Uppsala University

COURSE NAME

COURSE CODE

PROJECT NAME

Authors:
Axel Östfeldt

February 1, 2024

Abstract

Some test text for abstract

Acknowledgement

Some test text for Acknowledgement

Table of contents

1 Nomenclature

1.1 Terminology

1.2 Abbreviations and acronyms

1.3 Mathematical notation

2 Introduction

In 1953 Dr Warren Weaver summarised the problem of communication in three levels:

- Level A. How accurate are the transmission of symbols? (The technical problem)
- Level B. How well do the symbols transmitted convey the desired meaning (The semantic problem)
- Level C. How effectively does the received meaning affect conduct in the desired way? (The effectiveness problem)

This paper will look at a communication problem from a Level A perspective. Since Weaver wrote his paper over 70 years ago the amount of data communicated all over the world have increased many times over. The question Dr Weaver stated to formulate a Level A problem can be considered more relevant than ever[1].

(a) How does one measure amount of information?

(b) How does one measure the capacity of a communication channel?

(c) What are the characteristics of an efficient coding process?

(d) With efficient coding, at what rate can the communication channel convey information?

(e) How does the signal being transmitted being continuous, such as audio signal, affect the problem?

Although the question asked in this paper uses different semantics (a Level B problem as Weaver would say) they are all relevant through out this work. This paper looks at the specific problem of compressing audio data with the aim to decrease bit rate over a communication channel. To be able to do this relevant sources regarding information theory and compression algorithm will be composed. When summarising the theory of compression algorithms it is divided between the model and the coder. The model captures a probability of the symbols being sent, while the coder takes advantage's of these probabilities to generate codes[2]. The information gathered will then be applied with Python and *FPGA* code in order to solve the problem of compressing audio data in a real time environment.

2.1 Background

Saab AB have during the last couple of summers develop a system to localise sound sources under the alias *Aucoustic Warfare*. The system uses beamforming in order to visualize the audio sources as a heat map over a webcam feed in real-time.

The system uses four PCBs containing 8x8 microphone arrays to record the audio. Each PCB consists of 64 ICS-52000 microphones with an ADC, anti-aliasing filter, a wide-band response and omni-directional directionality. The microphones on each PCB are daisy chained together in groups of 16, creating four groups on each card.

An FPGA on a Zybo-Z7 20 development board sampels the audio using the microphone arrays. The samplin starts with an incoming WS-pulse and are then sampled with a frequency of 48828,125 Hz. The microphones outputs 24-bits of data, these are padded with with two's compliment and concatenated in order to create 32-bits of data. The data is sent to a 32-bit Time-Division Multiplexing slot.

An AXI protocol is used to load the data packages in to the pacakge sender. The package sender on the FPGA board then uses User Dtagram Protocol, UDP, to send the packages to a package collector over a ethernet connection. The data is sent in packages containing several 32-bit slots. The first two slots in each package contains a header with meta-data. 8-bit for protocol version, followed by 8-bit for number of arrays, then 16-bit for frequency information, and lastly 32-bit for sample counter. The following 256 slots contains 32-bit data from each microphone.

Once received on the PC side the samples are stored in a buffer. This buffer is of size 256 by 256, meaning 256 samples from each of the 256 microphones. It is continuously updated by the information sent over the ethernet cable and old samples are over written. To perform the beam forming the samples are extracted from the buffer. The samples are then processed and a heat map is generated using different beam forming algorithms.[3]

2.2 Purpose

The purpose of this paper is to gather in depth knowledge around the subject data compression by finding, studying and compiling relevant literature. The in depth knowledge will be applied in a scientific and engineering manner solve the technical problem of data transfer rate in Saabs *Acoustic Warfare* system. By implementing an compression algorithm for the FPGA-board in order to reduce the average transfer

rate. This could allow for cheaper FPGA-boards to be used, which usually are limited to 100 Mbps transfer rate using ethernet.

2.3 Limitations

Compression algorithms are plenty full but there is no algorithm that can compress all types of data perfectly. The data being handled in this project is strictly audio data. Therefore the algorithms are limited to only look in to audio compression algorithms.

Compression algorithm are also usually divided into lossy- and lossless- algorithms. Lossy algorithms allows for some loss of data without corrupting the signal, while lossless algorithms compresses all data and decompresses it exactly to the original data (in theory). In audio compression lossy algorithms are quite common, since they are usually used in the context of humans listening to audio in some format. Losing data in this context can be fine if it does not affect how it sounds to human ears, this could be losing frequencies outside of the ears range to save data for example. In this project the compression will be applied to a system that uses audio data to perform beam forming and localise the sound source, and not for listening to audio. Lossy algorithms have therefore been excluded and the project have been limited to looking at lossless audio compression algorithms.

Three different algorithms that matches the limitation set will be looked at in this paper, that are: Shorten, LPC, and FLAC.

2.4 Goal

The goal is to bring forth a compression algorithm for the FPGA-board to be used in Saabs Audio Warfare system that brings the average transfer rate below 90 Mbps and preferably down to 45 Mbps.

3 Methodology

3.1 Research and information gathering

The research for this project was centered around two points of interest, the current system and compression algorithms. The current system was examined by looking through the report from the project done in *Acoustic Warfare*[3].

When looking at compression algorithms it was focused around which ones was most commonly used for FPGA and for audio data. For FPGA GZIP was found to be the most common, a modified version for LZ77, how ever when looking at audio compression it was stated that GZIP was not well suited for the task. For the algorithms that was found most interesting around compressing audio in a lossless manner no research on how to implement them on FPGA could be found. This leaves an interesting scientific vacum for this paper to explore.

3.2 Python programming

After researching compression algorithms the most well suited for the task was picked to be implement with python. An audio file was recorded with system created in *Acoustic Warfare* with saved timestamps. This could then be used to simulate how the different algorithm would perform in a real time environment. This helped find which of the algorithms had the most promise and should be implemented in the *Acoustic Warfare* system. Although the compression ratio was central for determining which compression algorithm was best this was weighed against the computing power needed. Settling for a compression ratio that is not the best but good enough might be the best choice if it saves computing power.

3.3 Implementation on system

The best performing compression algorithm was focused on when implementing it on the Acoustic Warefare system. The performance in the system was tested out and evaluated.

4 Theory

4.1 Information Theory

Information theory aims to measure how much information is contained in messages sent during communication. The simplest case of measuring information is by taking the twos logarithm of the number of available choices. For example if a communication channel have 16 alternative messages and each are as likely to be sent the information is calculated with $\log_2 16 = 4$. The unit for measuring information is "bit", the previous example have 4 bits of information[1].

With compression algorithms the aim is to reduce the amount of bits sent over the communication channel. But it is not possible for a lossless compression algorithm to be able to compress all messages, at least in the case where the message sent can contain any bit-sequence. For example, if a communication channel sends messages of 100 bits it is possible to send 2^{100} different messages. If these messages where to be compressed by just one bit, so all messages contain 99 bits, the communication channel would only be able to send 2^{99} different messages. This shows that in order to compress some messages in the communication channel, other messages have to be lengthened. In fact for a set of input messages with fixed length, if one message is compressed the average length of all possible inputs will always get longer than the original inputs. Since it is not possible to achieve compression of all messages compression algorithm takes advantage of the messages probabilities. Compression algorithms take advantage of the bias for different messages by giving shorter codes to messages with high bias and longer to those with low bias[2]. An example of this can bee seen when looking at telegraphy, where common letters, such as *E*, have shorter messages and more unlikely letters, such as *Q*, have longer messages[4].

4.1.1 Entropy

The concept of entropy is a measurement of information. If there are n different possible messages that can be sent on the communication channel, each with the possibility p_i, the entropy is calculated with the equation:

$$H = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{1}$$

Where H is the entropy in bits per message. H = 0 can only be achieved if there is only one possible choice, that is all p_i are zero but one. In all other cases H is positive. If all

the probabilities of p_i are equal $\mathrm{H} = \log_2 n$. This is the maximum value H can achieve for a given set of n possible messages. This shows how entropy relates to uncertainty. When there is only one possible choice the uncertainty vanishes and entropy is zero. When all choices are equally possible the uncertainty is at maximum, and so is the entropy. As the possibilities get closer to each other the entropy increases[4]. This aligns with lower entropy indicating more biased probabilities in the messages.

The concept of self information represents how many bits of information a message contains and is a good indicator of how many bits the message should be encoded in. Self information of a message can be calculates with the equation:

$$i(s) = \log_2 \frac{1}{p(s)} \tag{2}$$

Where i(s) is the self information of the message and and p(s) is the probability of the message. A messages information is inversely related to its probability, meaning messages with less information have higher probability and vice versa. Entropy is the probability weighted average of the self information for each messages, with larger entropys representing a larger average of information per message. Since higher entropy indicates less biased probabilities a higher average of information information indicates a more random set of messages[2].

A noiseless communication channel with the capacity C bits per second used to transfer messages with entropy H bits information per message can never have an average transfer of information rate higher than $\frac{C}{H}$ bits per second[4]. The greater the uncertainty of the source the more information is being transmitted. It follows that higher uncertainty of the source will need more bits to represent all messages, which will affect the information rate for a given communication channel[5].

4.2 Coding

4.2.1 Run length encoding (RLE)

Run length encoding, RLE, utilises that the same data item is repeated several times consecutively in the data. When a run of symbol d occours n times in a row it can be written as:

@nd

Where @ is an *escpae character*, indicating to the decoder that an RLE is coming, n is how many times the data item is repeated, and d is the data item to repeat. A

given run of data item can then be replaced with thees three items, the compression factor from this can be calculated with the following equation:

$$\frac{N}{N - M(L-3)} \tag{3}$$

In equation 3 M is the number of repetitions in the data, L is the average length of repetition and N is a data string[6].

4.2.2 Golomb codes

Golomb codes are variable-lenghts codes that assumes that larger integers have a lower probability of occurring[7]. Golomb code is ideal to use when the list of possible outcomes are infinite but follow a distribution law. In order to use Golomb code to encode and decode a none-negative integer n a choice for the parameter m have to be made (m should preferably be set to an integer). The ideal choice of m can be derived from the following equation:

$$m = -\frac{\log 2}{\log p} \tag{4}$$

Where p is the probability of a "0" occurring in a bit stream. In cases where the ideal m is not an integer the ideal dictionary will switch to and from different m values. For large m the penalty for picking the nearest integer value is very small[8].

With n and m it is possible to calculate the three quantities needed for Golomb coding with the following equations:[6](Page 1011)

$$q = \lfloor \frac{n}{m} \rfloor \tag{5}$$

$$r = n\%m \tag{6}$$

$$c = \lceil \log_2 m \rceil \tag{7}$$

Where q is the *quotient* and r is the *remainder*. The floor sign in equation 5 indicates that q is rounded down to the closest integer, and ceiling sign in equation 7 indicates that c is rounded up to the closest interger. The *quotient* is coded as unary prefix,

meaning that it is q 0's followed by a 1 (alternatively q 1's followed by a 0). The *remainder* is then binary coded, but the length of the code depends on the value of r in the following way:

$$\text{If } r < 2^c - m \text{ then } b = c - 1$$

$$\text{If } r \geq 2^c - m \text{ then } b = c$$

Where b is the number of bits r is coded in. The first $2^c - m$ values of r, that is when $b = c$ - 1, are encoded in bits as normal. The rest, that is when $b = c$, are encoded so that the biggest possible residual consists of only 1's (the rest are written in bits decremented from this value, that is if b = c = 3 the largest residual is written as "111" and the the second largest as "110" and so on until $r < 2^c - m$). An exception to this is when m is a power of 2, then r is always written in bits as normal with the length c. The *remainder* in bits are then appended to the unary coded *quotient* to get the final code. In the table bellow are some examples of Golomb codes:

m/n	0	1	2	3	4
2	0|0	0|1	10|0	10|1	110|0
3	0|0	0|10	0|11	10|0	10|10
4	0|00	0|01	0|10	0|11	10|00
5	0|00	0|01	0|10	0|110	0|111

Table 1: Some Golomb codes, on left side of | is the quotient and on the right the remainder

It can be seen in table 1 for m = 5 that even though the r=3 and r=4 for n=3 and n=4 respectively that the binary value does not match the remainder value. This is because of how the binary value is set for r when r $< 2^c - m$ as stated earlier. If the data being handled by the Golomb code is signed there is two extra step to encoding. The sign bit for n is removed saved as s. The encoding is done the same way as stated earlier, once this is done the sign bit is added back on to the encoded message as the MSB.

In order for the decoder to correctly decode the Golomb code it needs to know two things, the value of m and if the code is signed or unsigned. With m known the decoder can calculate the value of c with equation 7. Assuming the unsigned case for Golomb codes first, the following equations are used for decoding:

$$n = m * A + R \tag{8}$$

$$n = m * A + R' - (2^c - m) \tag{9}$$

The A value is always derived the same way, by counting how many 1's occur before the first 0 (or vice versa depending on how the unary code is set up). If m is a power of 2 the c last bits are decoded to their binary value and denoted R. To get n equation number 8 is used.

In cases where m is not a power of 2 the c-1 bits following the first 0 is denoted as R. R is then compared in the same way as r:

$$\text{If } R < 2^c - m \text{ then } B = A + c$$

$$\text{If } R \geq 2^c - m \text{ then } B = A + c + 1$$

Where B is the total length of the code word. If B = A + c then R captures all the remaining bits in the code word. R can then be converted to integer and equation number 8 can be used to calculate n. If B = A + c + 1 the c bits following the first 0 is denoted as *R'*. *R'* is converted to an integer and equation number9 is used to calculate n.[6](Page 162 - 163).

For the signed decoding case in Golomb code some extra steps are needed. The MSB of the Golomb code will be the signed bit, this is first removed and saved as S. The rest of the bits are decoded as previously stated. Equation number 8 and 9 will always give a positive n value, if the sign bit, S, is a "1" an extra step is added where n = - n.[6](Page 994-995).

4.2.3 Rice codes

Golomb codes are simplified in cases where m is a power of 2[8], these cases are caled Rice codes (or Golomb-Rice codes). Instead of choosing the parameter m the parameter k is chosen in Rice codes. The relation between m and k is as follow:

$$m = 2^k$$

This simplifies the encoding, to code the binary value n in Rice code the following steps are taken:

1. The sign bit is separated, denoted s, and will later be used as the MSB for the encoded n. This step is only done if the data contains signed values.
2. The k number of LSBs are separated, denoted r, and will later be the LSB for the enconded n.

3. The remaining bits are converted to an integer and unary coded, denoted q.

The final code word will be as follow:

$$"s" + "q" + "r"$$

For decoding the following steps are taken:

1. Save the MSB as sign bit, if the data is signed. This is denoted S.
2. Count the following 0's after the sign bit until the first 1 occurs. Convert the number of 0's to binary, denoted Q.
3. Denote the last k bits that remain as R

The binary value n will be gain from:

$$"S" + "Q" + "R"$$

Rice coding allows for n to be encoded and decoded only using logical operations, which speeds up the computing for encoding and decoding. Especially the decoding is very computationally fast since it only requires one loop trough the code word to get the original n.

Rice codes are ideal when the data to be encoded follows a Laplace distribution. The value of k decides the amount of LSB to be included in the Rice code, and are linearly related to the variance of the data to be encoded. This leads to a formula to compute the ideal k value for a dataset x that follows:

$$k = \log_2(\log(2) \cdot \mathbb{E}[x]) \tag{10}$$

Where $\mathbb{E}[x]$ is the expected value over the data $x[6]$ (Sida 166-169).

4.3 Modeling

4.3.1 Shorten

Shorten is a lossless compression model that was created by Tony Robinson specifically to compress audio by exploiting that audio data often have a significant sample to sample correlation. Shorten utilises this correlation by predicting the waveform using a linear combination of previous samples. This can be written with the following equation:

$$\hat{s}(t) = \sum_{i=1}^{p} a_i \cdot s(t-i) \tag{11}$$

Where *ŝ(t)* is the predicted value of the waveform at time t, a_i is a set of coefficients, *s(t)* is the waveform value at time t, and p is the how far back in the wave form samples are used to predict the current sample (The typical value for p is 3, it never goes larger than this in shorten).

With the predicted value ŝ(t) a residual can be calculated using the equation:

$$e(t) = s(t) - \hat{s}(t) \tag{12}$$

Where e(t) is the residual, that is the difference between the predicted value and actual value of the wave form.

The ideal value of a_i and p may vary through out the wave form. Robinson suggest to divide the data into blocks and find the ideal value for a_i and p in each block. The block size suggested by Robinson is 256 values.[9] If the file have more than one audio channel each channel shorten divides each channel into separate blocks[6] (sida 992).

For calculating the prediction Shorten uses one of the following equations with different levels of p:[9].

$$\hat{s}(t) = 0 \tag{13}$$

$$\hat{s}(t) = s(t-1) \tag{14}$$

$$\hat{s}(t) = 2s(t-1) - s(t-2) \tag{15}$$

$$\hat{s}(t) = 3s(t-1) - 3s(t-2) + s(t-3) \tag{16}$$

The choice between equations 13, 14, 15, or 16 is dependent on which gives the best residual versus the computational time for the current block. The residuals are variable length coded. Correlation between audio samples implies that the residuals

are small and experiment shows that they follow the Laplace distribution. This suites the residuals well to be to be encoded with Rice codes[6] (sida 994-995).

4.3.2 Linear Predictive Coding LPC

Linear Predictive Coding utilizes correlation between audio samples to create a linear predictor of future wave form values from preceding values, similarly to Shorten, with equation 11. The difference between LPC and Shorten is how it chooses its coefficients, a_i[6]. LPC does not have set coefficients for different orders but tries to find the coefficients that minimizes the mean-squared error [10]. This can be done in three steps:

- Finding the squared error:

  $d(t) = (s(t) - \sum_{i=1}^{n} a_i \cdot s(t-i))^2$

- Differentiate the squared error with respect to coefficient a_p and set derivative to zero for p = 1, 2,..., n:

  $\frac{\partial d(t)}{\partial a_p} = 0$ for p = 1, 2,..., n

  $\Rightarrow -2 \cdot [(s(t) - \sum_{i=1}^{n} a_i \cdot s(t-i)) \cdot s(i-p)] = 0$ for p = 1, 2,..., n

- Take the expected value

  $\Rightarrow \sum_{i=1}^{n} a_i \cdot \mathbb{E}[s(t-i) \cdot s(t-p)] = \mathbb{E}[s(t) \cdot s(t-p)]$ for p = 1, 2,..., n

To simplify the expression the autocorrelation of the wave form can be used that is the correlation between two samples. The equation for the autocorrelation is as follows:

$$R_s(i) = \mathbb{E}[s(t) \cdot s(t+i)] \tag{17}$$

Applying equation 17 to the earlier expression it can be written as a matrix multiplication.

$$\begin{bmatrix} R(0) & R(1) & R(2) & ... & R(n-1) \\ R(1) & R(0) & R(1) & ... & R(n-2) \\ R(2) & R(1) & R(0) & ... & R(n-3) \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ R(n-1) & R(n-2) & R(n-3) & ... & R(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ . \\ . \\ . \\ a_n \end{bmatrix} = \begin{bmatrix} R(1) \\ R(2) \\ R(3) \\ . \\ . \\ . \\ R(4) \end{bmatrix}$$

This can be written as $\mathbf{R}$C = P. Normaly solving the equation would require O(n^3) but the matching diagonals of $\mathbf{R}$ and that it is symetrical allows for faster computations using the Levinson-Durbin algorithm[6].

The Levinson-Durbin is a recursive algorithm that finds the ith-order optimum predictor by looking at the optimum predictor of order i-1. The algorithm finds all optimal coefficients for order 0 to n along with the corresponding mean-squared prediction error E. The algorithm are computed with the following steps:

$E^0 = R(0)$

for i = 1, 2,..., n

$k_i = \frac{R(i) - \sum_{j=1}^{i-1} a_j^{i-1} \cdot R(i-j)}{E^{i-1}}$

$a_i^i = k_i$

if i > 1 then for j = 1, 2,..., i-1

$a_j^i = a_j^{i-1} - k_i \cdot a_{i-j}^{i-1}$

end

$E^i = (1 - k_i^2) \cdot E^{i-1}$

end

E^i indicates that it is the mean-squared prediction error for the coefficients of order i, a_j^i indicates that it is coefficient number j for order i. That is if a third order predictor would have the following coefficients:

$$\hat{s}(t) = 5 \cdot s(t-1) + 6 \cdot s(t-2) + 7 \cdot s(t-3)$$

The mean squared prediction error from this particular ŝ(t) would be E^3 and the coefficients would be: $a_1^3 = 5$, $a_2^3 = 6$, and $a_3^3 = 7$ [10].

4.3.3 FLAC

FLAC stands for *free lossless audio codec* and is also a compression algorithm specialised for audio data. It shares a lot of commonalities with the previously mentioned Shorten and LPC, it also exploits the fact that audio data have high correlation between each sample. The steps FLAC takes to compress the code is yet another simularity it have to other audio compression algorithms, it follows the 4 steps:

1. **Blocking.** The data is divided up into continuous block. These blocks may vary in size and span several channels.
2. **Interchannel Decorrelation.** If the audio is stereo, that is it is using two channels, the algorithm will create a mid and side signals from the average and difference of the left and right channels. Based on which of these gives the best result, left and right vs mid and side, it chooses which one to encode. This can vary from block to block.
3. **Prediction.** FLAC uses one of four predictors on each data block and calculates the residual. Which predictor is chosen is based on which one gives the smalest residual.
4. **Residual coding.** The residual is passed through an encoder for the final compression step.

Choosing a good block size is essential for the compression ratio. With every block a frame header is sent containing meta-data to the decoder in order for it to decode the data correctly. If the block size is to small a lot of space is wasted on the frame header. On the other hand, if the block size is to large the data can have large variation in its characteristics and therefore make it hard for the encoder to find a good predictor. FLAC imposes a block size of 16 to 65535 samples in order to be able to compress the data optimally.

FLAC calculates the residual the same way as Shorten, with equation 12. There are four different methods FLAC can use to model its predictor, $\hat{s}(t)$:

1. **Verbatim**
2. **Constant**
3. **Fixed linear predictor**
4. **FIR Linear prediction**

Verbatim is essentially equation 13. Since this equation always predicts a zero the residual will be the raw signal sent verbatim. This is the baseline which all the other predictors are measured against when FLAC decides which method to use.

Constant, as suggested by its name, is used when the data signal is a constant value. For this case no predictor is used and the code is instead encoded using RLE.[11]

Fixed linear predictor is the same predictors used in Shorten equation 14, 15, and 16. FLAC also implements a forth order predictor with the following equation[6](sida

1005):

$$\hat{s}(t) = 4s(t-1) - 6s(t-2) + 4s(t-3) - s(t-4) \tag{18}$$

FIR linear prediction implements the same steps as LPC in order to calculate coefficients to get the most accurate predictor, at the cost of slower encoding. FLAC supports FIR linear predictor up to the 32nd order. Just as LPC, FLAC utilises the Levinson-Durbin algorithm to calculate the coefficients for the FIR predictor from the autocorrelation in the data.

FLAC can vary which method to use in each subblock, meaning different channels in the same block can use different methods to calculate the residual. The flexibility of FLAC improves its capability of compressing the data but also means that more calculations have to be done to find the best option and more meta-data have to be sent to the decoder side for it to know which method is used.

As stated earlier for the prediction method *constant* RLE is used to encode the data. In the other cases Rice-codes are used. The value for k can be chosen in two different ways in FLAC. The first is to base it on the variance of the entire residual and use it to encode the entire residual. The second is to divide the residual into equal-length regions of contiguous samples and calculate each regions mean. The mean for each region will be used as the k value for that specific region[11].

5 Results

5.1 Python implementations

5.2 System implementations

6 Analysis and discussion

7 References

[1] *RECENT CONTRIBUTIONS TO THE MATHEMATICAL THEORY OF COMMUNICATION, Author: Warren Weaver, Published in: ETC: A Review of General Semantics, Vol. 10. No. 4, SPECIAL ISSUE ON INFOMRATION THEORY (SUMMER 1953), Published by: Institute of General Semantics, Year of Publication: 1953* Available at: `https://www.jstor.org/stable/42581364?seq=1&cid=pdf-reference#references_tab_contents`, Accessed: January 31, 2024.

[2] *Introduction to Data Compression, Author: Guy E. Blelloch, Date of Publication: 31 January 2013* Available at: `https://www.cs.cmu.edu/~15750/notes/compression_guy_blelloch_book_chapter_draft.pdf`, Accessed: February 1, 2024.

[3] *Acoutstic Warfare, Author: Marcus Allen, Tuva Björnberg, Isac Bruce, Ivar Nilsson, Mika Söderström, Jonas Teglund, Date of Publication: July 2023* Available at: `https://github.com/acoustic-warfare/Beamforming/blob/main/Ljudkriget.pdf`, Accessed: January 29, 2024.

[4] *A Mathematical Theory of Communication, Author: C. E. Shannon, Published in: The Bell System Technical Journal, Vol. 27, Published by: Nokia Bell Labs, Year of Publication: 1948* Available at: `https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf`, Accessed: January 31, 2024.

[5] *Information Security Science, Author: Carl S. Young, Published by: Elsecier Inc, Year of Publication: 2016* Available at: `https://www-sciencedirect-com.ezproxy.its.uu.se/book/9780128096437/information-security-science#book-description`, Accessed: February 1, 2024.

[6] *Handbook of Data Compression Fifth Edition, Author: David Salomon and Giovanni Motta with contributions by David Bryant, Published by: Springer London, Date of Publication: Januray 18 2010* Ebok available at: `https://link.springer.com/book/10.1007/978-1-84882-903-9`, Accessed: January 26, 2024.

[7] *Introduction to Data Compression, Author: Khalid Sayood, Published by: Elsevier Inc, Date of Publication: 2012* Ebok available at: `https://www.sciencedirect.com/book/9780124157965/`

introduction-to-data-compression#book-description, Accessed: January 26, 2024.

[8] *Run-length encdoings, Author: S. Golomb, Published in: IEEE Transactions on Information Theory (Volume: 12, Issue: 3), Date of Publication: July 1996* Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1053907, Accessed: January 26, 2024.

[9] *SHORTEN: Simple lossless and near-lossless waveform compression, Author: Tony Robinson, Published by: Cambridge University, Date of Publication: December 1994* Available at: https://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/robinson_tr156.pdf, Accessed: January 26, 2024.

[10] *Introduction to Digital Speech Processing, Author: Lawrence R. Rabiner and Ronal W. Shafer, Published by: now Publishers Inc., Date of Publication: 2007* Available at: https://research.iaun.ac.ir/pd/mahmoodian/pdfs/UploadFile_2643.pdf, Accessed: January 30, 2024.

[11] *FLAC, Format, Author: Josh Coalson, Published by: Xiph Org Fundation, Date of Publication: December 1994* Available at: https://xiph.org/flac/format.html, Accessed: January 29, 2024.

8 Appendix