of 44,000 Hz, which we found out has to have a frequency adding up to the sample rate else the program would not function probably with our implementation of FFmpeg. Therefore it has been chosen that the lowest valid frequency is 440 Hz and the following frequencies are multiples of 440 Hz. Theoretically,

the lower bound could also have been set to e.g. 220 Hz which is also suitable for the sample rate and therefore the program would in theory still function.

Besides the noise and sample rate, the lower bound has also been initialised to 440 Hz, arbitrarily because it is the frequency of the musical tone of A. These reasons led us to a lower bound of 440 Hz, but it was actually still an arbitrary decision, because there are likely other frequencies to choose, which are lower and live up to the same standards as the current. But this was the first that worked and it was good enough so we used our time elsewhere.

In our spectrum of valid frequencies, each valid frequency is spaced with 440 Hz. It could be discussed whether we utilize the whole spectrum well enough. Our current bandwidth is 6,160 Hz which includes 14 valid frequencies and a separator tone. Since the margin of error is $\pm$40 Hz, each valid frequency only occupies 80 Hz. So in theory it should be possible to fit 77 frequencies in the bandwidth and drastically improve the data rate, either by implementing a MIMO technique, or by improving the MFSK modulation scheme.

On the other hand to do this we would be forced to use much more equipment, and if possible add an error correction code which is a skill set that we are not particularly well versed in and additionally we are trying to send a low quality image which is possible without a more effective use of the spectrum. Therefore it would be an ineffective priority of our time to efficiently utilize our frequency spectrum.

Further development

For further continued development of the communication system, the bit rate of the receiver can be improved by using another method for recording audio that makes it possible to increase the sample rate of our Rx code. Currently, the transmitter is limited by the receiver's low sample rate, which means the bit duration is longer than the ideal length. The receiver also struggles to record the proper length of each tone, which results in the difficulty of calculating repeated tones of the same frequency. This is why a separator tone is needed, so the receiver can know when a tone ends. Ideally we would remove the separator tone to effectively double our bit rate. The current modulation scheme is quite simple, it only shifts between 14 information carrying frequencies. It is possible to increase the number of frequencies by utilizing more of the audio spectrum. That comes at a cost of range on the higher frequencies, so to preserve a good range, the space between the frequencies on the spectrum can be lowered to fit more transmitting frequencies in the same bandwidth. Multiple transmitters and receivers can be used to increase the data rate by splitting the data stream between the transmitters and receivers. This requires a complex decoding algorithm so the feature is out of the scope for this project.

Currently, the receiver will pick up any frequencies that gets recorded, but to reduce noise, the receiver only saves valid frequencies, but it is still possible for noise to peek over the signal and everything gets discarded. To prevent that, a low cut, high cut and band-reject filters can be implemented which effectively ignores the amplitude of frequencies outside of the bandwidth and between the valid frequencies, which will ignore any peeks outside the bandwidth which reduces the impact of noise received.

Additionally, we could reduce the file size by not transmitting most of the JPEG header containing metadata, DCT- and huffman tables and thumbnails. If all of the images contain the same metadata and compression tables, the headers would be almost identical and therefore there is only need for the crucial information to be transmitted. On one hand, this would save between 644 - 297 bytes per JPEG image[41], but on the other hand it would reduce the flexibility of the program, since the operator would not be able to choose a specific image resolution, because the header file for every combination of JPEG headers would have to be stored at the receiver. This could probably be solved by having a couple of presets for image quality.

Chapter 11

Conclusion

In this report, we have established our motivation; to improve the current UUV solutions, seeking to bridge the gap of wireless communication between operator and drone, and establish a continuous image stream that is fast and detailed enough to be utilized as a means of navigating a drone. We set out to develop an acoustic modulation scheme that could be used to achieve this goal, and further increase the usability of UUVs as a solution to underwater surveillance and data gathering. UUVs play a large role in the scientific community and the oil industry, and therefore we were motivated by increasing the operability of UUVs and broadening the use cases in which UUVs can be utilized for those purposes, with the potential of using UUVs for undiscovered tasks using our solution.

We have managed to make a fully functional encoding scheme, transmission system and decoder, and have succeeded in encoding, transmitting, and decoding an image wirelessly through these means. In the presented solution, the user is able to take a picture with a webcam, encode that image to a binary string, and then translate this binary string in sets of 3 bits. Each set of bits is translated to a frequency, and this then gets compiled into an audio file, that consists of several alternating tones over the duration of the signal. This signal can then be transmitted via a speaker and received with a microphone. Once the transmission is completed, the software will reconstruct the original image and display it to the user given no noise has exceeded the amplitude of the transmission. Our analysis of the problem has given us the means of understanding the need and gap in drone technology, and has allowed us to attempt to implement such a system. Although we have managed to understand transmission theory and implement a working prototype that is capable of such a transmission, it was not possible to perform transmission of a fast, continuous stream of images, with the purpose of using said image stream for drone navigation and

awareness. This is due to the design choices as described in chapter 7 being a product of learning as the development went along, and approaching the solution with a mindset of robustness and usability over speed, along with a firm belief that the solution is in its infant stage and leaves a lot of improvement on the table for future development. With that being said, we have managed to find a different use case for the program, namely data collection, and thus the program and project has still proven fruitful.

This leads to the conclusion that with the current technical knowledge of the authors of this paper along with the time constraint, we have been unable to reach our transmission goal in time, though a coding scheme for transmission of images based on acoustic signals has successfully been designed.

The finished product is in its nature usable, although further development is needed in order for it to be functionally placed and utilized aboard a UUV, and more time is needed to test the product in a real life scenario, so that unforeseen challenges and further requirements can be tackled.

Bibliography

[1] *About FFmpeg*. URL: https://www.ffmpeg.org/about.html. (Accessed on 19/05/2021).

[2] Mazin Ali A. Ali. "Characteristics of Optical Channel for an Underwater Optical Wireless Communications Based on Visible Light". In: *Australian Journal of Basic and Applied Sciences* 9.23 (2015). URL: https://www.researchgate.net/publication/280774714_Characteristics_of_Optical_Channel_for_an_Underwater_Optical_Wireless_Communications_Based_on_Visible_Light.

[3] Associated Press. *Pipeline firm gets $3.3-million fine for worst California oil spill in 25 years*. 2019. URL: https://www.latimes.com/local/lanow/la-me-ln-pipeline-spill-refugio-beach-fine-20190425-story.html. (Accessed on 22/02/2021).

[4] Abul Azad. *Binary Frequency Shift Keying - NIU - Internet Accessible Remote Laboratories*. 2021. URL: https://www.niu.edu/remote-lab/resources/graphical-user-interfaces/frequency-shift.shtml. (Accessed on 26/05/2021).

[5] F.A Azis et al. "Problem Identification for Underwater Remotely Operated Vehicle (ROV): A Case Study". In: *Procedia engineering* 41 (2012). DOI: 10.1016/j.proeng.2012.07.211.

[6] Christopher S. Baird. *Does sound travel faster in space?* 2013. URL: https://wtamu.edu/~cbaird/sq/2013/02/14/does-sound-travel-faster-in-space/#:~:text=Sound%20does%20not%20travel%20at,vibrate%20and%20therefore%20no%20sound.&text=Radio%20is%20a%20form%20of,vacuum%20of%20space%20just%20fine. (Accessed on 28/04/2021).

[7] Giuliano Benelli and Alessandro Pozzebon. *RFID Under Water: Technical Issues and Applications | IntechOpen*. 2013. URL: https://www.intechopen.com/books/radio-frequency-identification-from-system-to-applications/rfid-under-water-technical-issues-and-applications. (Accessed on 12/04/2021).

[8] Dhaval R. Bhojani. *Hybrid Video Compression Standard*. 1st ed. SpringerBriefs in Computational Intelligence. Springer Singapore, 2020. DOI: 10.1007\%2F978-981-15-0245-3.

[9] *Binary To Text Converter (Translator)*. URL: https://binarytotext.net/. (Accessed on 25/05/2021).

[10] Emil Björnson. *Basics of Antennas and Beamforming - Massive MIMO Networks*. 2018. URL: https://www.youtube.com/watch?v=xGkyZw98Tug. (Accessed 08/04/2021).

[11] D. Richard Blidberg. *The Development of Autonomous Underwater Vehicles (AUV); A Brief Summary*. 2001. URL: https://static.aminer.org/pdf/PDF/000/259/810/autonomous_underwater_vehicles_current_activities_and_research_opportunities.pdf. (Accessed on 26/05/2021).

[12] Sabine de Brabandere. *What Do You Hear Underwater?* Scientific American. 2019. URL: https://www.scientificamerican.com/article/what-do-you-hear-underwater/. (Accessed on 03/03/2021).

[13] Bulgin. *ROV's In The Oil & Gas Industry*. URL: https://blog.bulgin.com/blog/rovs-in-the-oil-gas-industry. (Accessed on 29/03/2021).

[14] Ted Burke. *A simple way to read and write audio and video files in C using FFmpeg (part 1: audio)*. 2017. URL: https://batchloaf.wordpress.com/2017/02/10/a-simple-way-to-read-and-write-audio-and-video-files-in-c-using-ffmpeg/. (Accessed on 19/05/2021).

[15] *C library function - system()*. URL: https://www.tutorialspoint.com/c_standard_library/c_function_system.htm. (Accessed on 19/05/2021).

[16] Justine Calma. *Offshore drilling has dug itself a deeper hole since Deepwater Horizon*. 2020. URL: https://www.theverge.com/2020/4/20/21228577/offshore-drilling-deepwater-horizon-10-year-anniversary. (Accessed on 29/03/2021).

[17] Filippo Campagnaro, Alberto Signori, and Michele Zorzi. "Wireless Remote Control for Underwater Vehicles". In: *Journal of Marine Science and Engineering* 8.736 (2020). DOI: 10.3390/jmse8100736.

[18] Romano Capocci et al. "Inspection-Class Remotely Operated Vehicles – A Review". In: *Journal of Marine Science and Engineering* 5.13 (2017). DOI: 10.3390/jmse5010013.

[19] Centre for Maritime Research & Experimentation. *CMRE - Autonomous Surveillance*. URL: https://www.cmre.nato.int/research/mine-countermeasures. (Accessed on 22/02/2021).

[20] Bill Chadwick. *Remotely Operated Vehicles (ROVs) and Autonomous Underwater Vehicles (AUVs)*. 2010. URL: https://oceanexplorer.noaa.gov/explorations/02fire/background/rovs_auvs/rov_auv.html. (Accessed on 22/02/2021).

[21] Yifei Chen et al. *OSA | 26 m/5.5 Gbps air-water optical wireless communication based on an OFDM-modulated 520-nm laser diode*. 2017. URL: https://www.osapublishing.org/oe/fulltext.cfm?uri=oe-25-13-14760&id=368163. (Accessed on 06/04/2021).

[22] Denise Chow. *Flight 370: Oil in Indian Ocean Not from Missing Jetliner*. 2014. URL: https://www.livescience.com/44918-flight-370-oil-slick.html. (Accessed on 22/02/2021).

[23] Leon Cohen. "The history of noise". In: *Proceedings of SPIE*. Vol. 5473. 1. SPIE, 2004. DOI: 10.1117/12.547847.

[24] *Compare Text Online*. URL: https://countwordsfree.com/comparetexts. (Accessed on 24/05/2021).

[25] *Conversion of Image, binary, image using C*. URL: https://stackoverflow.com/questions/32527351/conversion-of-image-binary-image-using-c. (Accessed on 19/05/2021).

[26] *Converting Binary to text using JavaScript*. URL: https://stackoverflow.com/questions/21354235/converting-binary-to-text-using-javascript. (Accessed on 20/05/2021).

[27] "Huffman codes". In: *Introduction to algorithms*. Ed. by Thomas Cormen et al. 2009, pp. 428–433. ISBN: 978-0-262-53305-8.

[28] Tony Drake David Bingham. *TS4_4_bingham_etal.pdf*. 2002. URL: https://www.fig.net/resources/proceedings/fig_proceedings/fig_2002/Ts4-4/TS4_4_bingham_etal.pdf. (Accessed on 17/03/2021).

[29] E.C. De Souza and N. Maruyama. "Intelligent UUVs: Some issues on ROV dynamic positioning". In: *IEEE transactions on aerospace and electronic systems* 43.1 (2007). DOI: 10.1109/TAES.2007.357128.

[30] Deep Trekker. *Customer Success Story: MOWI*. URL: https://www.deeptrekker.com/resources/customer-success-story-mowi. (Accessed on 22/02/2021).

[31] Defence R&D Canada - Toronto. *(PDF) Human Factors Issues When Operating Unmanned Underwater Vehicles*. 2011. URL: https://www.researchgate.net/publication/266067859_Human_Factors_Issues_When_Operating_Unmanned_Underwater_Vehicles. (Accessed on 22/02/2021).

[32] Develogic. *HAM.Node*. 2021. URL: http://www.develogic.de/wp-content/uploads/2012/03/Modular-Hydro-Acoustic-Modems-03-2012.pdf. (Accessed on 10/03/2021).

[33] Diffen LLC. *AM vs FM*. URL: https://www.diffen.com/difference/AM_vs_FM. (Accessed on 19/05/2021).

[34] Electronics Notes. *Amplitude Modulation, AM*. 2021. URL: https://www.electronics-notes.com/articles/radio/modulation/amplitude-modulation-am.php. (Accessed on 26/05/2021).

[35] Ming-Chung Fang, Chang-Shang Hou, and Jhih-Hong Luo. "On the motions of the underwater remotely operated vehicle with the umbilical cable effect". In: *Ocean Engineering* 34.8 (2007). DOI: 10.1016/j.oceaneng.2006.04.014.

[36] Federation of American Scientists. *Extremely Low Frequency Transmitter Site, Clam Lake, Wisconsin*. 2001. URL: https://fas.org/nuke/guide/usa/c3i/fs_clam_lake_elf2003.pdf. (Accessed on 24/03/2021).

[37] *FFmpeg Codecs Documentation*. URL: https://ffmpeg.org/ffmpeg-codecs.html. (Accessed on 19/05/2021).

[38] Reinhard Gentz and Sean Peisert. *An Examination and Survey of Random Bit Flips and Scientific Computing*. 2019. URL: http://hdl.handle.net/2022/24910. (Accessed on 24/03/2021).

[39] D. Gesbert et al. "From theory to practice: an overview of MIMO space-time coded wireless systems". In: *IEEE Journal on Selected Areas in Communications* 21.3 (2003). DOI: 10.1109/JSAC.2003.809458.

[40] Mohamad Ghaddar et al. "Mm-waves propagation measurements in underground mine using directional MIMO antennas". In: *IET microwaves, antennas and propagation* 10.5 (2016). DOI: 10.1049/iet-map.2015.0408.

[41] Calvin Hass. *JPEG Huffman Coding Tutorial*. URL: https://www.impulseadventure.com/photo/jpeg-huffman-coding.html. (Accessed on 17/05/2021).

[42] Duncan Haughey. *MoSCoW Method*. URL: https://www.projectsmart.co.uk/moscow-method.php. (Accessed on 19/05/2021).

[43] Mark Hoferitza. *Using RF Technology to Form a Wireless Network Under Water - Aerospace & Defense Technology*. 2013. URL: https://www.aerodefensetech.com/component/content/article/adt/supplements/rfm/features/articles/17526. (Accessed on 10/03/2021).

[44] Homeland Security. *Underwater Hull Scanning with Remotely Operated Vehicles*. 2009. URL: https://www.hsdl.org/?view&did=811496. (Accessed on 22/02/2021).

[45] *How can I insert a character after every n characters in javascript?* URL: https://stackoverflow.com/questions/1772941/how-can-i-insert-a-character-after-every-n-characters-in-javascript. (Accessed on 20/05/2021).

[46] Marecomms Inc. *ROAM (Roboust Acoustic Modem)*. URL: http://marecomms.ca/index.html. (Accessed on 10/03/2021).

[47] Å. Irgens et al. *Mortality among professional divers in Norway*. 2013. DOI: https://doi.org/10.1093/occmed/kqt112.

[48] IUPAC. *Compendium of Chemical Terminology: The Beer-Lambert Law*. 1997. DOI: 10.1351/goldbook.B00626.

[49] S.K. Jayaweera and H.V. Poor. "Capacity of multiple-antenna systems with both receiver and transmitter channel state information". In: *IEEE Transactions on Information Theory* 49.10 (2003). DOI: 10.1109/TIT.2003.817479.

[50] Christopher G. Jennings. *How JPEG works*. 2017. URL: https://cgjennings.ca/articles/jpeg-compression. (Accessed on 23/05/2021).

[51] Robert Johnson. *The OceanWorks International Submarine Rescue Drone*. 2012. URL: https://www.businessinsider.com/the-srdrs-submarines-rescue-drone-2012-10?r=US&IR=T. (Accessed on 22/02/2021).

[52] D. Karimanzira et al. "First testing of an AUV mission planning and guidance system for water quality monitoring and fish behavior observation in net cage fish farming". In: *Information Processing in Agriculture* 1.2 (2014). DOI: 10.1016/j.inpa.2014.12.001.

[53] Hemani Kaushal and Georges Kaddoum. *Underwater Optical Wireless Communication*. 2016. DOI: 10.1109/ACCESS.2016.2552538.

[54] Kawasaki. *Kawasaki has successfully verified the Close-range Subsea Pipeline Inspection by Autonomous Underwater Vehicle (AUV)*. 2020. URL: https://global.kawasaki.com/en/corp/newsroom/news/detail/?f=20200715_8265. (Accessed on 22/02/2021).

[55] Sergii Kortnieiev, Volodymyr Shuliak, and Kirill Otradnov. *Underwater Wireless Video Communication*. 2016. URL: https://www.hydro-international.com/content/article/underwater-wireless-video-communication. (Accessed on 22/02/2021).

[56] Marco Lanzagorta. *Synthesis lectures on communications: Underwater communications*. 2013. DOI: 10.2200/S00409ED1V01Y201203COM006.

[57] M. D. Macleod. "14 - Coding". In: *Telecommunications Engineer's Reference Book*. Ed. by Fraidoon Mazda. Butterworth-Heinemann, 1993. DOI: 10.1016/B978-0-7506-1162-6.50020-4.

[58] Thomas L Marzetta. "Massive MIMO: An Introduction". In: *Bell Labs technical journal* 20 (2015). DOI: 10.15325/BLTJ.2015.2407793.

[59] National Geographic. *Ocean | National Geographic Society*. URL: https://www.nationalgeographic.org/encyclopedia/ocean/. (Accessed on 22/02/2021).

[60] National Oceanic and Atmospheric Administration. *Understanding Ocean Acoustics*. URL: https://oceanexplorer.noaa.gov/explorations/sound01/background/acoustics/acoustics.html. (Accessed on 10/03/2021).

[61] National Physical Laboratory. *Mechanisms of absorption*. URL: http://resource.npl.co.uk/acoustics/techguides/seaabsorption/physics.html. (Accessed on 29/03/2021).

[62] Nipun. *Difference Between Radio Waves and Sound Waves*. Pediaa.Com. Section: Physics. 2015. URL: https://pediaa.com/difference-between-radio-waves-and-sound-waves/. (Accessed on 03/03/2021).

[63] NOAA Ocean Exploration. *What is an AUV?* URL: https://oceanexplorer.noaa.gov/facts/auv.html. (Accessed on 24/02/2021).

[64] Ocean Exploration and Research. *Introduction to Remotely Operated Vehicles and Autonomous Underwater Vehicles*. URL: https://oceanexplorer.noaa.gov/okeanos/edu/collection/media/hdwe-URintro.pdf. (Accessed on 24/03/2021).

[65] Ocean Exploration and Research. *What is an ROV?* URL: https://oceanexplorer.noaa.gov/facts/rov.html. (Accessed on 23/03/2021).

[66] *p5.js*. URL: https://p5js.org/. (Accessed on 26/05/2021).

[67] *p5.js reference*. URL: https://p5js.org/reference/#/p5.FFT. (Accessed on 19/05/2021).

[68] Fabio Palmigiani. *Petrobras to tender for underwater gear to map offshore developments*. 2020. URL: https://www.upstreamonline.com/exploration/petrobras-to-tender-for-underwater-gear-to-map-offshore-developments/2-1-897870. (Accessed on 19/03/2021).

[69] George Pararas-Carayannis. "Turtle: A Revolutionary Submarine". In: *Sea Frontiers* 22.4 (1976). (Accessed on 22/02/2021).

[70] PC Plus. *All you need to know about JPEG compression*. 2009. URL: https://www.techradar.com/news/computing/all-you-need-to-know-about-jpeg-compression-586268/2. (Accessed on 23/05/2021).

[71] M. Rahmati, A. Gurney, and D. Pompili. "Adaptive Underwater Video Transmission via Software-Defined MIMO Acoustic Modems". In: *OCEANS 2018 MTS/IEEE Charleston*. 2018. DOI: 10.1109/OCEANS.2018.8604782.

[72] John Rander. *Understanding color loss underwater*. URL: http://johnrander.chez-alice.fr/absorption_en.html. (Accessed on 03/03/2021).

[73] Lynnette Reese. *What is signal attenuation?* 2020. URL: https://www.analogictips.com/what-is-signal-attenuation/. (Accessed on 24/02/2021).

[74] Sean Riley. *Error Detection and Flipping the Bits - Computerphile*. 2013. URL: https://www.youtube.com/watch?v=-15nx57tbfc&ab_channel=Computerphile. (Accessed on 24/03/2021).

[75] SAAB SEAEYE. *Falcon – The professional portable underwater vehicle*. URL: https://www.saabseaeye.com/solutions/underwater-vehicles/falcon. (Accessed on 29/03/2021).

[76] A Said and W.A Pearlman. "An image multiresolution representation for lossless and lossy compression". In: *IEEE transactions on image processing* 5.9 (1996). DOI: 10.1109/83.535842.

[77] David Salomon. *Data Compression : The Complete Reference*. Fourth Edition. Springer-Verlag London Limited, 2007. DOI: 10.1007%2F978-1-84628-603-2.

[78] Schmidt Ocean Institute. *Ship specifications, R/V Falkor*. URL: https://schmidtocean.org/rv-falkor/ship-specifications/.

[79] SeaTechnology. *A Brief History of ROVs Sea Technology magazine*. 2019. URL: https://sea-technology.com/a-brief-history-of-rovs. (Accessed on 22/02/2021).

[80] Hugerles S. Silva et al. "Bit error probability of the M-QAM scheme subject to multilevel (double) gated additive white Gaussian noise and Nakagami-m fading". In: *Wireless Networks* 25.7 (2019). DOI: 10.1007/s11276-019-02100-9.

[81] Aarti Singh. *Information Processing and Learning, Lecture 16: Shannon's Noisy Coding Theorem*. 2012. URL: http://www.cs.cmu.edu/~aarti/Class/10704/lec16-shannonnoisythrm.pdf. (Accessed on 27/04/2021).

[82] Sonardyne. *BLUECOMM 200 – WIRELESS UNDERWATER VIDEO AND VEHICLE CONTROL*. URL: https://www.sonardyne.com/product/bluecomm-200-wireless-underwater-video/. (Accessed on 09/04/2021).

[83] Sonardyne. *WORLD'S FIRST LIVE VIDEO BROADCAST FROM UNDERWATER USES SONARDYNE'S BLUECOMM*. URL: https://www.sonardyne.com/worlds-first-live-video-broadcast-from-underwater-uses-sonardynes-bluecomm/. (Accessed on 09/04/2021).

[84] A.V. Straube et al. "Rapid onset of molecular friction in liquids bridging between the atomistic and hydrodynamic pictures". In: *Communications Physics* 3.126 (2020). DOI: 10.1038/s42005-020-0389-0. (Accessed on 28/04/2021).

[85] Techopedia. *What is Multiple Frequency-Shift Keying (MFSK)? - Definition from Techopedia*. 2014. URL: https://www.techopedia.com/definition/14828/multiple-frequency-shift-keying-mfsk. (Accessed on 26/05/2021).

[86] The University of Arizona. *Scattering of Light*. 2013. URL: https://wtamu.edu/~cbaird/sq/2013/02/14/does-sound-travel-faster-in-space/#:~:text=Sound%20does%20not%20travel%20at,vibrate%20and%20therefore%20no%20sound.&text=Radio%20is%20a%20form%20of,vacuum%20of%20space%20just%20fine. (Accessed on 28/04/2021).

[87] TheEngineeringToolbox.com. *Understanding color loss underwater*. URL: https://www.engineeringtoolbox.com/air-density-specific-weight-d_600.html. (Accessed on 03/03/2021).

[88] David Tse and Pramod Viswanath. "MIMO I: spatial multiplexing and channel modeling". In: *Fundamentals of Wireless Communication*. Cambridge University Press, 2005. DOI: 10.1017/CBO9780511807213.008.

[89] tutorialspoint. *Antenna Theory - Half-Wave Dipole*. URL: https://www.tutorialspoint.com/antenna_theory/antenna_theory_half_wave_dipole.htm. (Accessed on 22/03/2021).

[90] University of Hawaii: Department of Physics. *RECEIVER SENSITIVITY / NOISE*. URL: https://www.phys.hawaii.edu/~anita/new/papers/militaryHandbook/rcvr_sen.pdf. (Accessed on 28/04/2021).

[91] UTMconsultants. *A Day in the life of an ROV Pilot*. 2020. URL: https://www.utmconsultants.com/news/a-day-in-the-life-of-an-rov-pilot/41104/. (Accessed on 22/02/2021).

[92] M.C. Valenti and M. Fanaei. "Chapter 5 - The interplay between modulation and channel coding". In: *Transmission Techniques for Digital Communications*. Ed. by Ezio Biglieri, Katie Wilson, and Stephen G. Wilson. Elsevier Science & Technology, 2016. DOI: 10.1016/B978-0-12-398281-0.00005-3.

[93] *Very low frequency*. 2021. URL: https://en.wikipedia.org/wiki/Very_low_frequency#VLF_transmitting_antennas. (Accessed on 24/02/2021).

[94] Virginia Institute of Marine Science. *What is an AUV?* URL: https://www.vims.edu/research/units/legacy/cornwallis/auv/index.php. (Accessed on 24/03/2021).

[95] Virginia Institute of Marine Science. *What is an ROV?* URL: https://www.vims.edu/research/units/legacy/cornwallis/rov/index.php. (Accessed on 24/03/2021).

[96] *What is Base64?* URL: https://base64.guru/learn/what-is-base64. (Accessed on 19/05/2021).

[97] Edith Widder. *Deep Light*. URL: https://oceanexplorer.noaa.gov/explorations/04deepscope/background/deeplight/deeplight.html. (Accessed on 11/03/2021).

[98] Wikipedia. *Radix*. 2021. URL: https://en.wikipedia.org/wiki/Radix. (Accessed on 27/05/2021).

[99] *WindowOrWorkerGlobalScope.btoa*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/btoa. (Accessed on 20/05/2021).

[100] Yingjun Zhou et al. *Four-Channel WDM 640 Gb/s 256 QAM Transmission Utilizing Kramers-Kronig Receiver | IEEE Journals & Magazine | IEEE Xplore*. 2019. DOI: 10.1109/JLT.2019.2943122.