Olufsen, Emil
Svingeseth, Kristian O'Brien
Østgård, Hågen

Passive acoustic localization

Detection of distance and direction using multiple microphones

Bachelor's thesis in Automation and Intelligent Systems
Supervisor: Hoff, Kai Erik
May 2024

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

NTNU
Norwegian University of
Science and Technology

Olufsen, Emil
Svingeseth, Kristian O'Brien
Østgård, Hågen

Passive acoustic localization

Detection of distance and direction using multiple microphones

Bachelor’s thesis in Automation and Intelligent Systems
Supervisor: Hoff, Kai Erik
May 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

0.1 Preface

This bachelor thesis was written by three students from NTNU in Ålesund: Kristian O'Brien Svingeseth, Hågen Østgård and Emil Olufsen. They were all students of NTNU's Automation and intelligent systems class. The bachelor task was originally requested by Kongsberg Group as a task for the bachelor students in 2023.

NTNU CPS-lab offered this project to our group due to our interest in the signal processing subject, although Kongsberg did not wish to get involved again. On the other hand, Vard showed some interest, but due to some delays, their influence on the project was minimal and their role can be summarised as an experienced advisor. CPS-lab is the Cyber-Physical Systems Lab at NTNU in Ålesund and mainly consists of members who are part of the bachelor program in Engineering and Automation.

0.2 Acknowledgement

First, we would like to thank CPS-lab for the task definition, access to their resources, and for funding the project.

Next, we would like to show our gratitude to our supervisor Kai Erik Hoff, who has encouraged, guided us, and showed support throughout this project. His knowledge and experience in the subject helped us gain more insight and guided our focus on what was more important.

We also want to show appreciation to Lars Ivar Hatledal for guidance with the STM32 programming environment.

Lastly, we would like to show our appreciation to Vard for offering to share knowledge and advice from the industry.

0.3 Abstract

This thesis shows the development and results of a passive sound localization detector. The device includes several signal-processing implementations. Physical components were designed and manufactured locally with the intention of 3D printing and laser cutting in mind. This proved to be a cheap, fast, and efficient method for building the model.

The use of an STM32 microcontroller provided its challenges, but it resulted in outputting the four separate audio channels that we needed. The use of Python for audio processing provided a quick and easy implementation of new features as well as the ease of prototyping new functions.

Testing the simulated and physical system shows reliable results under controlled circumstances when finding the angle ($\sigma = 0.62°$), fundamental frequency, and estimated ship size efficiently. The test also revealed areas of improvement, especially when it comes to audio cut-off while recording.

0.4 Abstract - Norwegian

Denne rapporten viser utviklingen og resultatene av en passiv lyd lokaliserings-sensor. Denne enheten bruker flere signalbehandlings implementasjoner. Fysiske komponenter var designet og laget lokalt med tanker på 3D printing og laserkutting. Dette viste seg å være en billig, rask og effektiv metode for å bygge modellen.

Bruken av en STM32 mikrokontroller viste sine egne utfordringer, men resulterte med å sende ut lydsignalene i fire separate lydkanaler, slik vi trengte. Bruken av Python for å gjøre audio prosessering viste seg å være raskt og enkelt for implementering av nye funksjoner. Dette gjorde det lettere å prototype modellene som ble brukt.

Testingen av simulatoren og det fysiske systemet viste pålitelige resultater under kontrollerte omgivelser ved å finne vinkel ($\sigma = 0.62°$), fundamental frekvens og estimeringen av båt-størrelse effektivt. Testene viste også områder som trenger forbedringer, spesielt når det kommer til lyder som blir kuttet av under opptak.

Table of Contents

List of Figures

List of Codes

Part I

Prepwork for this documentation

Chapter 1

Abbreviations

- **I2S:** Inter-IC Sound
- **I2C:** Inter-Integrated Circuit
- **MISO:** Main in, Subnode Out
- **MOSI:** Main Out, Subnode In
- **MEMS:** Micro-Electro-Mechanical Systems
- **MEMS microphone:** Micro-Electro-mechanical Systems Microphone
- **PDM:** Pulse Density Modulation
- **PCM:** Pulse Code Modulation
- **PWM:** Pulse Width Modulation
- **FFT:** Fast Fourier Transform
- **IFFT:** Inverse Fast Fourier Transform
- **SDFT:** Sliding Discrete Fourier Transform
- **TDOA:** Time Delay Of Arrival
- **HPS:** Harmonic Product Spectrum
- **SWC:** Spectral Weighting Coefficient
- **SNR:** Signal to Noise Ratio
- **PWM:** Pulse Width Modulation
- **ST:** The company STMicroelectronics, also referred to as STM.
- **RSS:** Residual Sum of Squares
- **GUI:** Graphical User Interface

Chapter 2

Notations

- σ: Represents standard deviation in the statistical analysis of the results
- $X_n(k)$: Standard deviation of power density spectrum between all microphones
- α: Coefficient for spectral weighting
- γ: Coefficient for spectral weighting
- d: Maximum distance between two microphones
- $v_s(20)$: The speed of sound at 20°C through air
- f_s: Sampling frequency
- T_s: Time per sample
- t_{max}: Maximum time delay
- $\vec{u}$: Calculated direction
- $\vec{x_{ij}}$: Microphone position
- Rank: Dimension of the subspace spanned by a matrix's rows
- External Noise: Echo, Background sounds and similar unwanted noise
- Internal Noise: All the unwanted noise the internal equipment makes. Noise in cables, conversion from analog to digital, etc.
- COLREGs: Convention on the International Regulations for Preventing Collisions at Sea, 1972
- Pseudo inverse: Generalisation of the inverse matrix

Part II

Introduction

Chapter 3

Background

The safety of the maritime industry is a global concern. There are therefore rules and regulations in place to reduce risk when traveling with seafaring vessels. A heavily regulated situation is reducing the risk of collision between vessels. Two important aspects of reducing collision is communication and vision, in circumstances where vision is impaired it's therefore crucial to maintain good communication to relay position, speed and heading.

In instances where fog or other vision impairing phenomena is prevalent, ships are required to sound off with their foghorn or other acoustic equipment that meet the regulations specified in COLREGs [1].

Examples of situations of lowered vision [2]:

- Fog
- Snowfall
- Rainstorms
- Sandstorms

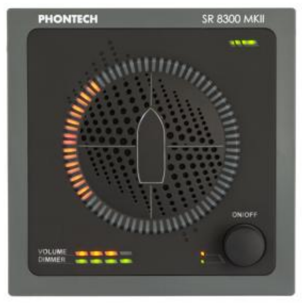
A vessel is required to have multiple sensory and communication systems, as well as a requirement for a "lookout" that listens for the acoustic signal of other vessels in these situations. This creates a potential for a solution that uses digital signal processing to relay positional information of other ships to the crew.

Chapter 4

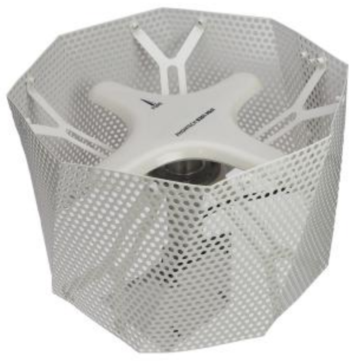
Current solution

There are existing solutions available on the market that utilizes acoustic localisation of seafaring vessels. These were presented to us at a meeting with Vard where they outlined the issues with the current solutions.

There are multiple standards that needs to be followed like ISO 14859, IEC 60945: 2002, IEC 61162-1:2016 / -450:2018. One solution following these standards is the Zenitel P-8300 MkII Sound Reception Display Unit connected to the P-8301 MkII Microphone Unit. It has been approved by DNV (Det Norske Veritas), ABS (American Bureau of Shipping) and CSS (China Classification Society). [3]



(a) P-8300 MkII Sound Reception Display Unit



(b) P-8301 MkII Microphone Unit

Figure 4.1: One of the current solutions

Source: Zenitel [3] & [4]

The microphone antenna [Fig: 4.1b] uses 4 microphones in a array to continuously monitor the surrounding noise in search of a foghorn. The audio from the microphone antenna then goes into the Sound Reception Display Unit [Fig: 4.1a], where it calculates the angle in which the sound comes from.

From the picture [Fig: 4.1a], it shows that the angle estimation is is somewhere between $\sigma = 22.5°$. This is the estimation in real time, but it also displays a more accurate direction after 3 seconds at $\sigma = 5°$. A problem with this deviation is the wide range it can represent.

The problem with this solution as outlined by Vard, is the lack of information presented to the user. The direction functions more like a rough estimate with its wide range as well as no information on the detected boat's size.

Chapter 5

Objective

The system was developed with the following goals in mind:

1. A modular system that allows for expansion
2. Robust source localization
3. Decrease the estimation of the angle compared to today's solution
4. The use of cheap and low powered electronics
5. Estimate the size of a ship/vessel
6. Produce a physical model that can be compared to a simulator

Chapter 6

Structure

The remaining part of this report is structured as follows:

Part 3: Theory Gives an introduction to the theoretical background needed to fully understand and recreate the report.

Part 4: Method Presents how both the simulator and the physical model was built, designed and made.

Part 5: Results Represents the results of the simulator and the physical model, including a comparison to its accuracy.

Part 6: Discussion Contains a summary of the goals, test results and future work.

Part 7: Conclusion Presents the overall project conclusion.

Part III

Theory

Chapter 7

Communication protocols

To receive data between a computer, microcontroller, expansion board, and microphones a form of digital communication must be established. There are different ways of solving this, from singular wire to wireless communication. To achieve this, the project utilizes different wired microcontroller communication protocols with modulated signals.

7.1 Communication bus

7.1.1 I2S

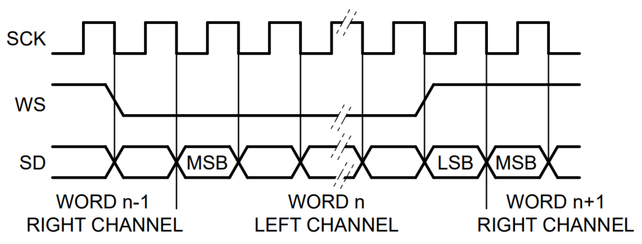


Figure 7.1: Visual representation of I2S data bus.

Source: nxp I2S bus specification [5]

I2S is a standard developed by NXP Semiconductors (formerly Phillips Semiconductors) for digital audio serial linking. The serial data is sent with the most significant bit representing the sign (positive or negative) of the value, this is called two's complement. The most significant bit is sent first as the transmitter and receiver could have different word lengths. The bus only handles the audio data while the sub-coding and control are sent separately. This reduces the number of pins required and the wiring is therefore simpler. The serial bus consists of three lines, one line for two multiplexed data channels (SD), a line for channel selection (Word Select), and one for the clock (SCK) [5]. This protocol is utilized in data transfer between the microcontroller and expansion board within the project [Sec. 15.2.1].

7.1.2 SPI

SPI is a slower but less resource-intensive communication protocol and is commonly used in microcontrollers, SD card readers, and RFID readers. A benefit of SPI is that the data is sent over as packets, which allows for non-interrupted communication. It works by synchronizing the master's output to the bits of the slave unit. A total of one bit is transferred in each clock cycle, which makes the speed dependent on the frequency of the integrated clock. This means that the SPI master and slave have to be synchronized by the clock. There is also an option to run SPI with multiple slave modules, but then the slaves would have to be connected in parallel if the master has multiple slave pins. If the master only has one slave pin, the slaves have to be daisy-chained [6].

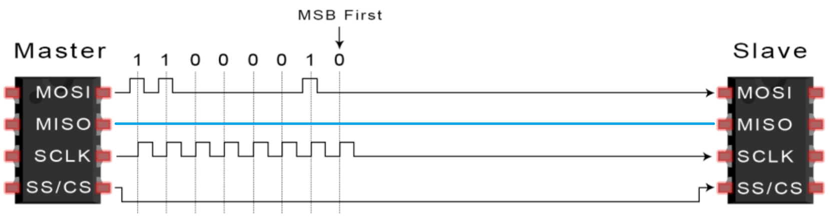


Figure 7.2: SPI communication.

Source: CircuitBasics.com basics of SPI [7] [Modified with blue line to indicate MISO communication]

The communication works by having the master send out the clock signal. Next, it changes the SS/CC pin to a low voltage state, which enables the slave device. Then the master sends a bit at the time over the MOSI line, and the slave reads it when they are received. Then if a response is needed from the slave, it returns one bit at a time over the MISO line [6] (blue line in [Fig: 7.2]). SPI is implemented to send microphone data from the expansion board to the microcontroller in the project [Sec. 15.2.1].

7.2 Modulation

7.2.1 PDM

PDM is the technique of changing the density of the power signal. Or described by Thomas Kite: *However, it is really better summarized as "oversampled 1-bit audio", as it is nothing more than a high sampling rate, single-bit digital system.* [9]

As shown in [Fig: 7.3], the PDM sine wave increases and decreases the length of the on/off signal similar to the analog sine wave's amplitude. This is similar to PWM, but instead of packing all the ones and zeroes together, it distributes the ones and zeroes across the whole period. PDM is used in the microphone recording as the samples are modulated to be represented digitally using this protocol [Sec. 15.2.1].

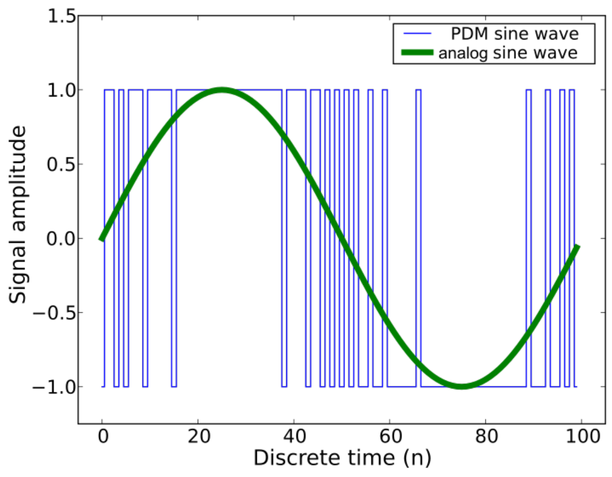


Figure 7.3: PDM sine wave and time domain sine wave

Source: Adafruit PDM microphone breakout [8]

7.2.2 PCM

Pulse code modulation is a method used for digitally representing analog signals and is the most common format for storing digital audio. The analog signal is sampled at regular time intervals and approximated to the nearest digital representation of the read value, the sampling rate is several times higher than the maximum frequency of the sampled signal [10]. PCM utilizes a single sound channel, multi-channel support therefore relies on interleaving or synchronization of multiple PCM streams.[11].

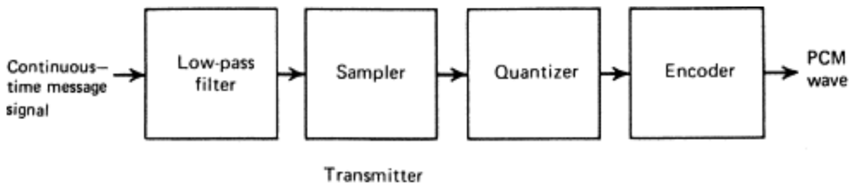


Figure 7.4: Time signal to PCM signal

Source: ScienceDirect PCM [10]

PCM is used to convert the data arrays containing the microphone data on the microcontroller before sending it over USB to the computer [Sec. 15.2.1].

Chapter 8

Filtering

The signals acquired by the microphones are prone to different sources of noise which can make source detection harder as it populates the signal with unwanted information. These sources of noise are defined into two groups in this report.

- **Internal noise:** The noise originating from the wires acting as antennas from the other wires within the system as well as noise from the microcontroller and its different conversion processes.
- **External noise:** The noise originating from the wires acting as antennas from outer electromagnetic waves as well as real-world audible noise such as echo, reverberation, and sounds.

8.1 Filtering

8.1.1 Gibbs phenomenon

The Gibbs phenomenon is a mathematical concept concerning the oscillatory behavior of the Fourier transform. The phenomenon manifests when a function with an instant discontinuity is represented with a Fourier series. No finite amount of sinusoids will be able to mirror the instant change required to be identical to the instant discontinuity [12]. This can be visualized with a Fourier representation of a square wave as seen in [Fig: 8.1].

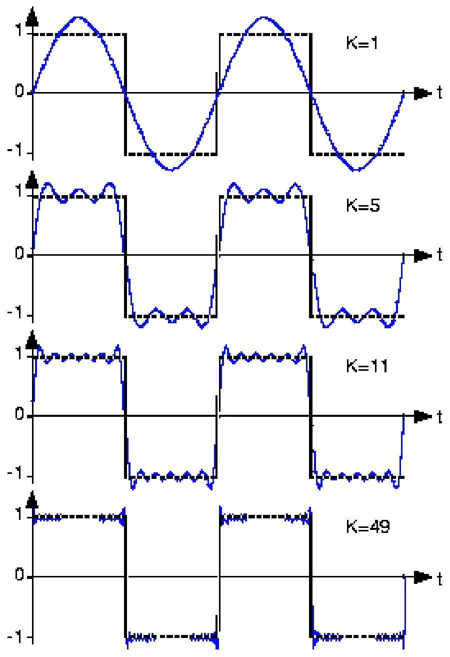


Figure 8.1: Fourier representation of square wave. K is the number of sinusoids for the approximation

Source: eng.libretexts.org [13]

8.1.2 High-pass filter

A high-pass filter is a type of electronic filter designed to allow signals with frequencies above a specified cutoff frequency to pass (shown in blue: [Fig: 8.2]), while reducing the strength of signals with frequencies below this threshold [15].

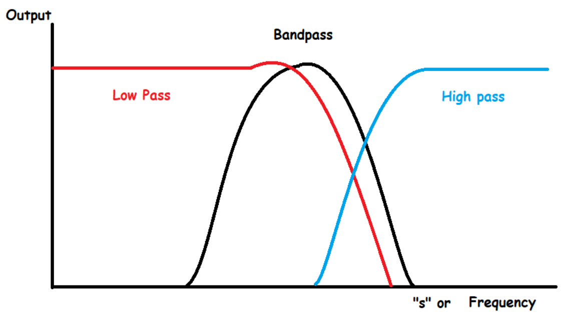


Figure 8.2: Low pass in red, Bandpass in black, High pass in blue

Source: Quora identify type of filter [14]

8.1.3 Low-pass filter

A low-pass filter is a type of electronic filter designed to allow signals with frequencies below a specified cutoff frequency to pass (shown in red: [Fig: 8.2]), while reducing the strength of signals with frequencies above this threshold [15].

8.1.4 Band-pass filter

A band-pass filter is a type of electronic filter that combines the results of a high-pass filter and low-pass filter (shown in black: [Fig: 8.2]). It's designed to allow signals within a specified frequency range to pass while reducing the strength of signals outside this range [15].

8.1.5 Frequency domain filtering

When analyzing the signal in the frequency domain (discussed in chapter 9) it's possible to change the magnitude of frequency bins to reduce or increase the presence of a specific frequency. Setting the unwanted frequencies to zero is considered bad practice in audio signal processing as it leads to instant discontinuities [12].

8.1.6 Spectral whitening filter

A filter can be defined as a spectral whitening filter if it gives a signal's power density spectrum white noise characteristics [16]. Used in our project for lowering the wide maxima caused by correlations within the signal [17].

Chapter 9

Frequency Domain

9.1 Fourier transform

The Fourier transform is a mathematical tool that breaks down a time domain signal into an alternative representation using sine and cosine waveforms. It shows that any waveform can be rewritten into sine and cosine functions [9].

9.1.1 DFT

$$X_k = \sum_{n=0}^{N-1} x[n] \cdot e^{\frac{-i2\pi kn}{N}} \tag{9.1}$$

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{\frac{i2\pi kn}{N}} \tag{9.2}$$

$$DFT_{equations} = O(N^2) \tag{9.3}$$

Proved by [18]

These functions explain how the Fourier transform is the sum of all the sinusoidal functions of the original signal. As shown in figure [Fig: 9.1] the time-based waveform is split into its sinusoidal frequencies before its plotted onto the frequency domain amplitude plot. [19]

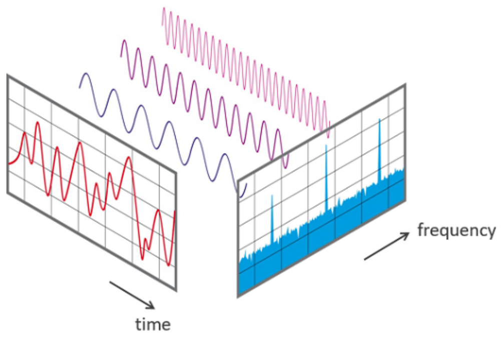


Figure 9.1: Fourier transform from time domain to frequency domain

Source: NTI-audio [19]

9.1.2 FFT

Cooley and Tukey developed in 1965 the fast Fourier transform. FFT also converts the signal into individual sinusoidal components, like the regular Fourier transform. [18] The difference is that FFT uses an optimized algorithm of the Fourier transform. It only requires:

$$FFT_{equations} = O(N \log_2(N)) \tag{9.4}$$

Proved by [18]

equations, which is a lot less than DFT (as shown in [Fig: 9.2])

This means that the FFT algorithm requires less computational power with a higher N value compared to regular DFT. The FFT is faster, but using N as an integer power of 2 is heavily recommended. There are ways of calculating the FFT for other values, but the merits of N as an integer power of 2 make it close to mandatory. [20]

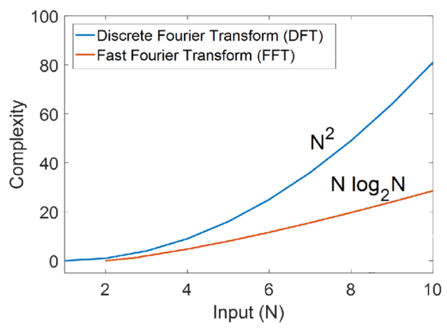


Figure 9.2: DFT vs FFT

Source: researchgate (edited image, removed QFT.) [21]

This is the most common way to get the discrete Fourier transform in programming, due to Python's Numpy and other code languages implementation.

9.1.3 IFFT

IFFT is the inverse FFT. It is used to go from the frequency domain and back to the time domain. IFFT is described by the formula (9.5)

$$X_k = \frac{1}{N} \cdot X[k] \tag{9.5}$$

Proved by [22]

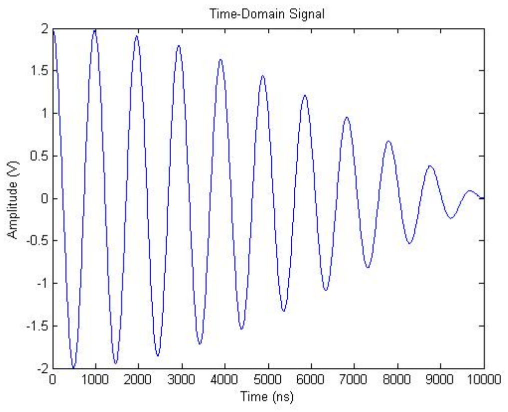
9.2 Frequency Domain compared to Time domain

The frequency domain allows for digital control of an audio signal. It is both faster and cheaper to change audio parameters digitally, although it can lead to corruption or data loss if handled incorrectly. The loss of data is minuscule and often not noticeable. There is often more loss of data when digitizing analog audio due to a smaller sample rate. Another reason to use the frequency domain is the use of discrete controllers. These include Zero-order hold, Tustin's method, Discrete approximation of differential equations, and more. This is useful since it can easily be implemented on computers, microcontrollers, and other digital systems. [23]

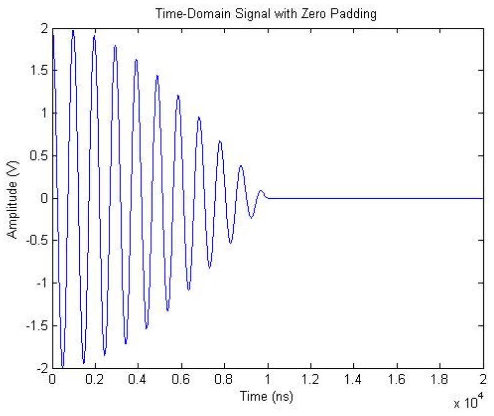
Chapter 10

Zero padding

Zero padding is the concept of adding data points consisting of zeroes to a signal in the time domain.



(a) Example 1 MHz and 1.05 MHz real-valued sinusoidal waveforms



(b) Same plot, but added 1000 samples of zero to do zero padding

Figure 10.1: Example of zero Padding of a signal

Source: NTI-audio [19]

The reason to use zero padding in our project is to increase the amount of samples in a signal. This can help make the signal easier to work with by allowing it to be the length of other signals or multitudes of that signal. Allowing the signal length to be multitudes of another signal helps us in the cross-correlation, explained in [Sec. 11.2].

Chapter 11

Correlation

11.1 Cross-correlation

Cross-correlation calculates the similarity between two signals. One data set is kept as it is, while the other gets shifted for comparison, commonly time-shifted. Resulting in a peak when the datasets overlap [Fig: 11.1]. The time shift is represented on the cross-correlation's x-axis, with correlation on the y-axis. [24]

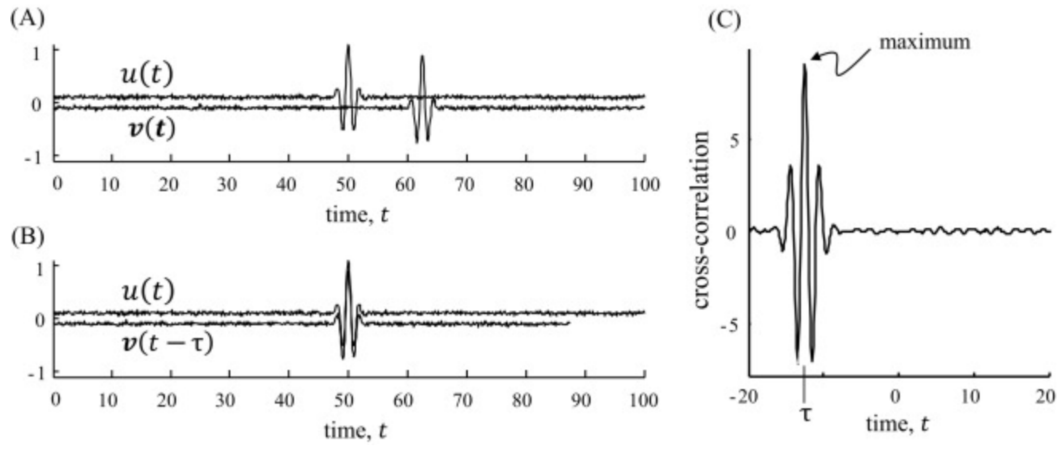


Figure 11.1: Example of cross-correlation

Source: Environmental data analysis [24]

The model for discrete cross-correlation is shown in [eq. (11.1)]. u and v from [Fig: 11.1] becomes x_i and x_j in [eq. (11.1)].

$$R_{ij}(\tau) = \sum_{n=0}^{N-1} x_i[n]x_j[n-\tau] \tag{11.1}$$

From [17]

The discrete model is useful, however, it is more relevant for this project to calculate cross-correlation in the frequency domain, as shown in robust [17]. The process is shown in [eq. (11.2)]. Where the Fourier transform of both signals is multiplied by each other, while the time-shifted signal is conjugated. The inverse Fourier transform can then be applied to show the time component [17].

$$R_{ij}(\tau) \approx \sum_{k \approx 0}^{N-1} X_i(k) X_j(k)^* \times e^{\frac{i2\pi k\tau}{N}} \tag{11.2}$$

From [17]

11.2 Circular and non circular cross-correlation

The circular cross-correlation [eq. (11.3)] will naturally assume that the signal is periodic when using DFT. This can be problematic when cross-correlating non-periodic signals. A more realistic model for this would be the unbiased cross-correlation [eq. (11.4)] since it does not assume this cyclic behavior. Where the new limit prevents the cross-correlation from extending beyond the overlapping signals [25].

This can alternatively be achieved through zero padding in practice. This is best described by [Fig: 11.2] which shows how a circular convolution can wrap around a finite signal, and how zero padding prevents this. It also allows for a linear convolution between two finite length signals using DFT [26]. This is true for convolution, and in turn also true for the cross-correlation, given their similar definitions [eq. (11.1)] and [Fig: 11.2].

$$\hat{r}^u_{xy} \triangleq \frac{1}{N} \sum_{n=0}^{N-l} x(\bar{n}) y(n+l), l = 0, 1, 2, ..., N-1 \tag{11.3}$$

From [25]

$$\hat{r}^u_{xy} \triangleq \frac{1}{N-1-l} \sum_{n=0}^{N-l} \bar{x}(n) y(n+l), l = 0, 1, 2, ..., L-1 \tag{11.4}$$

from [25]

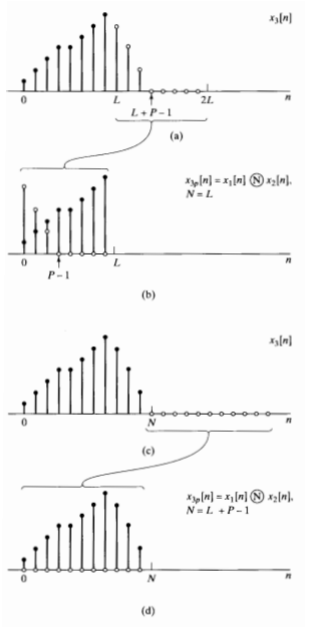


Figure 11.2: Zero padding and convolution

Source: [26]

Chapter 12

Sound source localization using TDOA

12.1 TDOA

Time delay of arrival (TDOA) can be used for measuring both the angle and position of an object in space. It utilizes at least three microphones for sound localization, making its estimates from the distance between each receiver, the difference in time of arrival of sound, and its velocity. [27]

It is important to note that positional estimates are only accurate for objects in the microphone array center. As shown in [Fig: 12.1]. However angular estimates are accurate also in the area where positional estimates are invalid, as long as it is far between the source and array. [29] [27]

A common method for finding the delay of arrival is cross-correlation [eq. (12.1)] [27] [29].

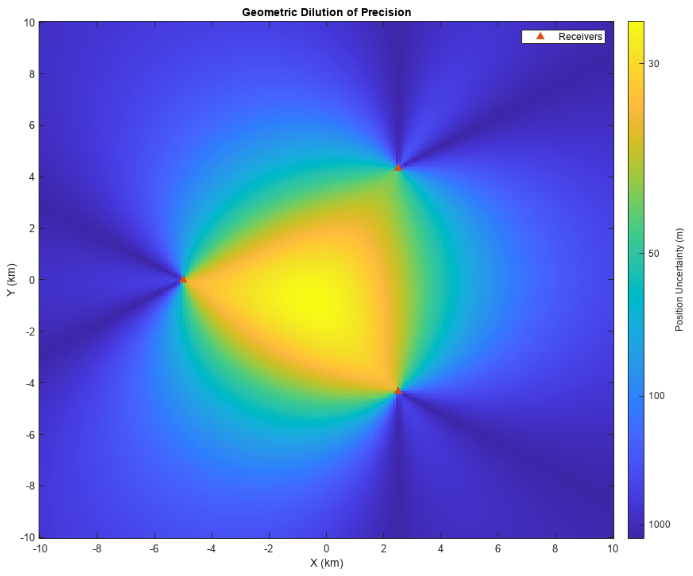


Figure 12.1: Positional accuracy TDOA
(Red triangles: microphones)

Source: Mathworks [28]

$$
\begin{aligned}
TDOA &= TOA_2 - TOA_1 \\
TDOA &= argmax(S1 \star S2)
\end{aligned}
\tag{12.1}
$$

From [28]

12.2 Least squares

Least squares come in different variants depending on the goals, parameters, and complexity of a model. The method discussed here is ordinary least squares. The goal is to obtain estimates of parameters that minimize a quantity called RSS (residual sum of squares) between measured and estimated values [30].

$$S(\beta) = \sum_{i=1}^{n} (y_i - f(x_i, \beta))^2 \tag{12.2}$$

Where y_i are the observed values, x_i are the explanatory variables, and $f(x_i, \beta)$ is the model function parameterized by β. This approximation is used in our distance estimation algorithm.

12.3 Angle and distance

TDOA in angle and distance estimates utilizes trigonometry as shown in [Fig: 12.2]. Two expressions are derived from it, [eq. (12.3)] and [eq. (12.4)], both equal to $\cos(\phi)$ [17]. Both equations are set equal to each other, and $||\vec{x_{ij}}||$ cancels out. The result is the expression shown in [eq. (12.5)].

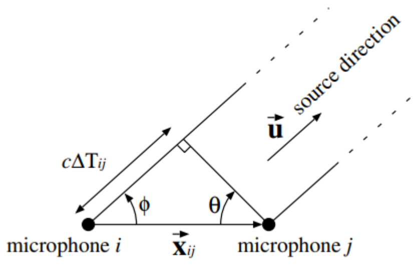


Figure 12.2: Trigonometry for source direction

Source: Robust sound source localization [17]

$$\cos(\phi) = \frac{\vec{u} \cdot \vec{x}_{ij}}{||\vec{u}||||\vec{x}_{ij}||} = \frac{\vec{u} \cdot \vec{x}_{ij}}{||\vec{x}_{ij}||} \tag{12.3}$$

$$\cos(\phi) = \sin(\theta) = \frac{c\Delta T_{ij}}{||\vec{x}_{ij}||} \tag{12.4}$$

$$\vec{u} \cdot \vec{x}_{ij} = c\Delta T_{ij} \tag{12.5}$$

[eq. (12.3)], [eq. (12.4)] and [eq. (12.5)] is proved by [17]

The final equation [eq. (12.5)] shows that the source direction ($\vec{u}$) can be found as long as the relative microphone position ($\vec{x_{ij}}$), speed of sound (c), and time delay between microphones (ΔT_{ij}) is known. It can then be further generalized [eq. (12.6)] and organized in matrix formation [eq. (12.7)] for solving with multiple microphones. [17]

$$u(x_j - x_I) + v(y_j - y_i) + w(z_j - z_i) = c\Delta T_{ij} \tag{12.6}$$

From [17]

$$\begin{bmatrix} (x_2 - x_1) & (y_2 - y_1) & (z_2 - z_1) \\ (x_3 - x_1) & (y_2 - y_1) & (z_3 - z_1) \\ \vdots & \vdots & \vdots \\ (x_N - x_1) & (y_N - y_1) & (z_N - z_1) \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} c\Delta T_{12} \\ c\Delta T_{13} \\ \vdots \\ c\Delta T_{1N} \end{bmatrix} \tag{12.7}$$

From [17]

12.4 Power Spectral Density

The general idea behind power spectral density is estimating the total power distribution of different frequencies within a finite signal. It's useful in many fields and applications, like speech analysis, vibration monitoring, and economics to name a few. For radar and sonar systems, the spectral contents can provide information to localize sources or targets from the signal [31].

- $\hat{\phi}_p(\omega)$ represents the estimated power spectral density at frequency ω.
- $\frac{1}{N}$ is a normalization factor where N is the number of samples.
- $\sum_{t=1}^{N} y(t)e^{-i\omega t}$ is the Discrete Fourier Transform (DFT) of the signal $y(t)$.
- $|\cdot|^2$ indicates taking the magnitude squared of the DFT result.

$$\hat{\phi}_p(\omega) = \frac{1}{N} \left| \sum_{t=1}^{N} y(t)e^{-i\omega t} \right|^2 \tag{12.8}$$

Source: Spectral Analysis of Signals [31]

Chapter 13

Classification of ships

To classify a ship the Norwegian ”Rules of the seas” (Sjøveisreglene), say that

- A ship over 200 meters in length must have a fog horn with a frequency within the 70 to 200 hertz range.
- A ship between 75 and 200 meters must have a fog horn with a frequency within the 130 and 350 hertz range.
- A ship smaller than 75 meters must have a fog horn within the 250 to 700 hertz range.

[32]

13.1 HPS

There are a few ways of classifying the ship with the known info above, but one of the solutions is to find the harmonic base frequency. That is because the harmonic base frequency is what the frequency of the actual signal is. This can be solved using a HPS algorithm.

$$Y(\omega) = \prod_{r=1}^{R} |X(\omega r)| \tag{13.1}$$

$$\hat{Y} = \max_{\omega_i} \quad \{Y(\omega_i)\} \tag{13.2}$$

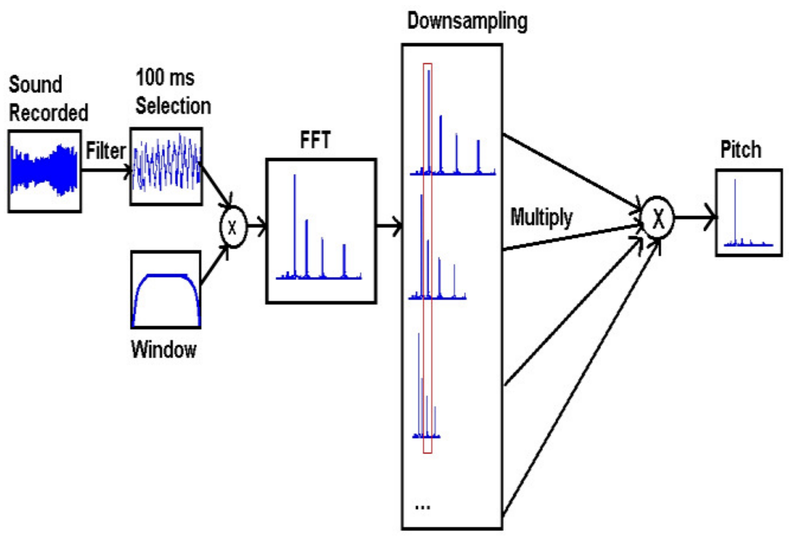


Figure 13.1: HPS algorithm overview

Source: ResearchGate [33]

As seen in the figure above [Fig: 13.1] the HPS algorithm starts by taking an audio source or signal and changing the values to the frequency domain by using FFT. It then down-samples the signals by an integer factor. This is to emphasize the harmonic signature of the signal. Next, it combines the spectra by multiplying them together. This process enhances the periodicity of harmonics present in the signal. Lastly, it detects the peaks of the signal, which correspond to the fundamental frequency and all of its harmonics.[33]

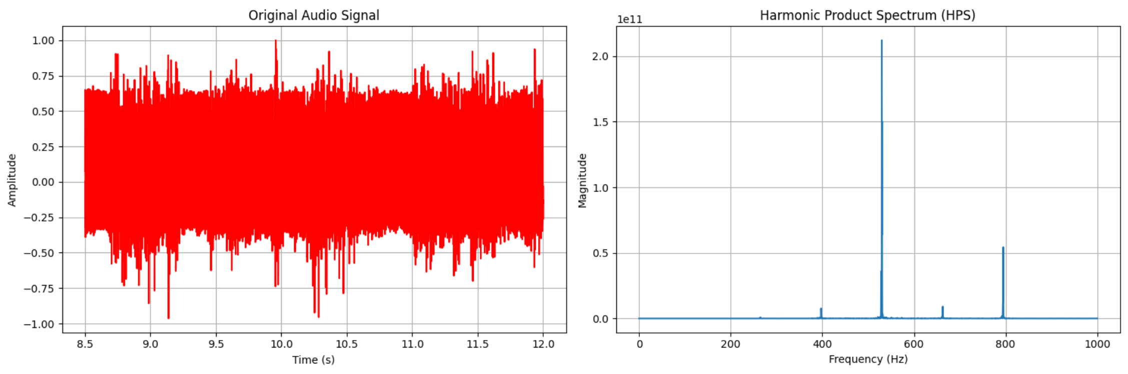


Figure 13.2: Audio source and its HPS

As shown in the figure above [Fig: 13.2]. The left plot shows the audio source with its amplitude over time. The plot to the left is the HPS of the audio source and shows the amplitude over the frequency. That shows what frequency the fundamental harmonic frequency of the original signal is.

The problem with classifying the sips will be a multitude of factors. some of these issues are:

- Overlapping frequencies
- Different intervals depending on the action of other ships (changing speed, reversing, etc)
- Non-ship fog horns (Lighthouse, boat plane, military ships, etc)

Part IV

Method

Chapter 14

Simulator

The goal of the simulations was to tune the parameters of the system and to check its accuracy. Data used for testing was sine waves and fog horns with Gaussian white noise added on top. The simulator's requirement was to run the project algorithm without the need for physical microphones.

14.1 Generating simulated environment

The simulated environment consisted of a Cartesian coordinate system represented in meters. The challenge of the simulated environment was to mimic the distance of each microphone to the sound source. It was therefore necessary to ”place” the audio source within the environment to find the corresponding delay that would be present in the microphones in a real-world system. The true position of the sound source was known, but the goal of the simulator was to find the location using the algorithm so the true location was only used to create the simulated time delay.

14.2 Mic placement in coordinate system

The virtual microphones had to be placed in the virtual environment, this was done by defining a location matrix where each row represented a new microphone and the columns defined the X and Y coordinates accordingly.

$$\begin{bmatrix} Mic0_x, & Mic0_y \\ Mic1_x, & Mic1_y \\ Mic2_x, & Mic2_y \\ Mic3_x, & Mic3_y \end{bmatrix}$$

```
Coord0 = np.array([0, -0.36])  # Coordinates of mic0 compared to center (m)
Coord1 = np.array([-0.36, 0])  # Coordinates of mic1 compared to center (m)
Coord2 = np.array([0, 0.36])  # Coordinates of mic2 compared to center (m)
Coord3 = np.array([0.36, 0])  # Coordinates of mic3 compared to center (m)
```

Code 1: Placing microphone's position into the mic position matrix

14.3 Trigonometry

Before creating the time delay, the distance from the sound source to the microphones had to be found using the known location of the sound source. This distance was crucial since it specified the amount of delay to append to each microphone.

```
    # Calculate the distance from the sound source to the microphones
distanceMic0 = np.linalg.norm(actualLocation - mic0Location)
distanceMic1 = np.linalg.norm(actualLocation - mic1Location)
distanceMic2 = np.linalg.norm(actualLocation - mic2Location)
distanceMic3 = np.linalg.norm(actualLocation - mic3Location)
```

Code 2: Distance from sound source to microphones.

The simulator utilized the Numpy Linalg norm function. This function calculated the norm (or length) of a vector. When used with the difference between two points, it computed the Euclidean distance between those two points.

14.4 Time delay between microphones

Using the previously found distance between each microphone and dividing them with the speed of sound at 20°Celsius (343m/s). It was possible to find the time it would take for the sound waves to reach each microphone, this then represented the delay for each microphone in seconds [34].

```
# Calculate what the time delay is using the distance to the microphones.
timeDelayMic0 = distanceMic0/343
timeDelayMic1 = distanceMic1/343
timeDelayMic2 = distanceMic2/343
timeDelayMic3 = distanceMic3/343
```

Code 3: Calculate the time delay from the audio source to the microphones.

14.5 Signal shifting

The delay for the microphones was then converted from seconds into the number of indexes to shift the signal to simulate the time delay. This was done by multiplying the time delay by the sampling rate of the signal, which was done in four instances to find the shift for each microphone.

```
# Create empty data points to use in shifting the signal
signalShift0 = np.zeros(round(timeDelayMic0*fs))
signalShift1 = np.zeros(round(timeDelayMic1*fs))
signalShift2 = np.zeros(round(timeDelayMic2*fs))
signalShift3 = np.zeros(round(timeDelayMic3*fs))
```

Code 4: Creating empty data points representing delay for each separate signal.

When the empty data arrays with the number of shifts were generated it was ready to be applied to the signal, this was where the four data arrays were created to simulate the four microphone channels. The delay was inserted at the start of each array with the Numpy shift function, it was important to note that this would cause the arrays to be of different lengths to each other.

```python
# Shift the signal using the empty data points, this will cause the arrays to be
↪  asynchronous in length
long_shiftedSignal0 = np.insert(signal0, 0, signalShift0, axis=0)
long_shiftedSignal1 = np.insert(signal1, 0, signalShift1, axis=0)
long_shiftedSignal2 = np.insert(signal2, 0, signalShift2, axis=0)
long_shiftedSignal3 = np.insert(signal3, 0, signalShift3, axis=0)
```

Code 5: Inserting the zero data points at the start of each signal to integrate the time delay.

14.6 Cutting down the audio array

The data arrays had to be of the same length so that they would stay synchronized in time for the cross-correlation. This was simply done by cutting off the end of the signal to reduce the arrays to the original size. This led to data loss, but for our use case and testing the approach was deemed functional and had no impact on our results.

```python
# Cut down the signals so they are of equal and original length, this will cause
↪  some information loss
shiftedSignal0 = long_shiftedSignal0[0:len(signal0)]
shiftedSignal1 = long_shiftedSignal1[0:len(signal1)]
shiftedSignal2 = long_shiftedSignal2[0:len(signal2)]
shiftedSignal3 = long_shiftedSignal3[0:len(signal3)]
```

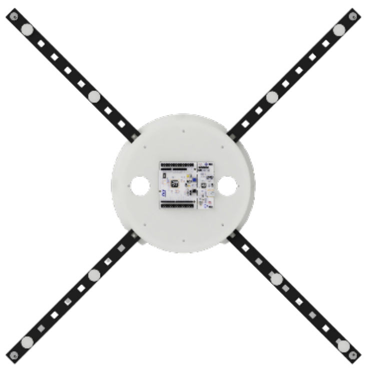
Code 6: Removing data points from the end of each signal to make them equal in length.

A different approach that was considered was zero-padding the shorter arrays to be of the same length as the longest array. The problem with this solution would be the abrupt change from sound to zero indexes. This could impact the frequency domain characteristics of the signal and would require a window function or similar technique to taper the data.

Chapter 15

Real world system

15.1 3D models


(a) Top view of the rig, where it shows the placement of the mics and tags

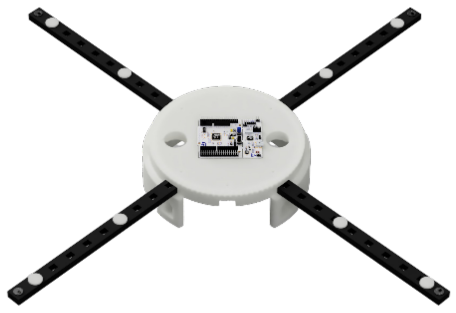
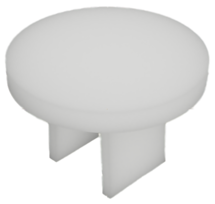
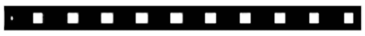

(b) Side view of the rig where you can see cable passthrough

Figure 15.1: The rig of this project

(a) The housing of the rig (main body)


(b) Name-pin. Used to name each arm mic


(c) The arm elements that allow for different placements of the microphones

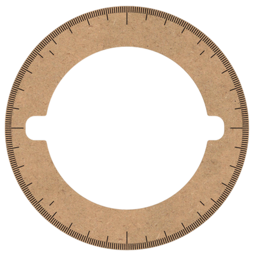

(d) The degree disc

Figure 15.3: 3D models split up into the individual parts

Housing

The main component of the 3D model is the housing. It's designed to be both modular but also secure all the components together. On the top side, there is space for the STM32 F446RE to sit flush, while also passing through the cables and wires to the microphones and the jetson nano (or a computer). There are a total of 8 placements for the arms [Sec. 15.1], which allows the modularity of where the microphones are placed. Underneath the holes for the arms, there are some cable passthrough holes. There are screw holes to fasten and secure the arms in place on top of the housing, as well as 90 markers that represent 4° of rotation. On the inside/underneath the housing, 4 legs extend down and is designed to screw down the jetson nano without having the components on its underside touch the housing.

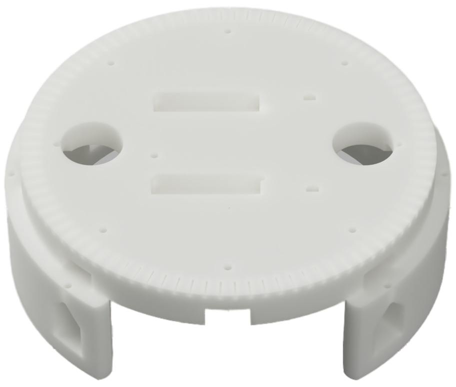


Figure 15.2: The housing of the rig

Arms

The arms are designed to fit into each of the 8 openings on the main housing. It is also designed to have multiple openings that allow the placement of microphones and tags along the length of the arm. There is a small circular hole at one of the ends that allows the fastening with a screw to the housing, which ensures that it's secure.

Pins

The pins are designed to fit onto the arms. It should be printed in white/-black and use a contrasting marker to write on it. It's a quite simple design but holds well onto the arms.

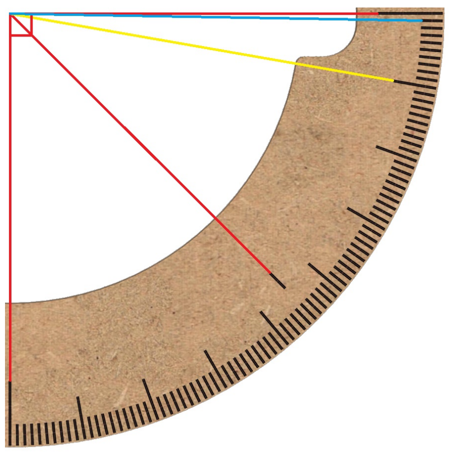


Figure 15.4: Red: 45°, Yellow: 10°, Blue 1°

Degree Disk

The degree disk is designed to fit over the main housing and lay on top of the arms. It has 360 small markers that indicate each of the 360 °. There are also 36 longer markers which indicate each 10 ° as well as 8 inner markers (4 line up with the longer lines), which show the 45 ° marks.

Wiring

The wires between the microphones and the expansion board made a lot of noise. That was probably due to them picking up electromagnetic signals from each other and the devices around them. The solution was to twist the wires and tape them together to secure them. This helped reduce a lot of the internal noise.

15.2 Hardware

Parts used for the project can be split into two categories, a microphone array and a data processing unit. The microphone array is constructed by a microcontroller [Sec. 15.2.1] and an expansion board [Sec. 15.2.1] with mems-microphones attached [Sec. 15.2.1]. The data processing unit chosen is a small ARM64 computer [Sec. 15.2.2].

The reasoning behind using different boards for data acquisition and processing is ease of use. The STM32 F446RE microcontroller is considered by the supervisor as an easy way to collect data. The STM32 F446RE allows for programming in C/C++ and has many resources available online, including pre-built libraries that become relevant later [Sec. 15.2.1]. The nanoc100 is a computer running Linux, opening the possibility of processing data in Python. This is advantageous since Python has predefined signal processing libraries which the group has experience with from the signal processing course AIS2201 [35].

15.2.1 STM32 NUCLEO64

STM32 F446RE

The STM32 F446RE development board is a part of the STM Nucleo-64 development board family. This board was chosen due to its reliability, open development style, price, and features. The STM32 microcontrollers are designed to be flexible with multiple options and features.

The STM32 F446RE is designed with ease of use with its many GPIO pins. It includes the regular Arduino R3 headers as well as alternative GPIO pins for use with STM's expansion boards.

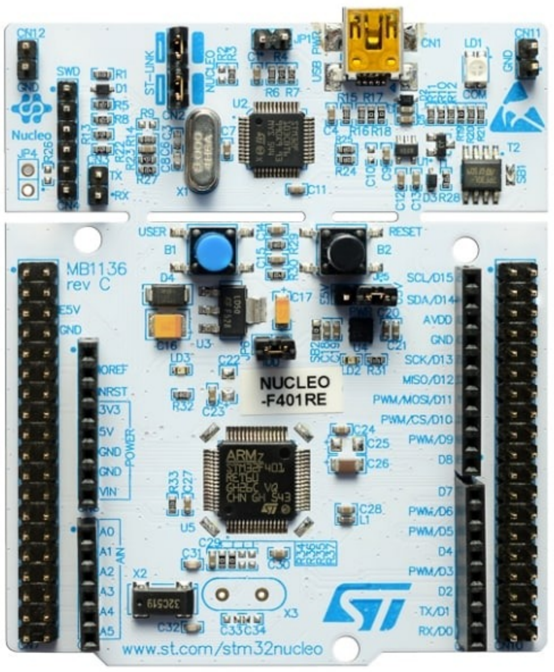


Figure 15.5: STM32 F446RE microcontroller

Source: STMicroelectronics [36]

The STM32 F446RE microcontroller is responsible for data acquisition. It supports serial communication, I2S, and PCM. Serial communication is necessary for sending data to the processing unit. I2S is used as the microphone communication standard and PCM

STEVAL-MIC002v1

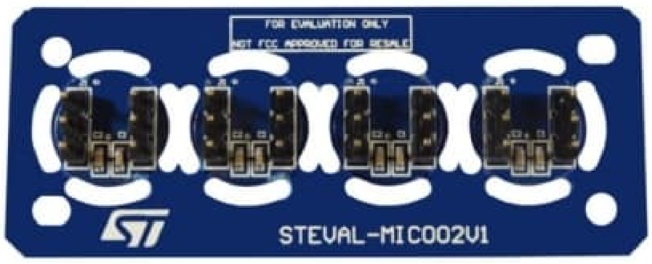


Figure 15.6: STEVAL-MIC002v1

Source: STMicroelectronics [37]

Steval-MIC002v1 is a coupon board consisting of four MP34DT06J MEMS microphones specifically designed to be used with the CCA02M2 expansion board [37]. The compatibility with the board as well as their omnidirectional sensitivity is the reasoning behind the inclusion of this component. The microphone coupons are designed to be broken off and placed in a wanted configuration.

X-NUCLEO-CCA02M2

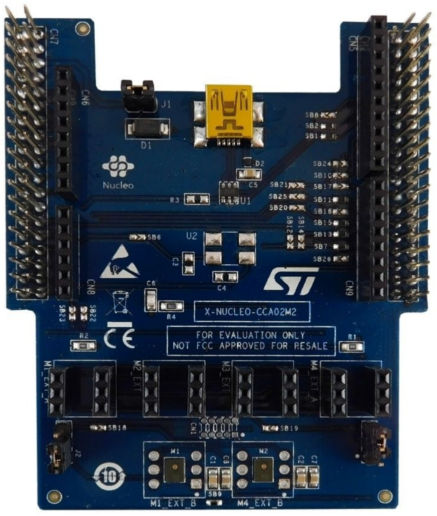
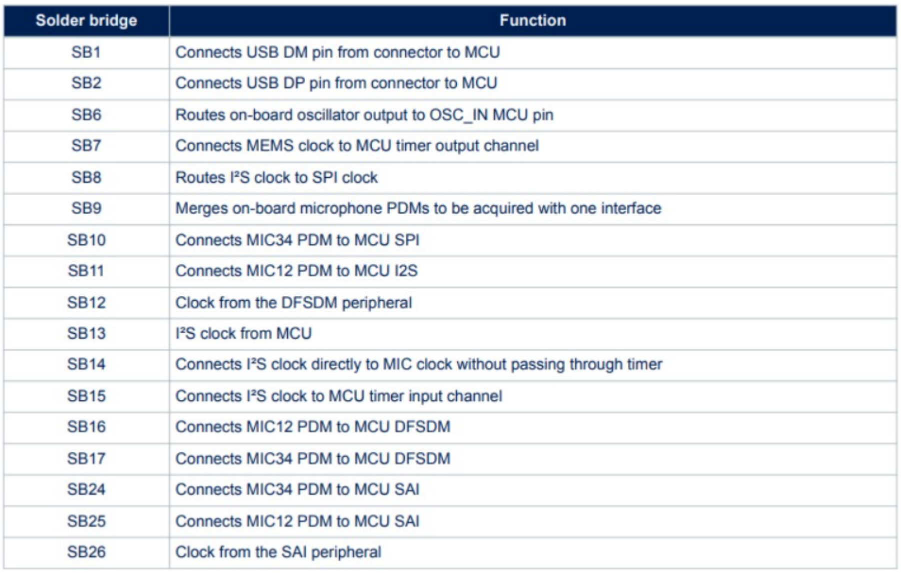


Figure 15.7: STM32 X-NUCLEO-CCA02M2 expansion board

Source: STMicroelectronics [38]

CCA02M2 is an expansion board that allows for streaming of up to four microphones in real time [38]. It comes pre-mounted with two MP34DT06J microphones, as well as ST's proprietary connectors for easy connection to the microcontroller. The expansion board supports multiple hardware configurations and streaming formats, which means the board had to be modified for the project goal. The different solder bridge configurations [Fig: 15.8] are outlined in the CCA02M2 documentation [38].



Solder bridge	Function
SB1	Connects USB DM pin from connector to MCU
SB2	Connects USB DP pin from connector to MCU
SB6	Routes on-board oscillator output to OSC_IN MCU pin
SB7	Connects MEMS clock to MCU timer output channel
SB8	Routes I²S clock to SPI clock
SB9	Merges on-board microphone PDMs to be acquired with one interface
SB10	Connects MIC34 PDM to MCU SPI
SB11	Connects MIC12 PDM to MCU I2S
SB12	Clock from the DFSDM peripheral
SB13	I²S clock from MCU
SB14	Connects I²S clock directly to MIC clock without passing through timer
SB15	Connects I²S clock to MCU timer input channel
SB16	Connects MIC12 PDM to MCU DFSDM
SB17	Connects MIC34 PDM to MCU DFSDM
SB24	Connects MIC34 PDM to MCU SAI
SB25	Connects MIC12 PDM to MCU SAI
SB26	Clock from the SAI peripheral

Figure 15.8: STM32 X-NUCLEO-CCA02M2 solder mapping.

Source: STMicroelectronics [38]

CCA02M2 Soldering

The soldering changes to be made for four mic acquisition were outlined in the documentation of the FP-AUD-SMARTMIC1 function pack [39]. The first change to be made was adding two more external microphone headers above the two internal microphones as shown in [Fig: 15.9].

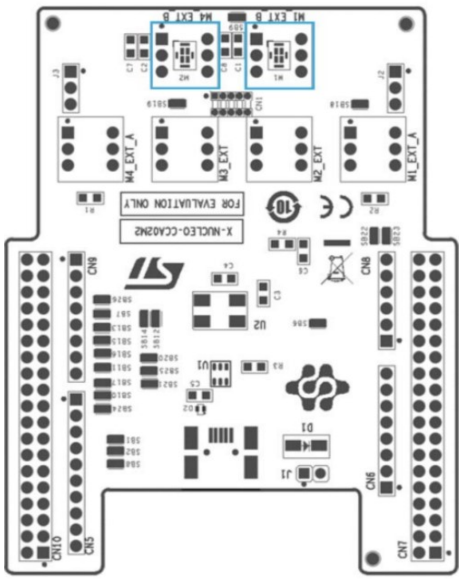


Figure 15.9: Soldering on microphone headers. Headers to be mounted in blue squares.

Source: STMicroelectronics [39]

The two internal microphones were not used in the system so they had to be disabled. The next soldering step disables these and enables the headers instead. The next changes also define the communication protocol, Mic1 and Mic2 send the PDM signal with I2S to the MCU. While Mic3 and Mic4 send the PDM signal with SPI. All soldering changes are shown in [Fig: 15.10] with their respective changes shown in [Fig: 15.8].

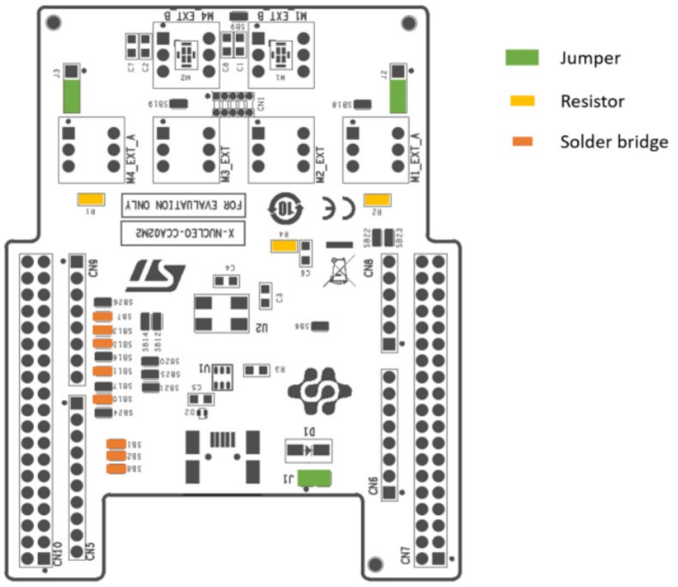


Figure 15.10: Solder bridge configuration for four mic acquisition.

Source: STMicroelectronics [39]

STM32 software (programming interpreter)

The program used for testing and implementing the changes to the library was STM32CubeIDE. It functions as an IDE with integration with the ST product catalog. the program also allows for custom definitions to the I/O pins for the MCU as well as changing the clock signals and internal protocols [40]. The I/O and clock functions were not required for the project as the current configuration uses a slightly modified function pack developed by ST.

STM32 code

For audio acquisition, the FP-AUD-SMARTMIC1 function pack from ST was utilized[39]. The pack comes equipped with various functions suitable for the project. It's also distributed with a built-in demo that the project uses for recording audio with small adjustments to the code.

STMicroelectronics' demo comes equipped with an audio recording from multiple microphones, beam forming, general acoustic localization, active noise canceling, and audio output to an external loudspeaker board. The acoustic localization function of the demo was deemed to be too general and the goal was to get the microphone data into Python for localization with more freedom and options. There were multiple problems with using this demo unaltered for audio recording as well. Beamforming and noise cancellation are automatically applied in the standard configuration.

Another problem with the unaltered demo is that the microphone data sent over USB consists of two channels. Both channels contain interleaved data from only two microphones, with one channel filtered and the other unfiltered as shown in [Fig: 15.11].

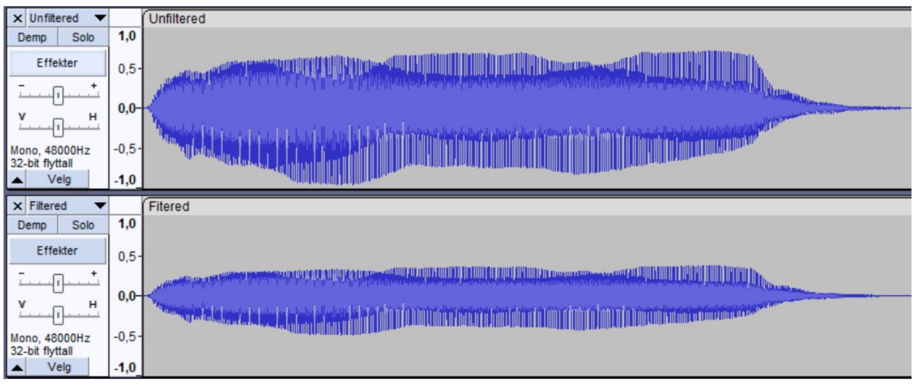


Figure 15.11: Visualization of FP-AUD-SMARTMIC1 data recording.

The demo was a good starting point, but some alterations had to be made to fit our requirements, these requirements were:

- All microphones streaming in real time via USB. Preferably one channel for each separate microphone.
- Microphone data is "raw" and each microphone is treated the same, as in no beam forming or noise cancellation.
- Microphone is supported in Linux.
- Data is reachable by Python script for calculation and analysis.

Two distinct changes had to be made for data acquisition from all four microphones over USB with 4 channels. Firstly, the configuration for the CCA02M2 has a line for defining the amount of channels to use as input. This helps shape the buffer array to support 4 channels as well as correctly placing each microphone data point at each index.

```c
#define AUDIO_IN_CHANNELS                4
#define AUDIO_IN_SAMPLING_FREQUENCY      16000

#define AUDIO_IN_BUFFER_SIZE             DEFAULT_AUDIO_IN_BUFFER_SIZE
#define AUDIO_VOLUME_INPUT               64U
#define CCA02M2_AUDIO_IN_INSTANCE        0U
#define CCA02M2_AUDIO_IN_IT_PRIORITY     6U
```

Code 7: Changed from 2 to 4 audio in channels in cca02m2_conf.h.

A similar change was also applied to the CCA01M1 configuration for better compatibility with the code, even though the CCA01M1 expansion board is not present in our system. There are multiple references to the definitions in this configuration so we changed the number of output channels from 2 to 4.

```c
#define AUDIO_OUT_CHANNELS    4
#define AUDIO_OUT_SAMPLING_FREQUENCY  16000

#define AUDIO_OUT_BUFFER_SIZE           (AUDIO_OUT_SAMPLING_FREQUENCY/1000 * 2 * 8)
#define AUDIO_VOLUME_OUT          30U
#define CCA01M1_AUDIO_OUT_INSTANCE  1U
#define CCA01M1_AUDIO_OUT_IT_PRIORITY   6U
```

Code 8: Changed from 2 to 4 audio out channels in cca01m1_conf.h

To achieve the raw data from each microphone being sent through USB, we created a for loop that iterates over every index from the PCM in buffer copying them to the corresponding index in the PCM out buffer. This creates an interleaved data structure that both Windows and Linux systems can unpack to separate the microphone channels. This data structure is easier to visualize by looking at each iteration of the for loop as a frame. These frames are packed together in the buffer and sent through USB using the predefined Send_Audio_to_USB() function.

```c
// // Iterate over every sample point input from the microphones. Set each output index to corresponding mic.
for (i = 0; i < AUDIO_IN_SAMPLING_FREQUENCY / 1000; i++)
{
        aPCMBufferOUT[AUDIO_OUT_CHANNELS * i] = aPCMBufferIN[AUDIO_IN_CHANNELS * i]; // Mic1 to channel 1
        aPCMBufferOUT[AUDIO_OUT_CHANNELS * i + 1] = aPCMBufferIN[AUDIO_IN_CHANNELS * i + 1]; // Mic2 to channel 2
        aPCMBufferOUT[AUDIO_OUT_CHANNELS * i + 2] = aPCMBufferIN[AUDIO_IN_CHANNELS * i + 2]; // Mic3 to channel 3
        aPCMBufferOUT[AUDIO_OUT_CHANNELS * i + 3] = aPCMBufferIN[AUDIO_IN_CHANNELS * i + 3]; // Mic4 to channel 4
}

        Send_Audio_to_USB((int16_t *)aPCMBufferOUT, AUDIO_IN_SAMPLING_FREQUENCY / 1000 * AUDIO_OUT_CHANNELS);
}
```

Code 9: For loop to create each frame of data. Then send data via USB.

$$\begin{aligned}
\text{Frame}_0 &= \begin{bmatrix} Mic0_0, & Mic1_0, & Mic2_0, & Mic3_0 \end{bmatrix} \\
\text{Frame}_1 &= \begin{bmatrix} Mic0_1, & Mic1_1, & Mic2_1, & Mic3_1 \end{bmatrix} \\
&\vdots \\
\text{Frame}_n &= \begin{bmatrix} Mic0_n, & Mic1_n, & Mic2_n, & Mic3_n \end{bmatrix}
\end{aligned} \tag{15.1}$$

The PCMBufferOUT is then filled with the frames consisting of one sample from each microphone. These frames are placed consecutively as shown in [eq. (15.2)].

$$\text{PCMBufferOUT} = \begin{bmatrix} Frame_0, & Frame_1, & \dots & Frame_n \end{bmatrix} \tag{15.2}$$

15.2.2 OKdo nano C100

The OKdo nano C100 is a Jetson nano developer kit that uses the Jetson nano 4 GB module. It runs a modified version of Nvidia Jetpack version 4.6. The jetpack software is a modified version of Ubuntu 18.04 LTS, which is made to run on an ARM processor. This ARM64-based computer allows the use of multiple devices and has a Quad-core ARM A57 CPU, a NVIDIA® Maxwell GPU, and 4 GB 64-bit LPDDR4. It runs on 5V over a DC barrel jack or micro USB power, which makes it suitable for use with the STM32 F446RE [Fig: 15.5] microcontroller. The Nvidia Jetson module was designed to be used with machine learning and algorithms which makes it offer great processing power compared to the form factor, which makes it suitable for the project.

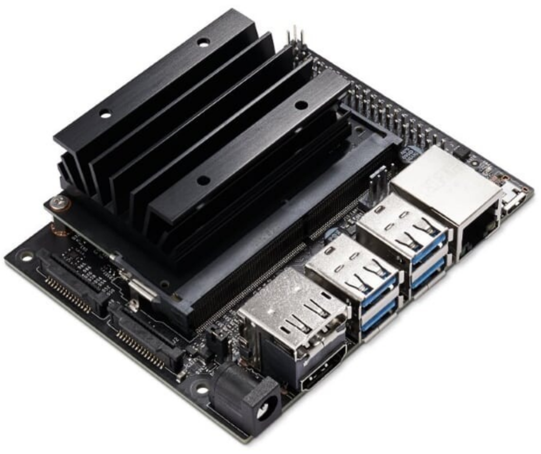


Figure 15.12: OKdo nano C100 developer kit

Source: Nvidia [41]

As seen in the picture above [Fig: 15.12], the OKdo nano C100 has native support for up to 4 USB 3.0 ports. These ports allow the use of 2 STM32 F446RE [Sec. 15.2.1] and X-NUCLEO-CCA02M2 [Sec. 15.2.1] modules to be used.

Jetson Nano Software

The OKdo nano as mentioned in [Sec. 15.2.2] runs a modified version of Ubuntu 18.04 LTS. This is a Linux distro that is simple to use with its included GUI mode.

Python libraries

The Linux distro comes with Python 3.6.9 pre-installed and includes some basic packages like sudo, apt, and pip.

The Python packages used are:

Package name	Version	specified modules
Python	3.9	n/a
queue	standard	n/a
threading	standard	n/a
matplotlib	3.7.1	backends.backend_tkagg.FigureCanvasTkAgg & figure.Figure
numpy	1.19.4	n/a
scipy	1.11.3	signal & signal.spectrogram & Optimize.minimize
pyaudio	0.2.11	n/a
tkinter	standard	filedialog
time	standard	n/a
PIL	standard	Image
math	standard	n/a
sys	standard	n/a
wave	standard	n/a

15.3 Record audio

To record the audio from external microphones the use of pyaudio is used. The full process of collecting audio recordings to be used for the angle and distance calculation is split up into 3 parts. The first part [Code: 10] and [Code: 11] pics up the external microphones as an audio source, and saves the data into 4 different arrays. One for each microphone channel.

```python
# Function to save 4 induvidual channels (only saves data when the array is
↪  fully finished)
def write_audio(data, fileName, Audio_instance, FORMAT, RATE):
    wf = wave.open(fileName, 'wb')
    wf.setnchannels(1)
    wf.setsampwidth(Audio_instance.get_sample_size(FORMAT))
    wf.setframerate(RATE)
    wf.writeframes(b''.join(data))
    wf.close()

# Function to get the 4 channel mic array into python for further processing
def audioGet(CHUNK, FORMAT, CHANNELS, RATE, RECORD_SECONDS, MIC_INDEX):
    # Indicate that its working and recording
    print("* recording")
    #Initiate the pyaudio instance and set the recording variables
    pyaudio_instance = pyaudio.PyAudio()

    stream = pyaudio_instance.open(format=FORMAT,
                                   channels=CHANNELS,
                                   rate=RATE,
                                   input=True,
                                   frames_per_buffer=CHUNK,
                                   input_device_index=MIC_INDEX
                                   )

    # Creating the "frames" buffer arrays
    frames0 = []
    frames1 = []
    frames2 = []
    frames3 = []
```

Code 10: Reads audio from microphones and saves them in frames arrays

```python
# Save the data into the frames arrays for futher processing
for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
    data = stream.read(CHUNK, exception_on_overflow=False)

    # convert string to numpy array
    data_array = np.frombuffer(data, dtype='int16')

    # deinterleave, select 1 channel
    channel0 = data_array[0::CHANNELS]
    channel1 = data_array[1::CHANNELS]
    channel2 = data_array[2::CHANNELS]
    channel3 = data_array[3::CHANNELS]

    # convert numpy array to string
    data0 = channel0.tobytes()
    data1 = channel1.tobytes()
    data2 = channel2.tobytes()
    data3 = channel3.tobytes()
    frames0.append(data0)
    frames1.append(data1)
    frames2.append(data2)
    frames3.append(data3)

# Indicate that the recording is done and its time to save the 4 channel audio
↪  files
print("* done recording")

stream.stop_stream()
stream.close()
pyaudio_instance.terminate()
```

Code 11: Saves the data from frames array for further processing

The second part [Code: 12] of the code takes the individual microphone channel array and saves them as a .wav file. This is done to ensure that a full array is saved before continuing to the localization part.

```python
write_audio(frames0, 'out_ch0.wav', pyaudio_instance, FORMAT, RATE)
write_audio(frames1, 'out_ch1.wav', pyaudio_instance, FORMAT, RATE)
write_audio(frames2, 'out_ch2.wav', pyaudio_instance, FORMAT, RATE)
write_audio(frames3, 'out_ch3.wav', pyaudio_instance, FORMAT, RATE)
```

Code 12: Saves the buffer arrays into a finished .wav file for further processing

The third part of the code [Code: 13] takes the .wav file generated from [Code: 12] and turns them back into arrays. This is done due to the audio processing requiring finished arrays, and not arrays that get overwritten after a certain amount of time.

```python
ch0Wav = wave.open("out_ch0.wav", "r")
signal_0 = ch0Wav.readframes(-1)

ch1Wav = wave.open("out_ch1.wav", "r")
signal_1 = ch1Wav.readframes(-1)

ch2Wav = wave.open("out_ch2.wav", "r")
signal_2 = ch2Wav.readframes(-1)

ch3Wav = wave.open("out_ch3.wav", "r")
signal_3 = ch3Wav.readframes(-1)

soundwave_ch0 = np.frombuffer(signal_0, dtype="int16")
soundwave_ch1 = np.frombuffer(signal_1, dtype="int16")
soundwave_ch2 = np.frombuffer(signal_2, dtype="int16")
soundwave_ch3 = np.frombuffer(signal_3, dtype="int16")

return soundwave_ch0, soundwave_ch1, soundwave_ch2, soundwave_ch3
```

Code 13: Uses the .wav file to fill in new arrays ready for audio processing

This makes it so the remaining code only works on a full array of data, and finishes with that data before it gets overwritten again.

Chapter 16

Audio processing

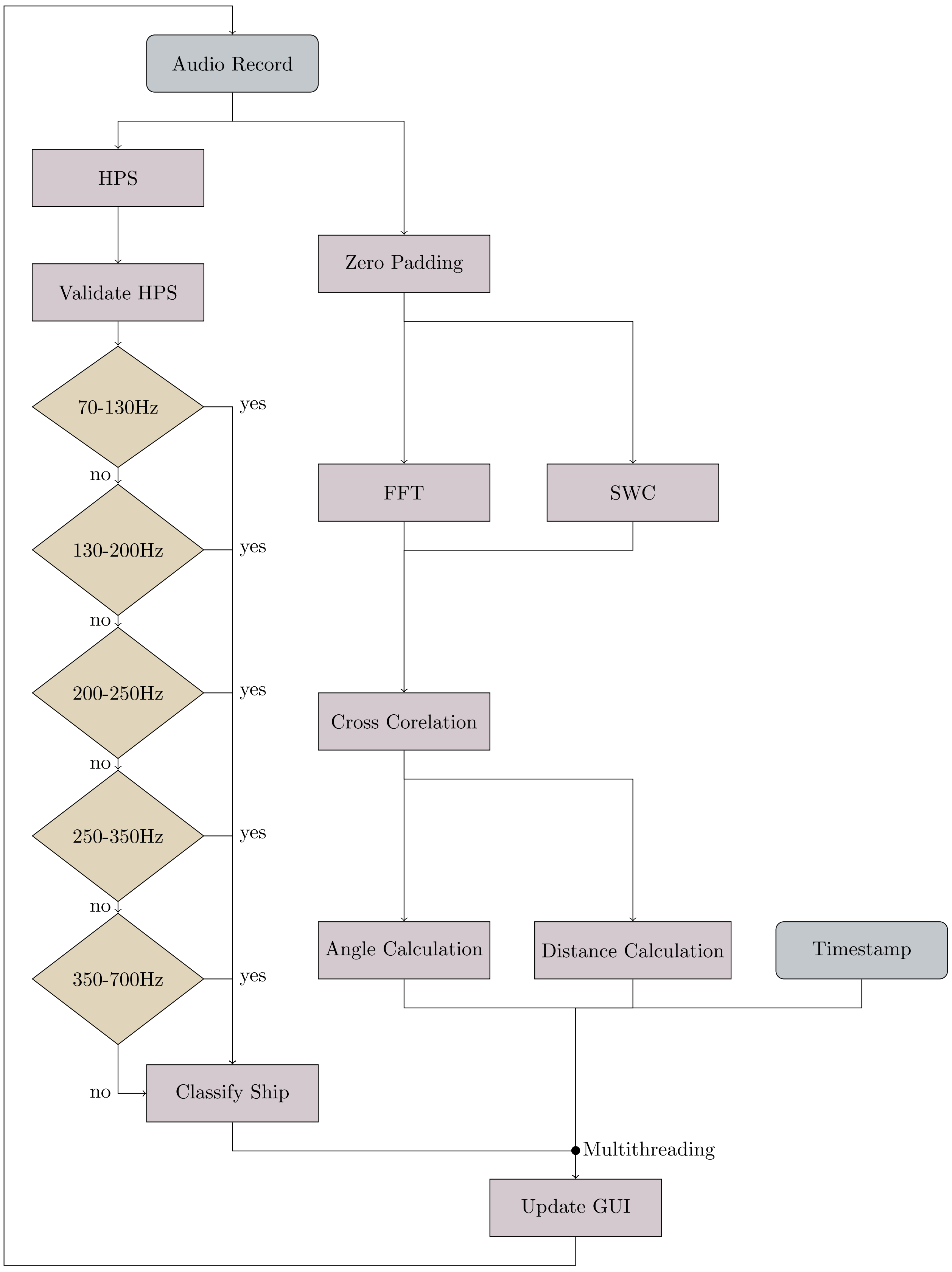
The audio from either the simulator [Ch. 14] or the real-world system [Ch. 15] needs to be further processed. To figure out the angle, distance, and to classify a ship there are different processes to do so. There is a process flowchart [Sec. 16.1] to get a general overview of the audio processing.

The audio processing is split into three main parts. The first one is the HPS algorithm [Sec. 16.3], which uses the raw recorded audio to find the fundamental frequency.

The second part is the angle and distance estimation [Sec. 16.7] which relies on some audio processing before being calculated. Here the audio gets a zero-padded and runs through a cross-correlation before any actual angle or distance can be estimated.

The third part takes the values from parts 1 and 2 and places them in a GUI [Sec. 16.8]. This is done in a different thread (multithreading). That is done to make the system more responsive, as it doesn't have to compute the GUI elements after each audio sample.

16.1 Process flow



16.2 Filter

Since the noise will impact our model's ability to differentiate the signals it's crucial to either suppress the unwanted frequencies or increase the presence of the wanted frequencies. The frequency range of interest is 70-700Hz as stated in section 13.1. Several strategies for filtering were considered.

16.2.1 Band-pass filter

This type of filter was considered as a combination of low-pass and high-pass filtering for removing the frequencies outside the wanted range but was scrapped as it could change the signal properties relating to the cross-correlation and HPS approach, making it harder to differentiate signal characteristics.

16.2.2 Frequency domain filtering

The removal of unwanted frequency bins was considered but was scrapped as it could change the signal characteristics in unforeseen ways. Doing this imitates a "square filter" and is considered bad practice as the response of the filter is prone to the Gibbs phenomenon and other unexpected behavior.

16.3 Ship classification

16.3.1 HPS

To find the fundamental frequency of a fog horn or any similar sounds, the use of an HPS algorithm was used. This was done by the code [Code: 14] created using OpenAI's ChatGPT-4 [42]. To find the fundamental frequency of the HPS, a few steps were involved. The first step was to find the FFT of the signal, and then the spectrum by using the absolute value of the FFT of the signal. Next up was to set the spectrum array from zero to the length of the spectrum array divided (whole number division) by 2.

```python
####
# Made in tandem with ChatGPT 4.0
# View code attachments for further details
####

def find_fundamental_frequency(signal, sampling_rate):
    # Compute the FFT
    spectrum = np.fft.fft(signal)
    spectrum = np.abs(spectrum)

    # Initialize HPS
    hps = spectrum[:len(spectrum) // 2].copy()

    # Downsample and multiply (HPS)
    for downsample_factor in range(2, 5):  # Typically up to 4 or 5
        downsampled = spectrum[::downsample_factor]
        hps[:len(downsampled)] *= downsampled

    # Find the peak in the HPS
    fundamental_freq_index = np.argmax(hps)
    fundamental_frequency = fundamental_freq_index * sampling_rate / len(signal)

    return fundamental_frequency
```

Code 14: HPS algorithm to find the fundamental frequency, created using OpenAI's ChatGPT-4 [42].

The signal got down-sampled within the specific range before the maximum value and its index of the down-sampled spectrum array was found. The last step was to use the indexes and multiply it by the sampling rate and divide the total by the length of the signal. This function returns the fundamental frequency of the input.

16.3.2 Validating HPS

There are multiple ways of validating the fundamental frequency of each of the microphones. One way is to use the average fundamental frequency found from all the microphones [Code: 15]. Another way is to use the median value instead [Code: 16].

The code [Code: 15], got the fundamental frequency of all the microphones, and found an average value from these. It then checked each microphone's fundamental frequency compared to the average. This code had a big weakness, it required that there isn't an outlier value. Meaning that one microphone could change the deviation number by a significant amount.

Another way was to do it like the code [Code: 16], which used the median frequency from the microphones. This reduced the effect of an outlier microphone affecting the deviation value. The other way was that only two of the microphones had to be within the threshold of the deviation to be accepted as the fundamental frequency. This increased the chance that a value was returned but was less accurate if there was a random noise applied to only two of the microphones.

```python
def validateFundementalFrequency(freq0, freq1, freq2, freq3):
    treshold = 20
    avgFreq = (freq0 + freq1 + freq2 + freq3) / 4

    deviation0 = abs(freq0 - avgFreq)
    deviation1 = abs(freq1 - avgFreq)
    deviation2 = abs(freq2 - avgFreq)
    deviation3 = abs(freq3 - avgFreq)
    print("dev: ", deviation0)
    print("dev: ", deviation1)
    print("dev: ", deviation2)
    print("dev: ", deviation3)
    if (deviation0 < treshold) and (deviation1 < treshold) and (deviation2 <
    ↪  treshold) and (deviation3 < treshold):
        return medianFreq
    else:
        return 0
```

Code 15: Validating the fundamental frequency of all microphones based on the average value

```python
def validateFundamentalFreqMedian(freq0, freq1, freq2, freq3):
    frequencies = np.array([freq0, freq1, freq2, freq3])
    medianFreq = np.median(frequencies)
    stdDev = np.std(frequencies)

    # Adaptive threshold: 1 standard deviation
    threshold = stdDev

    # Calculate deviations from the median
    deviations = np.abs(frequencies - medianFreq)
    print("Deviations: ", deviations)

    # Check which frequencies are within the threshold
    valid_frequencies = frequencies[deviations < threshold]

    # If at least half the frequencies are valid, calculate their average;
    ↪  otherwise, return 0
    if len(valid_frequencies) >= len(frequencies) / 2:
        return np.mean(valid_frequencies)
    else:
        return 0
```

Code 16: Validating the fundamental frequency of all microphones based on the median value

16.4 Zero padding

The audio array from [Code: 13] would most likely be non-periodic. This could cause issues for the reasons mentioned in [Sec. 11.2]. Zero padding was applied because of this as shown in [Code: 17]. The code took the sound array, found the length of it (n0), and doubled it (n). The doubling was used as the targeted new length of the signal. Then a conversion was done (nfft), converting the length to a number that was an integer power of two, for reasons mentioned in [Sec. 9.1.2]. Lastly, it applied zero padding, extending the signal length to match nfft.

```python
####
# Made in tandem with ChatGPT 4.0
# View code attachments for further details
####

def zeroPadding(sound):
    n0 = len(sound)
    n = n0 + n0
    nfft = 2 ** np.ceil(np.log2(n)).astype(int)

    sound = np.pad(sound, (0, nfft - n0), 'constant', constant_values=0)
    return sound
```

Code 17: Zero Padding the audio arrays, created using OpenAI's ChatGPT-4 [42]

16.5 Spectral weighting function and spectral whitening

16.5.1 The spectral whitening function

The spectral whitening was done the same as in robust sound source localization [17]. Using the absolute value in each element of the cross correlation to flatten the magnitude of the signal [eq. (16.1)].

$$R_{ij}^{(e)}(\tau) = \sum_{k=0}^{N-1} \frac{X_i(k)X_j(k)^*}{|X_i(k)||X_j(k)|} \times e^{\frac{i2\pi k\tau}{N}} \tag{16.1}$$

16.5.2 The spectral weighting function

A spectral weighting function was also applied in the same manner as in robust sound source localization [17]. Its purpose was to help the cross-correlation identify the correct time delay, as a countermeasure for the whitening.
First $X(k)$ and $X_n(k)$ was found. X(k) was defined by robust [17] as the mean value of the power density spectrum between all microphones. The definition of X_n was unclear but interpreted as the standard deviance of the same data set. The values were calculated in python [Code: 18]. The power density spectrum of the data from each microphone was calculated and stored in powerDensitySpectrumArray. The data was then used to find both the mean (X) and standard deviation (Xn).

```python
# signals from all microphones
signals = [soundwave_ch0, soundwave_ch1, soundwave_ch2, soundwave_ch3]

# Array containing power density spectrum for dataset from each microphone
powerDensitySpectrumArray = [np.abs(np.fft.fft(signal))**2/len(signal) for
↪  signal in signals]

 # converted to numpy array for further processing
powerDensitySpectrumArray = np.array(powerDensitySpectrumArray)

# Find the average and standard deviance between microphones
X = np.mean(powerDensitySpectrumArray, axis=0)
Xn = np.std(powerDensitySpectrumArray, axis=0)
```

Code 18: mean and standard deviation of magnitude spectrum

Then the weighing function was defined. w(k) and $w_e(k)$ [eq. (16.2)] were also taken from robust sound source localization [17]. w(k) was defined first. It took the largest value between 0.1 and an expression scaled by the parameter α. The expression used the difference between mean and standard deviance to filter noise not simultaneously present in all microphones. The second part of the weighting function was originally included in robust sound source localization to recognize tonal data. It was included with the hope that it would help for fog horns as well [17]. An interpretation of [eq. (16.2)] was implemented in python [Code: 19] as a continuation of [Code: 18].

$$
\begin{aligned}
w(k) &= max\left(0.1, \frac{X(k) - \alpha X_n(k)}{X(k)}\right) \\
w_e(k) &= \begin{cases} w(k) & , \quad X(k) \leq X_n(k) \\ w(k)(\frac{X(k)}{X_n(k)})^{\gamma} & , \quad X(k) > X_n(k) \end{cases}
\end{aligned}
\tag{16.2}
$$

From [17]

```python
# define coefficient array for population in the mathematical function
w = np.zeros(len(X)) + 0j

# mathematical function defining the weighing arrays coefficients
for k in range(len(X)):
    if X[k] <= Xn[k]:
        w[k] = np.maximum(0.1, (X[k] - alpha * Xn[k]) / X[k])
    else:
        w[k] = np.maximum(0.1, ((X[k] - alpha * Xn[k]) / X[k]) * (X[k] / Xn[k])
        ↪  ** gamma)
```

Code 19: Weighting function in python

The final implementation was done in the frequency domain 16.3, the same way as in robust sound source localization [17] [Code: 20].

$$
R_{ij}^{(e)}(\tau) = \sum_{k=0}^{N-1} \frac{w_e^2(k) X_i(k) X_j(k)^*}{|X_i(k)||X_j(k)|} \times e^{\frac{i2\pi k\tau}{N}} \tag{16.3}
$$

```
mean and standard deviation of magnitude spectrum}
R = (np.square(W) * fftSound1 * np.conj(fftSound2)) / (np.abs(fftSound1) *
↪  np.abs(fftSound2)
```

Code 20: Weighting (w) with cross-correlation

16.6 Correlation

16.6.1 Cross correlation

The cross-correlation was first implemented without weighting [Code: 21].
First, the zero-padded signal length was found (nfft), which could be done by repeating the calculations done in the zero-padding function. This was used later when cross-adjusting the data

The cross-correlation was done in the frequency domain (R) before it was converted back to the time domain (r). However the dataset was indexed with only positive indexes by default [Fig: 16.1], so a cross-adjustment function was necessary to shift the time delays as shown in [Fig: 16.2]. But it was not necessary to shift the whole data set as shown in [Fig: 16.2]. So only the index of the cross-correlations peak was shifted, as long as it was found past the dataset's halfway point.

Lastly, the time delay (TDOA) was multiplied by the speed of sound for processing in the mathematical model 16.7.1.

```
####
# Made in tandem with ChatGPT 4.0
# View code attachments for further details
####

def crossAdjustment(array_padded, nfft_local, fs_local):
    delay_index = np.argmax(np.abs(array_padded))
    if delay_index > nfft_local / 2:
        delay_index -= nfft_local

    return delay_index / fs_local

def crossCorelation(fftSound1, fftSound2, fs, sound):
        n0 = len(sound)
        n = n0 + n0

        nfft = 2 ** np.ceil(np.log2(n)).astype(int)

        R = fftSound1 * np.conj(fftSound2)
        r = np.fft.ifft(R)

        r = abs(r)

        TDOA = -crossAdjustment(r, nfft, fs)
        dist = TDOA * 343

        return dist
```

Code 21: Cross-correlation, created using OpenAI's ChatGPT-4 [42]

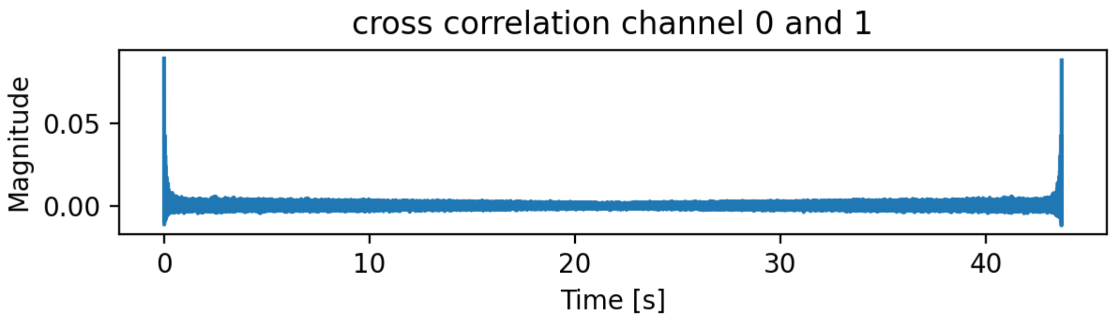


Figure 16.1: Default cross correlation

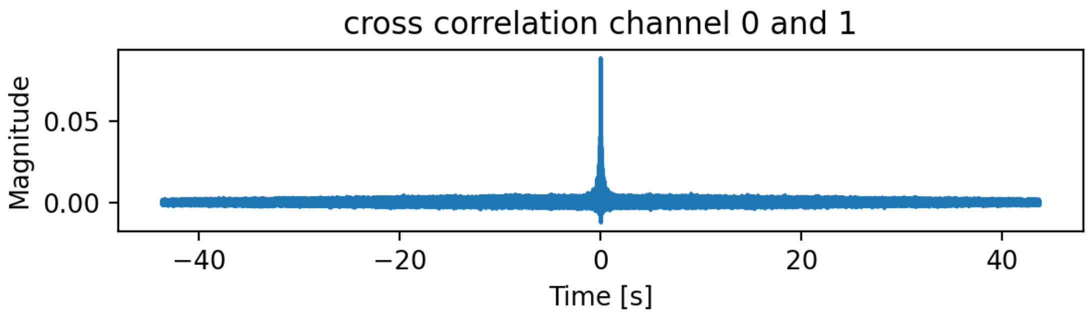


Figure 16.2: cross adjusted cross corelation

16.6.2 Cross correlation with weighting

The weighting was implemented together with the whitening function. The reason for this was that the weighting function was described as a countermeasure for the issue of noise in the flattened signal [17].

It was implemented in the frequency domain [Code: 22] as defined in robust sound source localization [17].

```
R = (np.square(W) * fftSound1 * np.conj(fftSound2)) / (np.abs(fftSound1) *
↪  np.abs(fftSound2))
```

Code 22: Cross Correlation with whitening/weighting

16.6.3 Narrower search area for cross correlation

There was a method of filtering out false positives from the time delay that was proposed. From the cross-correlation, we set the indexes above a certain threshold to zero so the values of interest are maintained. The speed at which sound propagates helped us set this cutoff point from where to expect unreasonable delays given the sampling rate.

1. **Maximum Distance d:**

$$d = 0.72\,\mathrm{m} \tag{16.4}$$

   Firstly, we define the maximum distance d at 0.72 meters. This represents the longest possible distance between any two microphones.

2. **Speed of Sound at 20°C v_s:**
$$v_s = 343\,\text{m/s} \tag{16.5}$$

   Then we assume 20°C for the speed of sound [34].

3. **Sampling Frequency f_s:**
$$f_s = 16000\,\text{Hz} \tag{16.6}$$

   Sampling frequency f_s, which is set at 16000 Hz, determines how often the microphones sample per second.

4. **Time per Sample T_s:**
$$T_s = \frac{1}{f_s} = \frac{1}{16000} = 6.25 \times 10^{-5}\,\text{s} \tag{16.7}$$

   Each sample takes T_s seconds, calculated as the inverse of the sampling frequency.

5. **Maximum Time Delay $t_{\max}$:**
$$t_{\max} = \frac{d}{v_s} = \frac{0.72}{343} \approx 2.1 \times 10^{-3}\,\text{s} \tag{16.8}$$

   The maximum time delay $t_{\max}$ is the time it takes for sound to travel the maximum distance at the given speed of sound.

By assuming the speed of sound at 20°C it was important to note how this would affect the method at temperatures deviating from this assumption. We compared the behavior of the parameters by testing at 30°C.

The changes to the speed of sound at different temperatures were taken into account as given by [eq. (16.9)] [43].

$$v_s = 331\sqrt{\frac{T}{273K}}(\text{m/s}) \tag{16.9}$$

Difference in seconds

At 20°C:
$$t_{\max}(20) = \frac{d}{v_s(20)} = \frac{0.72}{343.42} \approx 0.002097\,\text{s} \tag{16.10}$$

At 30°C:
$$t_{\max}(30) = \frac{d}{v_s(30)} = \frac{0.72}{349.48} \approx 0.002060\,\text{s} \tag{16.11}$$

Difference in samples

At 20°C:
$$N_{\max}(20) = \frac{t_{\max}(20)}{T_s} = \frac{0.002097}{6.25 \times 10^{-5}} \approx 33.55\,\text{samples} \tag{16.12}$$

At 30°C:
$$N_{\max}(30) = \frac{t_{\max}(30)}{T_s} = \frac{0.002060}{6.25 \times 10^{-5}} \approx 32.96\,\text{samples} \tag{16.13}$$

We saw from [eq. (16.13)] that higher temperatures caused the maximum time delay to decrease, the method of limiting the correlation to a certain cutoff point would therefore not affect higher temperatures.

From [eq. (16.9)] one could see that the speed of sound decreases with lower temperatures, which we had to consider. The solution in the code was to add four extra samples of delay to the threshold which helped us maintain some tolerance for different temperatures and other variables affecting the sampling [Code: 23].

```python
speed_of_sound = 343  # Speed of sound in m/s at 20C sea level
max_mic_distance = 0.72  # The max distance between microphones
max_time_delay = max_mic_distance / speed_of_sound
time_per_sample = 1/fs_local
max_sample_delay = (max_time_delay//time_per_sample) + 4  # Adding +4 for some
↪ tolerance.

realistic_array = array_padded  # Copy of array to modify
realistic_array[max_sample_delay:-max_sample_delay] = 0  # Setting the
↪ unrealistic delays to 0.
```

Code 23: Narrowing the cross correlation.

16.7 Position calculations/estimation

16.7.1 Angle estimates using TDOA

Angle was calculated using [eq. (16.14)] taken from robust sound source localization [17], and was found using the unit vector $\vec{u}$. The microphone position $(\vec{x_{ij}})$ and the corresponding time delay $(c\Delta T)$ were then used for solving the equation.

$$\vec{u} \cdot \vec{x}_{ij} = c\Delta T \tag{16.14}$$

However, the system used in practice deviated in multiple ways from the model presented in robust sound source localization [17].
The first deviation came in the form of dimensions of microphone position $(\vec{x_{ij}})$. The original equation [eq. (12.7)] was defined for three-dimensional estimates. That for stability reasons results in the need for microphones placed in three dimensions [17]. However, the model used did not need height, since the height of the fog horn direction would be considered less relevant. This also made the physical build easier to design. The model used was a matrix with two dimensions [eq. (16.15)].

$$\begin{bmatrix} (x_0 - x_1), & (y_0 - y_1) \\ (x_0 - x_2), & (y_0 - y_2) \\ (x_0 - x_3), & (y_0 - y_3) \\ (x_1 - x_2), & (y_1 - y_2) \\ (x_1 - x_3), & (y_1 - y_3) \\ (x_2 - x_3), & (y_2 - y_3) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} c\Delta T_{01} \\ c\Delta T_{02} \\ c\Delta T_{03} \\ c\Delta T_{12} \\ c\Delta T_{13} \\ c\Delta T_{23} \end{bmatrix} \tag{16.15}$$

The second deviation, also visible in [eq. (16.15)], involved the intervals for time delay estimates. The original equation [eq. (12.7)] has all estimates relative to microphone zero, while the system in practice used all possible combinations.

The deviations in format were re-evaluated with the same criteria set by robust sound source localization [17]. The stability of the system had to be kept, which was decided by the microphone positions. There had to be at least one microphone on each axis in the dimension the estimates were to be done in [eq. (16.16)]. Meaning in the case of four microphones, a maximum of three microphones could be on a single line. Resulting in the layout in [Fig: 16.3].

$$
\begin{aligned}
2 = rank(pos_a) = rank&\left(\begin{bmatrix} 0, & -0.36 \\ -0.36, & 0 \\ 0, & 0.36 \\ 0.36, & 0 \end{bmatrix}\right) \\
1 = rank(pos_b) = rank&\left(\begin{bmatrix} -0.10, & 0 \\ -0.36, & 0 \\ 0.10, & 0 \\ 0.36, & 0 \end{bmatrix}\right) \\
&\Downarrow \\
2 = rank(X_a) = rank&\left(\begin{bmatrix} (x_0 - x_1), & (y_0 - y_1) \\ (x_0 - x_2), & (y_0 - y_2) \\ (x_0 - x_3), & (y_0 - y_3) \\ (x_1 - x_2), & (y_1 - y_2) \\ (x_1 - x_3), & (y_1 - y_3) \\ (x_2 - x_3), & (y_2 - y_3) \end{bmatrix}\right) \\
1 = rank(X_b) = rank&\left(\begin{bmatrix} (x_0 - x_1), & (y_0 - y_1) \\ (x_0 - x_2), & (y_0 - y_2) \\ (x_0 - x_3), & (y_0 - y_3) \\ (x_1 - x_2), & (y_1 - y_2) \\ (x_1 - x_3), & (y_1 - y_3) \\ (x_2 - x_3), & (y_2 - y_3) \end{bmatrix}\right)
\end{aligned}
\tag{16.16}
$$

The microphone's position relative to the array center was put into a matrix ($\vec{x_{ij}}$), while their corresponding time delay was put into a vector ($c\Delta T$). As shown in [eq. (16.15)].

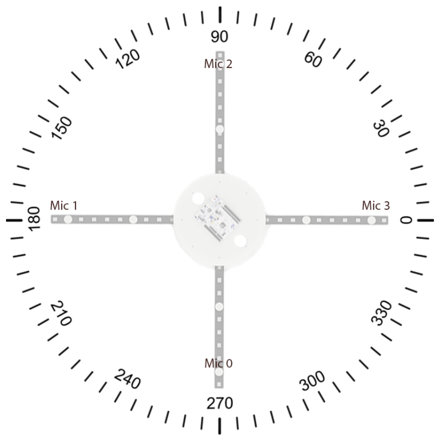


Figure 16.3: Relative placement

The same points mentioned in the article are relevant to this matrix as well. The system would get over-constrained given the amount of time delay intervals that were used. Robust sound source localization would solve this using the pseudo inverse [17]. However, the model used in practice is Numpy's least squares method. Reason being user preference.

All steps of angle estimates were implemented in Python. With the function defined in 16.14, but represented generally as [eq. (16.17)].

$$Ax = b \tag{16.17}$$

The general form was implemented in Python with two sets of parameters. Distance (b) calculated from time delay multiplied with the speed of sound, and the microphone's position relative to each other (A) [Code: 24].

```python
# time delay multiplied with speed of sound
b = np.array([
    [Dist_r01, Dist_r02, Dist_r03, Dist_r12, Dist_r13, Dist_r23]
])

# transposed for correct format
b = np.transpose(b)

# Distance between each microphone
A = np.array([
    Coord0 - Coord1,
    Coord0 - Coord2,
    Coord0 - Coord3,
    Coord1 - Coord2,
    Coord1 - Coord3,
    Coord2 - Coord3
])

# The directional vector (u) found from A and b
u, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
```

Code 24: Estimating directional vector ($\vec{u}$)

Finding the directional vector's angle was the last step for the angular estimation [Code: 25].

```python
# Angle used as estimate
angle = round(math.atan2(direction[1][0], direction[0][0]) * 180 / np.pi, 1)
```

Code 25: Finding angle of directional vector ($\vec{u}$)

16.7.2 Distance calculations/estimation

The distance calculation was crude and rudimentary and it was implemented right before the supervisor told us to scrap distance functionality and to focus more on robust angle estimation. The implementation came from a hunch to try and find the distance by minimizing an objective function using the least squares method which the group remembered from the cybernetics course AIS2202 [44].

Firstly, we created a function to calculate theoretical delays from each microphone [Code: 26]. It found the distance from the sound source to the microphones and returned the theoretical time delay, assuming perfect sound propagation.

```python
from scipy.optimize import minimize

speed_of_sound = 343  # Speed of sound in m/s

# Function for calculating the theoretical delays
def theoretical_delays(x, y, mic_positions):
    source_pos = np.array([x, y])  # Cartesian position of sound source.
    ↪ Estimated
    distances = np.linalg.norm(mic_positions - source_pos, axis=1)  # Finding
    ↪ euclidian distance between mics and sound
    return distances / speed_of_sound # Return time delay.
```

Code 26: Helper function, generate theoretical delays for RSS.

The previous time delay function [Code: 26] was then used within an objective function to find the error between the theoretical delays and the measured delays. This function found the pair-wise differences between the microphones. The theoretical differences were then compared to the measured differences, and the residual sum of squares (RSS) was calculated to represent the error [Code: 27].

```python
# Objective function to minimize for finding solution
def objective_function(params, mic_positions, measured_delays):
    x, y = params  # Location of sound source
    calc_delays = theoretical_delays(x, y, mic_positions)

    # Calculate the difference of the theoretical delays
    delay_differences = np.array([
        calc_delays[0] - calc_delays[1],
        calc_delays[0] - calc_delays[2],
        calc_delays[0] - calc_delays[3],
        calc_delays[1] - calc_delays[2],
        calc_delays[1] - calc_delays[3],
        calc_delays[2] - calc_delays[3]
    ])
    return np.sum((delay_differences - measured_delays)**2)  # Return sum of
    ↪ squared error
```

Code 27: Calculates RSS between theoretical and measured delay differences.

The final part was the main function [Code: 28], this function contained the initial guess which started at [0,0]. The initial guess was required to have a starting point for Scipy's minimize function, which it then used with the least squares method to find the guess with the least error inside the objective function. The guess with the least error was extracted and the distance was then found by using the Pythagorean theorem. The direction in this case was not of interest as the angle was being calculated separately in another function.

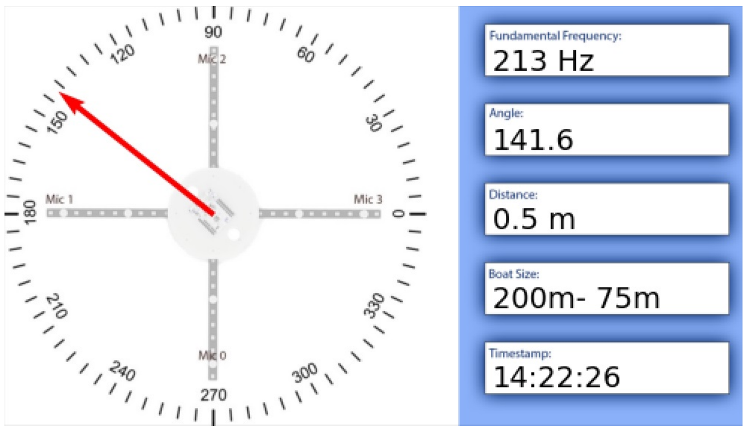
```python
def estimate_source_position(mic_positions, measured_delays):
    initial_guess = [0, 0]
    result = minimize(objective_function, initial_guess, args=(mic_positions,
    ↪  measured_delays))
    x, y = result.x  # Extract optimized parameters
    distance = np.sqrt(x**2 + y**2)  # Pythagoras to find distance
    return distance, x, y
```

Code 28: Main function to estimate source position.

The distance estimation was quickly deemed unreliable and further tuning, testing, and implementation of this was scrapped. This functionality was therefore not in a state where we could claim that the system is capable of estimating distance to the source.

16.8 GUI

A GUI was developed in tandem with OpenAI's GPT-4 [42] to present the computational results in an accessible manner [Fig: 16.4]. The GUI was split into two parts, the first part was a visual representation of the physical system with a vector plotting the direction of the sound source. The vector was scaled to always be the same size so the direction would be the only displayed information in this part of the GUI. The second part of the GUI contained the computed results. This included: fundamental frequency from the sound, angle to the source, distance to the source, estimated boat size from the frequency, and the timestamp at the recorded time window.



Fundamental Frequency:
213 Hz

Angle:
141.6

Distance:
0.5 m

Boat Size:
200m- 75m

Timestamp:
14:22:26

Figure 16.4: GUI Example

The GUI would be repeatedly updated in the main thread of the program, this allowed the program to run continuously with the plot constantly updating to represent the newest time window. Given the theme and scope of the project, the GUI is of less interest to the report, and the full code as well as the Chat-GPT4 prompts for the interface is included in the attachments.

Part V

Results

Chapter 17

Simulator

17.1 The accuracy of the model

17.1.1 Accuracy of angle estimation

The mathematical model from [Sec. 16.7.1] was tested using the simulator. They were carried out by directly using the time delays from [Sec. 14.4]. This minimized the error caused by delay estimates and allowed testing of the mathematical model alone.

The testing was done by moving the source in a circular motion in steps of one degree around the array center. There was done five tests in total. The radii of the circles were 4m, 6m, 12m, 24m and 3700m. The model's accuracy was then measured as the error between the target and the estimated angle. Represented in [Tab: 17.1], as the mean and standard deviation of the whole circular motion.

The table shows the model struggling at lower distances, which occurs in the angles between the axes spaced 45 degrees [Tab: 17.2]. With the pattern repeating for all quadrants. The inaccuracies do not occur for longer distances.

Distance	Mean (deg)	standard deviation (deg)
2m	0.137777778	0.074006365
4m	0.033333333	0.047206062
6m	0	0
12m	0	0
24m	0	0
3700m	0	0

Table 17.1: Error of model

Target Angle	Estimated Angle	Error
0.0	0.0	0.0
1.0	1.0	0.0
2.0	2.0	0.0
3.0	3.0	0.0
4.0	3.9	0.1
5.0	4.9	0.1
6.0	5.9	0.1
7.0	6.9	0.1
8.0	7.9	0.1
9.0	8.9	0.1
10.0	9.9	0.1
11.0	10.8	0.2
12.0	11.8	0.2
13.0	12.8	0.2
14.0	13.8	0.2
15.0	14.8	0.2
16.0	15.8	0.2
17.0	16.8	0.2
18.0	17.8	0.2
19.0	18.8	0.2
20.0	19.8	0.2
21.0	20.8	0.2
22.0	21.8	0.2
23.0	22.8	0.2
24.0	23.8	0.2
25.0	24.8	0.2
26.0	25.8	0.2
27.0	26.8	0.2
28.0	27.8	0.2
29.0	28.8	0.2
30.0	29.8	0.2
31.0	30.8	0.2
32.0	31.8	0.2
33.0	32.8	0.2
34.0	33.8	0.2
35.0	34.9	0.1
36.0	35.9	0.1
37.0	36.9	0.1
38.0	37.9	0.1
39.0	38.9	0.1
40.0	39.0	0.1
41.0	40.9	0.1
42.0	42.0	0.0
43.0	43.0	0.0
44.0	44.0	0.0
45.0	45.0	0.0

Table 17.2: Mathematical model simulated (2m)

17.1.2 Accuracy of time delay estimation

The cross-correlation with weighting was tested to confirm that the model functioned as intended. It used the same methods as angle estimation. A sample was virtually played in a circle around the center of the microphone, in steps of one degree. This time it was only carried out a test at a radius of 2m.

The weighing coefficients used were $\alpha = 0.4$ and $\gamma = 0.3$, the same coefficients used in robust sound source localization [17]. The reason was that the signal was not noisy enough to require more aggressive filtering. The accuracy for each cross-correlation was based on the collective error of all angles in the circular motion.

The results show that the cross-correlations work with small errors across all intervals [Tab: 17.3]. The spread of inaccuracy for all intervals loosely follows the pattern in [Tab: 17.4]. With accurate results along axes spaced 45 degrees, and varying results in between. Max error never exceeds $\pm 1.965E-5$ for the whole data set.

Cross-correlation	Mean error (s)	Standard deviation (s)
01	-1.7863E-19	1.01396E-05
02	-1.24984E-19	1.01055E-05
03	-5.47748E-20	1.01396E-05
12	1.12561E-19	1.01396E-05
13	4.14105E-20	1.01055E-05
23	1.38537E-19	1.01396E-05

Table 17.3: Error of cross correlations

Target Angle	Estimated Angle	Error	Target Delay	Estiamted Delay	Error
0.0	0.0	0.0	101.81E-05	102.08E-05	-2.73E-06
5.0	5.1	-0.1	92.33E-05	91.67E-05	6.59E-06
10.0	10.3	-0.3	82.2E-05	81.25E-05	9.56E-06
15.0	15.0	0.0	71.52E-05	70.83E-05	6.91E-06
20.0	19.7	0.3	60.36E-05	60.42E-05	-6.09E-07
25.0	24.5	0.5	48.78E-05	50.0E-05	-1.22E-05
30.0	29.6	0.4	36.87E-05	37.5E-05	-6.26E-06
35.0	35.3	-0.3	24.72E-05	22.92E-05	1.8E-05
40.0	40.2	-0.2	12.4E-05	12.5E-05	-9.83E-07
45.0	45.0	0.0	0.0	0.0	0.0

Table 17.4: Accuracy cross-correlation simulator microphone 0 and 1

17.1.3 Accuracy of the model combined

The tests for the whole model were done in two parts. First stress tests at set distances then targeted tests. The stress test was meant as a control, to check that not any anomalies occur. The targeted test was meant for direct comparisons with the real-world system 18.

Stress tests

The stress tests were carried out by moving a sound source in a circular motion three times, giving each angle multiple entries to affect the result. The radius was constant, and the step size was five degrees. Volume was not adjusted for longer distances, assuming ideal conditions, with the microphones hearing a clear signal at all times. Tests were done at 2m, 4m, 6m, 12m, 24m, 1850m and 3700m. It is important to note that the test at 3700m broke the simulator. The recording window was smaller than the time the sound used to arrive at the array, making the tests only record white noise. It would be treated as an outlier.

The estimates fall into the same pattern as mentioned in [Sec. 17.1.2]. With estimates spaced 45 degrees having the highest accuracy, and varying results in between. The error is small for all tests until 1850m [Sec. 17.5]. The errors never exceed 5.8979E-05s and -3.70971E-05s for all cross-correlations until 24m and 0.9°and -1.2°for angular errors. However, they get noticeably larger when the distance becomes 1850m. The largest errors become 1.82951E-04s and -1.29384E-04s for cross-correlations, with 3.2°and -2.9°for angular errors. The results at 3700m are only useful for demonstrating a weakness in the simulator.

Distance	Mean error (deg)	Standard deviation (deg)
2m	-0.057407407	0.354952654
4m	0.019907407	0.412391765
6m	-1.21302E-1	0.391538409
12m	-0.023148148	0.364623152
24m	0.025	0.387868355
1850	0.033333333	1.253014969
3700	-0.235185185	148.9170462

Table 17.5: Angular error stress test

Targeted tests

The targeted tests were carried out by setting the sound source position at a fixed point, with all parameters constant. The scenario was repeated 72 times. They were meant to address the issues of the blast being cut, false peaks, false instances of zero delay, and the issues at 90 degrees [Sec. 18]. The sound clip was scaled to the same magnitude as the real-world samples, and white noise was also added to match. The tests locked the angle at 45 degrees and ran loops of three scenarios with randomized noise [Tab: 17.6], [Tab: 17.7] and [Tab: 17.8].

The spread of the data is more random than [Tab: 17.4], but the tables show that the system works in all instances with small errors. The errors never exceed 4.86E-05s and -4.54E-5s for time delay and 0.9°and -0.7°for angular error.

Cross-correlation	Mean error (s)	Standard deviation (s)
01	0	1.60234E-05
02	8.66579E-06	1.43104E-05
03	8.08708E-06	1.51966E-05
12	8.66579E-06	1.70403E-05
13	9.24449E-06	1.42271E-05
23	8.68056E-07	1.46011E-05

Table 17.6: Error cross-correlation whole blast (2m and 45 degrees)

Cross-correlation	Mean error (s)	Standard deviation (s)
01	3.14941E-07	4.65035E-06
02	1.05603E-06	5.34472E-06
03	1.68591E-06	6.59186E-06
12	-2.03734E-07	7.36992E-06
13	2.31579E-06	7.58515E-06
23	9.44822E-07	5.94182E-06

Table 17.7: Error cross-correlation beginning of blast cut (2m and 45 degrees)

Cross-correlation	Mean error (s)	Standard deviation (s)
01	-1.25976E-06	1.60968E-05
02	-2.72326E-06	1.64307E-05
03	-5.87267E-06	1.86446E-05
12	-3.98302E-06	1.93124E-05
13	-4.92785E-06	1.72053E-05
23	0	0

Table 17.8: Error cross-correlation end of blast cut (2m and 45 degrees)

Sample type	Mean error (deg)	Standard deviation (deg)
Full blast	0.013888889	0.320784933
Beginning of blast cut	0.016666667	0.093447281
End of blast cut	-0.008333333	0.266828003

Table 17.9: Angular error

The tests regarding the physical systems blind spot were repeated the same way as the first test. By locking the angle at 90 degrees, and running 72 loops of the same scenario.

The errors of all cross-correlations never exceed 5.9937E-05s and -5.9937E-05s, and the errors of all angular estimates never exceed 0.6°and -0.9°. It can be noted that the performance of angular estimates of 4m is slightly worse.

Cross-correlation	Mean error (s)	Standard deviation (s)
01	4.52848E-07	1.56995E-05
02	-1.66601E-05	1.37383E-05
03	1.63496E-07	1.33401E-05
12	-2.02958E-05	1.53052E-05
13	-8.68056E-07	1.61894E-05
23	1.76916E-05	1.4405E-05

Table 17.10: Error cross-correlation whole blast (2m and 90 degrees)

Cross-correlation	Mean error (s)	Standard deviation (s)
01	2.47831E-06	1.64005E-05
02	-1.95536E-05	1.61474E-05
03	5.08248E-06	1.56128E-05
12	-1.62449E-05	1.53052E-05
13	2.89352E-06	1.57562E-05
23	1.82703E-05	1.67691E-05

Table 17.11: Error cross-correlation whole blast (4m and 90 degrees)

Distance	Mean error	Standard deviation
2m	-0.033333333	0.266959933
4m	0.073611111	0.334408477

Table 17.12: Angular error 90 degrees

Chapter 18

Real world results

18.1 The accuracy of the model

A degree disk [Sec. 15.1] and an adapter that could hold a piece of string at the center, was used to test the model. The string in combination with the degree disk was used to find the angle and distance that the speaker was placed at.

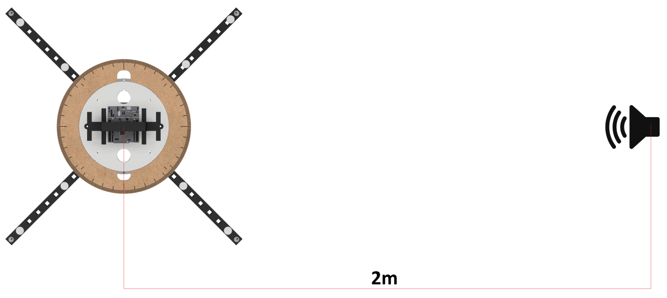
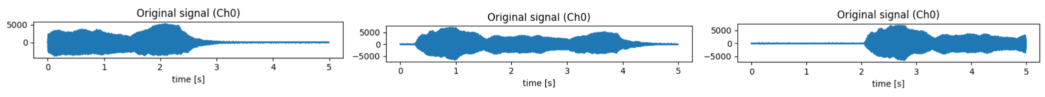


Figure 18.1: Test setup illustrated

There were markers placed every meter on the string to measure the distance between the rig and the speaker. This was essential to ensure repeatable test scenarios over multiple days.

18.1.1 Accuracy of angle estimation

The most important result revolving accuracy of the angle estimates in the physical system is the fact that they are highly dependent on how the fog horn is recorded [Fig: 18.2].



(a) Beginning of blast cut (b) Whole blast (c) End of blast cut

Figure 18.2: Singular fog horn blast with recording time of 5 seconds

Only when a whole blast is included in the recording window, will the model reliably find the right angle. This is visible when comparing angular estimates between different parts of the blast recorded [Tab: 18.1].

But the system shows better results when the right conditions are met. It will find the correct angle the majority of the time as long as the whole blast is present. But some miscalculations are still present even then. This is visible when testing for both two and four meters. [Tab: 18.2]. The angular errors for this data set would be $\sigma = 68.6^{\circ}(Withoutliers)$ and $\sigma = 0.53^{\circ}(Withoutoutliers)$.

Another result worth noting is the system accuracy when the target angle is at 90 degrees. It works well when the speaker is at two meters but struggles at four meters. This is visible in [Tab: 18.3]

Test ID	Blast part	Distance (m)	Target angle (deg)	Measured angle (deg)
06#18#1	whole blast	2	45	44.4
06#18#2	whole blast	2	45	44.4
06#18#3	whole blast	2	45	45
06#18#4	whole blast	2	45	43.4
06#18#5	whole blast	2	45	45
06#22#1	Beginning of blast cut	2	45	2.4
06#22#2	Beginning of blast cut	2	45	-92.9
06#22#3	Beginning of blast cut	2	45	81.9
06#22#4	Beginning of blast cut	2	45	-140.4
06#22#5	Beginning of blast cut	2	45	-146.3
06#23#1	End of blast cut	2	45	31
06#23#2	End of blast cut	2	45	45
06#23#3	End of blast cut	2	45	-45
06#23#4	End of blast cut	2	45	84.5
06#23#5	End of blast cut	2	45	-90

Table 18.1: Angle estimates based off recordings depicted in [Fig: 18.2b], [Fig: 18.2a] and [Fig: 18.2c]

Test ID	Blast part	Distance (m)	Target angle (deg)	Measured angle (deg)
06#18#1	whole blast	2	45	44.4
06#18#2	whole blast	2	45	44.4
06#18#3	whole blast	2	45	45
06#18#4	whole blast	2	45	43.4
06#18#5	whole blast	2	45	45
06#19#1	whole blast	2	45	43.8
06#19#2	whole blast	2	45	17.9
06#19#3	whole blast	2	45	44.3
06#19#4	whole blast	2	45	44.4
06#19#5	whole blast	2	45	104.7
09#15#1	whole blast	4	45	45
09#15#2	whole blast	4	45	-180
09#15#3	whole blast	4	45	45
09#15#4	whole blast	4	45	45
09#15#5	whole blast	4	45	45

Table 18.2: Angle estimates based off recordings depicted in [Fig: 18.2b]

Test ID	Blast part	Distance (m)	Target angle (deg)	Measured angle (deg)
09#10#1	whole blast	2	90	90.4
09#10#2	whole blast	2	90	89.6
09#10#3	whole blast	2	90	89.2
09#10#4	whole blast	2	90	89.6
09#10#5	whole blast	2	90	89.2
09#10#1	whole blast	4	90	142.1
09#10#2	whole blast	4	90	144.2
09#10#3	whole blast	4	90	144.2
09#10#4	whole blast	4	90	-126.9
09#10#5	whole blast	4	90	144.8

Table 18.3: Angle estimates based off recordings depicted in [Fig: 18.2b]

18.1.2 TDOA estimates accuracy

The model's ability to measure the correct angle is largely dependent on the TDOA estimates. Error in angle estimates is therefore also visible when analyzing the cross-correlation plots. The issue takes two forms. Some cases show how false peaks occur at the wrong time delay, while other times the delay is falsely set to zero.

This is visible when comparing the cases shown in [Fig: 18.4]. Cross-correlation 0_1 and 2_3 should be zero, while the rest should have the same delay given the target angle of 45 degrees [Fig: 18.3]. The estimate in [Fig: 18.4a] is correct for these reasons.
Similarly, it can be said for why [Fig: 18.4b] and [Fig: 18.4c] is wrong. Both false peaks and false zeroes occur multiple times throughout testing, and [Fig: 18.4] is selected as examples to illustrate this.

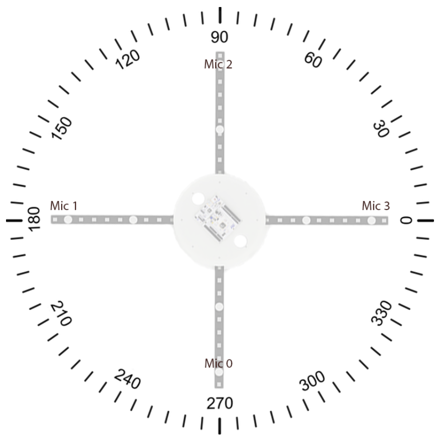
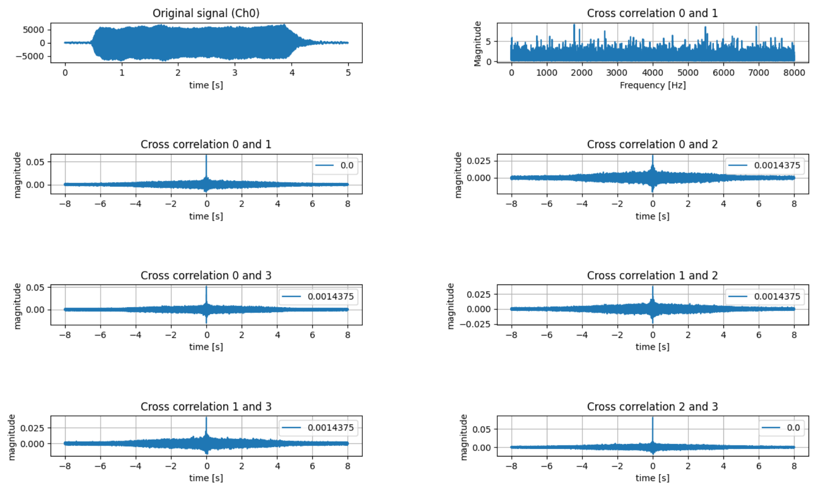
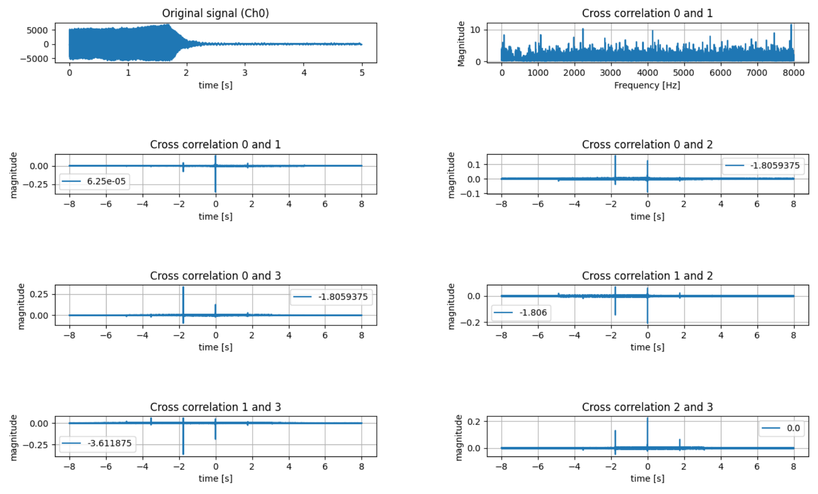


Figure 18.3: Microphone placement

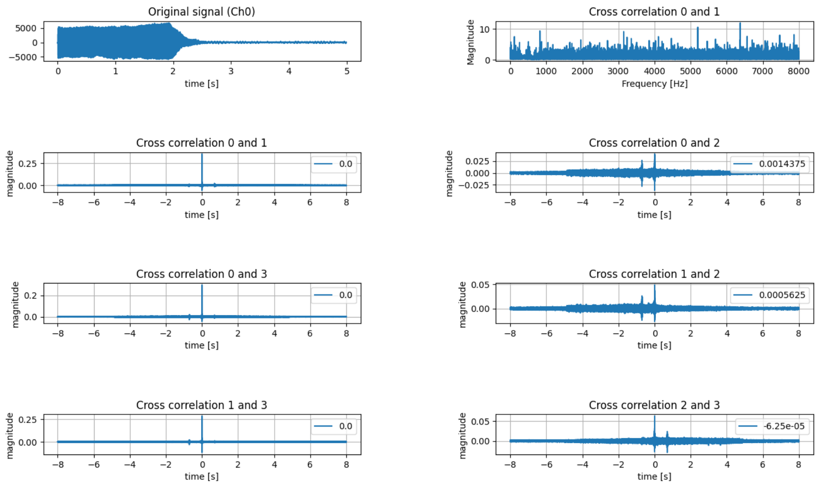
An attempt at removing the issues of radical false peaks was made by narrowing the search area as mentioned in [Sec. 16.6.3]. This worked well for the false peaks, however, this was not enough to compensate for the false zeroes issue. The correlations in [Fig: 18.5] should show 90 degrees, meaning only microphone 1_3 should be set to zero, as shown in [Fig: 18.5a]. But both [Fig: 18.5b] and [Fig: 18.5c] show cross-correlations other than this peaking at zero.



(a) Test ID: 06#18#3 (Correct estimate)



(b) Test ID: 06#22#5 (False peaks)

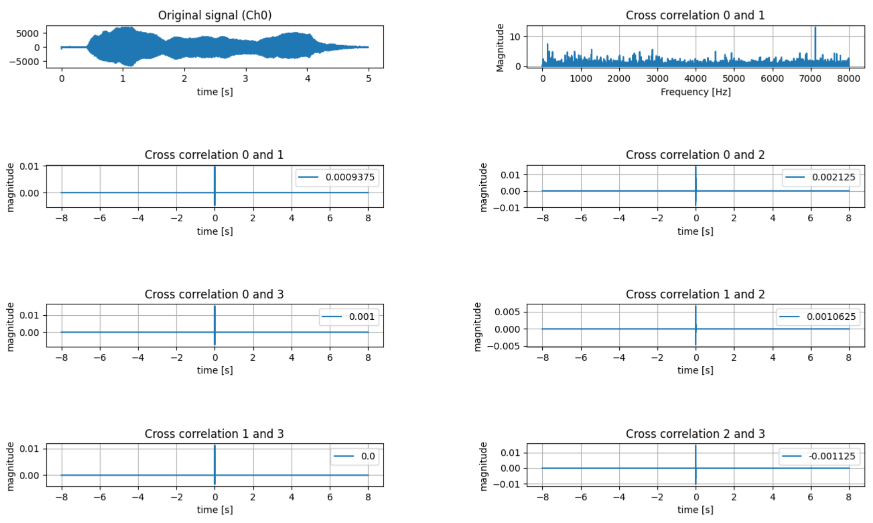


(c) Test ID: 06#22#3 (False delay of zero)

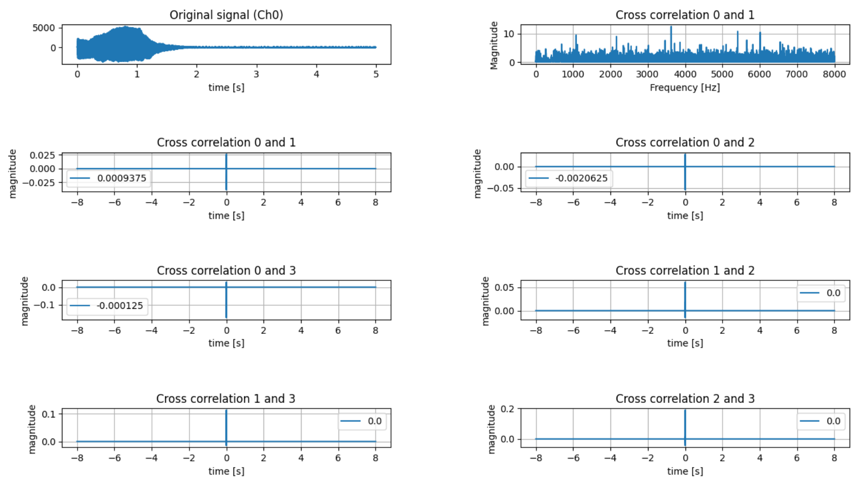
Figure 18.4: Cross correlations

Test ID	Blast part	Distance (m)	Target angle (deg)	Measured angle (deg)
09#5#1	whole blast	2	90	89.2
09#5#2	whole blast	2	90	88.7
09#5#3	whole blast	2	90	90.8
09#5#4	whole blast	2	90	90
09#5#5	whole blast	2	90	89.6
09#4#1	Beginning of blast cut	2	90	-127.9
09#4#2	Beginning of blast cut	2	90	n/a
09#4#3	Beginning of blast cut	2	90	n/a
09#4#4	Beginning of blast cut	2	90	n/a
09#4#5	Beginning of blast cut	2	90	103
09#6#1	End of blast cut	2	90	n/a
09#6#2	End of blast cut	2	90	-105.1
09#6#3	End of blast cut	2	90	87.1
09#6#4	End of blast cut	2	90	-71.6
09#6#5	End of blast cut	2	90	n/a

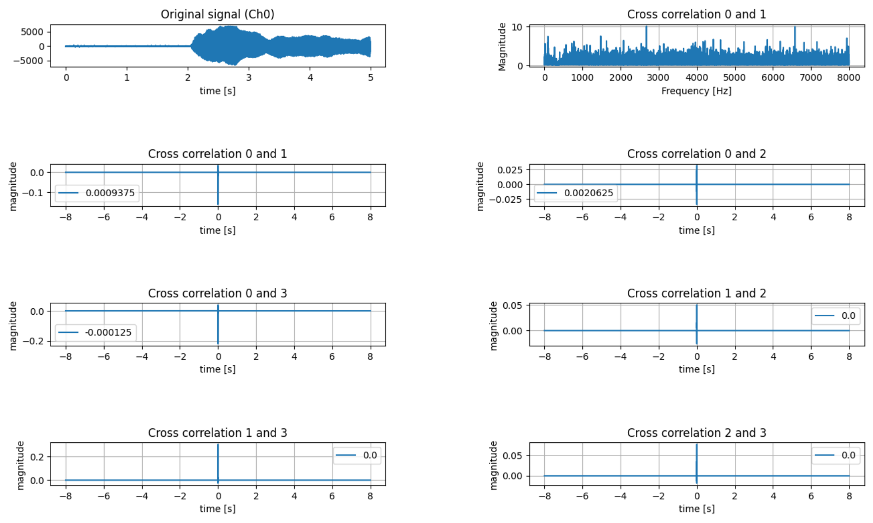
Table 18.4: Angle estimates based off recordings depicted in [Fig: 18.2b], [Fig: 18.2a] and [Fig: 18.2c]



(a) Test ID: 09#5#4 (Correct estimate)



(b) Test ID: 09#4#2 (False delay at zero)



(c) Test ID: 09#6#1 (False delay at zero)

Figure 18.5: Cross correlations

18.1.3 Accuracy of distance estimation

The distance estimation seems to fall into false solutions, which gives us a wrong estimate, even when tested at different distances and angles. This tendency to fall into the wrong estimate is represented in all of our tests and is included in the full test document.

Test ID	Blast part	Distance (m)	Angle (degrees)	Estimated distance (m)
06#15#1	Whole blast	2	90	3.8
06#15#2	Whole blast	2	90	0.4
06#15#3	Whole blast	2	90	1.3
06#15#4	Whole blast	2	90	0.3
06#15#5	Whole blast	2	90	1.9
09#1#1	Beginning of blast cut	2	90	0.3
09#1#2	Beginning of blast cut	2	90	0.2
09#1#3	Beginning of blast cut	2	90	0.3
09#1#4	Beginning of blast cut	2	90	1.3
09#1#5	Beginning of blast cut	2	90	0.3
09#13#1	Whole blast	4	90	0.3
09#13#2	Whole blast	4	90	0.3
09#13#3	Whole blast	4	90	0.3
09#13#4	Whole blast	4	90	0.3
09#13#5	Whole blast	4	90	0.3
09#15#1	End of blast cut	4	45	0.2
09#15#2	End of blast cut	4	45	0.4
09#15#3	End of blast cut	4	45	0.3
09#15#4	End of blast cut	4	45	0
09#15#5	End of blast cut	4	45	0.3

Table 18.5: Test data showcasing the distance estimation

18.2 HPS

HPS was tested in parallel with the angular, cross-correlation, and distance estimates. The method gave mixed results. It varied between defaulting to zero for a whole test [Tab: 18.7] to getting the frequency right for a whole test [Tab: 18.6], and everything in between [Tab: 18.8].

However, it is interesting to see that the algorithm at times finds spectral copies of a frequency. The result sometimes shows 84Hz, something that is interpreted as a spectral copy of 168Hz.

These results show that the algorithm works as intended but with mixed results. The tables are outtakes meant to represent a larger database.

Test ID	Target frequency (Hz)	Estimated frequency (Hz)
09#13#1	168	168
09#13#2	168	168
09#13#3	168	168
09#13#4	168	168
09#13#5	168	168

Table 18.6: Successful attempt

Test ID	Target frequency (Hz)	Estimated frequency (Hz)
09#6#1	168	0
09#6#2	168	0
09#6#3	168	0
09#6#4	168	0
09#6#5	168	0

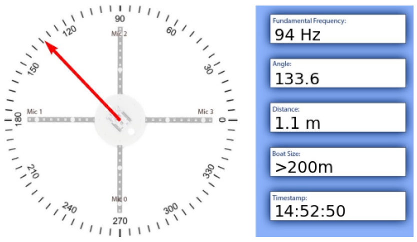
Table 18.7: Failed attempt

Test ID	Target frequency (Hz)	Estimated frequency (Hz)
06#11#1	168	112
06#11#2	168	112
06#11#3	168	169
06#11#4	168	84
06#11#5	168	84

Table 18.8: Mixed attempt

All of the figures above are examples of show how the HPS algorithm performs. It either defaulted to zero many times, found the spectral copy of the frequency of interest or got it wrong. All the tests was done with the frequency validation, setting the result to zero whenever it is unsure of the correct frequency. This explains why the HPS defaults to zero for many of the tests. This can be adjusted by changing the maximum threshold on the deviation allowed, but at a trade off for accuracy.

18.3 GUI and classification



Fundamental Frequency:
94 Hz

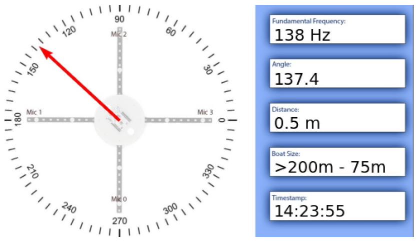
Angle:
133.6

Distance:
1.1 m

Boat Size:
>200m

Timestamp:
14:52:50

(a) HPS between 70-130Hz



Fundamental Frequency:
138 Hz

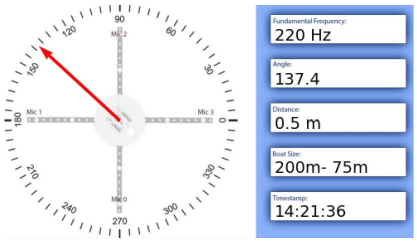
Angle:
137.4

Distance:
0.5 m

Boat Size:
>200m - 75m

Timestamp:
14:23:55

(b) HPS between 130-200Hz



Fundamental Frequency:
220 Hz

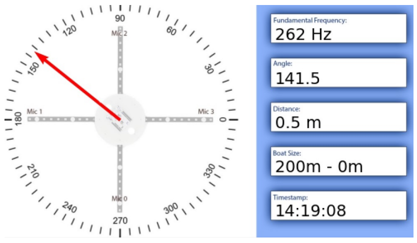
Angle:
137.4

Distance:
0.5 m

Boat Size:
200m- 75m

Timestamp:
14:21:36

(c) HPS between 200-250Hz



Fundamental Frequency:
262 Hz

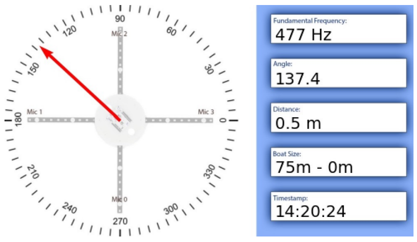
Angle:
141.5

Distance:
0.5 m

Boat Size:
200m - 0m

Timestamp:
14:19:08

(d) HPS between 250-350Hz



Fundamental Frequency:
477 Hz

Angle:
137.4

Distance:
0.5 m

Boat Size:
75m - 0m

Timestamp:
14:20:24

(e) HPS between 350-700Hz

Figure 18.6: Result of the HPS and ship classification

The figures show how the GUI represented values found in the system.

As shown in figure [Fig: 18.6a], the fundamental frequency is estimated at 94Hz. According to [32] this is the equivalent of a ship above 200 meters.

Figure [Fig: 18.6b] shows a fundamental frequency of 138Hz. This is between the 70Hz to 200Hz and 130Hz to 350Hz range, which means that the ship can be any size above 75 meters. [32]

The next figure [Fig: 18.6c] reports a fundamental frequency at 220Hz. This frequency is a part of the 130Hz to 350Hz range without overlapping with either the 70Hz to 200Hz and 250Hz to 700Hz range. That means that the ship's size is between 75 meters to 200 meters. [32]

Figure [Fig: 18.6d] is showing a fundamental frequency of 262Hz. This is between both the 130Hz to 350Hz range and the 250Hz to 700Hz range, which means that the ship's size can be between 0 meters to 200 meters. [32]

The last figure [Fig: 18.6e] shows a fundamental frequency at 477Hz. That frequency is only in the range of 250Hz to 700Hz and means that the ship is between 0 meters and 75 meters. [32]

Part VI

Discussion

Chapter 19

Hardware

19.1 Microcontroller

The sampling rate and transfer speed capabilities of the nucleo-F446RE are well within the project requirements. If a faster computation of data is required, one could write more lightweight and specialized software for microphone acquisition. It could also prove to be more efficient to process the digital signals in the microcontroller environment, but Python was chosen for this as the group felt more comfortable and experienced with Python.

19.2 CCA02M2

The expansion board CCA02M2 was utilized with the maximum amount of four microphones and there were no problems encountered with this component. The configuration outlined in the FP-AUD-SMARTMIC1 documentation helped us set up the soldering arrangement for the correct acquisition strategy. It's important to note that the maximum number of microphones streaming in real-time is four. If there is a need to expand upon the number of microphones, a second expansion board and microcontroller would be needed for this configuration.

19.3 Microphones

The STEVAL-Mic002V1 microphones fit the project requirements and there were no issues with this choice of microphone. However, further testing is required to explore the long-distance capabilities of this microphone model since the test framework of the project only encapsulated shorter-distance sound acquisition and distance estimation.

19.4 Jetson Nano

The Jetson Nano computer was used to run the main Python program. It has enough computing power to be sufficient for this project, and its requirement. It also has 4 USB ports allowing for the expansion of one more microphone array. Its low power draw, computational power, and the Linux operating system make it an excellent choice for the passive sensor.

Chapter 20

Model

20.1 Simulator

The simulator has the most prioritized functionality implemented, which is the angular estimates. But it still misses the HPS and distance estimation functionality. However, this is only a matter of transferring the code from the physical system.

20.2 Weighing

Tuning the coefficients in the spectral weighting function is not prioritized given how the test cases are conducted under a controlled environment. However, it will become more relevant as soon as the system is to be tested under realistic conditions.

20.3 Angle estimation

The angle estimation is largely dependent on the accuracy of the cross-correlation, as demonstrated by the tests done in the simulator and the physical system. The simulator shows how the mathematical model that calculates angle from time delays, works well for short distances, and even better for longer ones. However, the issues seem to not be entirely caused by the TDOA algorithm. Given that the simulator also gives good results with the whole system in use.

20.4 Time delay estimation

The most prominent issue is how the results vary depending on what part of the fog horn blast is recorded by the recording window [Fig: 18.1]. The cross-correlation gives false peaks and false zero delays when the recording window cuts the signal at the start or end. This in turn causes the cross-correlation to find false peaks or false zero delays. Something that does not occur in the simulator.

A reason for these issues could be the margin of error the TDOA algorithm has. The estimates may be unstable, with the highest peak in the frequency domain having close neighbors. The issue could occur only in the real physical system because it is unstable enough to expose these issues. And the simulator getting the estimates right because it is barely stable enough. However, there has not been any tests done where all cross-correlations have been analyzed in the frequency domain. So there is more testing that has to be done in this area.

The occurrence of these instances is not entirely isolated to when a part of the signal is cut. They also occur at times when the whole blast is recorded, although not as often.

Another reason for the issues regarding the real-world system could be in the zero padding. Maybe the target of doubling the signal length is too much, causing other issues together with the ones it solves.

However, the tests in the simulator show that the system should work in theory. Further testing and research would be useful and could lead to improvements down the line.

Chapter 21

Physical System

21.1 Distance estimation

The distance estimation of our model using the least squares method is non-functional. It is neither efficient nor accurate and can therefore not be trusted. The system has to be reliable for detecting ship traffic and the distance functionality is therefore not sufficient. However, there are ways to improve the least squares approach.

The initial guess of the function plays a vital part in the method's efficiency and accuracy. Starting at a more realistic point for the initial guess could prove to give more robustness against false solutions. A proposed idea for the initial guess is to use the detected angle to start the initial guess within the right quadrant and at a distance further out from the coordinate origin. This proposition was not implemented as the project was reaching its end and further development was halted to document the project.

Another option to estimate the distance would be to implement a beam-forming algorithm. However, the success rate of this is uncertain. The optimal option would be to add another module which is placed in a different area, and use trigonometry to calculate the distance. With the accuracy of our model, this would give a reliable estimation. New tests have to be carried out to test this. Since the original tests for the physical system only show the frequency domain cross-correlation for microphone zero and one.

21.2 HPS

The HPS algorithm used in this project partly worked. It delivered accurate frequencies when compared to the original audio source, but also missed many times.

The shortcoming of the HPS is the validation part. When it relies on more than one microphone to send an output. That is due to multiple reasons like external noise, internal noise, differences in build tolerance, translation layer from analog to digital signals, and more. A solution to fix these could be the use of different microphones with higher quality, the use of one microcontroller per microphone, or even just optimizing the controller code more.

Some other ways to estimate the ship's size based on the sound of a foghorn include Frequency analyses based on the Doppler shift effect. This approach would also estimate the speed of the ship. The use of machine learning is also possible. By training the machine on multiple foghorn sounds, and telling it to classify it based on the laws. This approach is more in-depth and requires the training of the machine learning algorithm.

Chapter 22

Goals

22.1 Modular system

The goal of creating a modular system would be considered successful in principle. The microphone array can be used as a USB microphone using two USB ports, meaning multiple arrays can be used on the same computer. Allowing for any processing that would be required, with as many microphones as every other USB interface on the computer.

But the goal is not yet physically realized. All testing has been done with a singular array, so the intricacies of the solution are not yet considered. But the framework is present.

22.2 Robust source localization

The goal of robust source localization was achieved in theory under ideal conditions. There is still a lot more to be done. This includes further testing with more noise. Noise like seagulls, idle boat sounds, and other maritime sounds. And further testing for the real-world system, to find the cause for its shortcomings.

22.3 Decrease the estimation of the angle compared to today's solution

This goal is not yet achieved. The physical system is not yet good enough to directly compare with other competitors. However, the group sees optimistically on the issue since the simulator gave good results.

22.4 The use of cheap and low-powered electronics

The electronics utilized in the project are considered cheap and low-powered compared to full-sized computers. Buying the individual components and configuring them for this application can therefore be considered cost-effective.

22.5 Estimate size of ship/vessel

This goal is partly realized. The testing showed mixed results, but the solution may not be far off in the future as further tuning to the frequency validation could increase the robustness and therefore make the estimation more reliable.

22.6 Produce a physical model that can be compared to a simulator

This would be considered a success. It has the most important functionality of estimating the source position using the same approach as the simulator, giving us the ability to compare them and find reasons for any discrepancies that occur.

Part VII

Conclusion

Chapter 23

Conclusion

In this project, a passive sensor was designed and built. It was made to estimate the angle that a sound comes from. It also estimates the size of the vessel depending on the fundamental frequency found. The housing was designed, manufactured, and built during the project, and was made to fit all of the electrical components used.

The sensor uses 4 microphones to find the angle in two dimensions (X, Y). These microphones sent their data through an expansion board to a microcontroller before being transferred to a mini-computer for audio processing.

After implementing the audio processing features, did the angle estimation have a theoretically high success rate. The simulated system has a standard deviation of up to $\sigma = 1.25$ while the physical model's deviation is roughly estimated at $\sigma = 68.6$ (with outliers), and a success rate of $\sigma = 0.53$ (with outliers removed). Although the real-life estimates are very limited in data quantity. The HPS algorithm's precision had a perceived moderate success rate before validation was implemented, and a lower success rate after.

With the developed model and the selected TDoA algorithm, distance estimation was not a trivial task. There was an attempt at adding this functionality using least squares, but the estimation proved to be crude and unreliable.

Chapter 24

Future work

Model

Future work for the model will include conducting more tests with more focus on the cross correlation's frequency domain. Comparing the differences in the simulator and the physical system. It will also be useful to do more tests on the spectral weighting coefficients. It will be interesting to see how far it is possible to push the filter and still get consistent results.

HPS

Future work for the HPS would include further reserch in tone detection.

Distance estimation

Implementing an accurate and trustworthy distance estimation. This can be done by using other algorithms, more equipment, or a combination of both.

Physical system

The expansion of the physical model to include more microphones, microcontrollers, and potentially a computer optimized for this application. By using two or more microphone arrays, it's possible to use triangulation between the two center points and their angle to find the distance to the object.

Real-world tests

More real-world tests. The system has been tested as a prototype with direction estimation as the main focus. The tests do not accurately represent the real-world situation of sea-faring vessels. Further testing with longer distances would therefore be required before the reliability of the system could be considered. The nature of the sound source is also beneficial to test further as the sound source used in our testing is a simulated fog horn using a speaker.

Reference list

[1] International Maritime Organization, *COLREG : Convention on the International Regulations for Preventing Collisions at Sea, 1972*, Consolidated ed. "4th ed., 2003". London: International Maritime Organization, 2003.

[2] *LocalizationTDOA/src/server_dgram/server.py at master · petrokn/LocalizationTDOA*. [Online]. Available: https://github.com/petrokn/LocalizationTDOA/blob/master/src/server_dgram/server.py (visited on 14th Mar. 2024).

[3] *Phontech P-8300 MkII Sound Reception Display Unit.* [Online]. Available: https://www.zenitel.com/product/p-8300-mkii-sound-reception-display-unit (visited on 14th May 2024).

[4] *P-8301 MkII Microphone/antenna for sound reception system.* [Online]. Available: https://www.zenitel.com/product/p-8301-mkii-microphone-unit (visited on 14th May 2024).

[5] NXP B.V 2022., 'I2S bus specification', en, vol. 2022, Feb. 2022. [Online]. Available: https://www.nxp.com/docs/en/user-manual/UM11732.pdf.

[6] F. Leens, *An introduction to I2C and SPI protocols*, Jan. 2009. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4762946.

[7] S. Campbell, *Basics of UART Communication*, en-US, Feb. 2016. [Online]. Available: https://www.circuitbasics.com/basics-uart-communication/ (visited on 30th Apr. 2024).

[8] *Adafruit PDM Microphone Breakout*, en-US. [Online]. Available: https://learn.adafruit.com/adafruit-pdm-microphone-breakout/overview (visited on 29th Apr. 2024).

[9] T. Kite, 'Understanding PDM Digital Audio', en,

[10] *Pulse Code Modulation - an overview — ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/engineering/pulse-code-modulation (visited on 29th Apr. 2024).

[11] Sustainability of Digital Formats: Planning for Library of Congress Collections, *PCM, Pulse Code Modulated Audio*, eng, web page, Apr. 2024. [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000016.shtml (visited on 29th Apr. 2024).

[12] Richard G. Lyons, *Understanding Digital Signal Processing*, 3rd Edition. Boston, MA 02116: Pearson Education, Inc, Jan. 2010, ISBN: 0-13-702741-9.

[13] Richard Baraniuk et al., *6.7: Gibbs Phenomena*, en, May 2020. [Online]. Available: https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Signal_Processing_and_Modeling/Signals_and_Systems_(Baraniuk_et_al.)/06%3A_Continuous_Time_Fourier_Series_(CTFS)/6.07%3A_Gibbs_Phenomena (visited on 2nd May 2024).

[14] *How do you identify the type of filter (low pass, high pass, or band pass) based on the transfer function?*, en. [Online]. Available: https://www.quora.com/How-do-you-identify-the-type-of-filter-low-pass-high-pass-or-band-pass-based-on-the-transfer-function (visited on 30th Apr. 2024).

[15] '13.6 Equalization', in *The Art of Sound Reproduction*, Waltham, Massachusetts, United States: Focal Press, 1998, ISBN: 978-0-240-51512-0.

[16] J. G. Proakis and D. G. Manolakis, *Digital signal processing*, 3rd Edition. Upper Saddle River. New Jersey 07458: Pretice-Hall, inc, 1996, ISBN: 0-13-158932-6.

[17] J.-M. Valin, F. Michaud, J. Rouat and D. Letourneau, 'Robust sound source localization using a microphone array on a mobile robot', in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, Oct. 2003, 1228–1233 vol.2. DOI: 10.1109/IROS.2003.1248813. [Online]. Available: https://ieeexplore.ieee.org/document/1248813 (visited on 15th Feb. 2024).

[18] *Fast Fourier Transform - an overview — ScienceDirect Topics*. [Online]. Available: https://www.sciencedirect.com/topics/engineering/fast-fourier-transform (visited on 30th Apr. 2024).

[19] *FFT*. [Online]. Available: https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft (visited on 30th Apr. 2024).

[20] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipiess in C: The art of scientific computing*, 2nd. New York, USA: Cambridge University Press, 2002, ISBN: 0 521 43108 5.

[21] *Figure 7. Comparison of the computational complexities of DFT, FFT and...* en. [Online]. Available: https://www.researchgate.net/figure/Comparison-of-the-computational-complexities-of-DFT-FFT-and-QFT-algorithms_fig2_338980039 (visited on 2nd May 2024).

[22] B. Delgutte and J. Greenberg, *THE DISCRETE FOURIER TRANSFORM*, English, 2005. [Online]. Available: https://web.mit.edu/~gari/teaching/6.555/lectures/ch_DFT.pdf#:~:text=URL%3A%20https%3A%2F%2Fweb.mit.edu%2F~gari%2Fteaching%2F6.555%2Flectures%2Fch_DFT.pdf%0AVisible%3A%200%25%20 (visited on 20th May 2024).

[23] B. Eriksson and J. Wikander, *Dynamics and motion control*, Aug. 2009. [Online]. Available: https://www.kth.se/social/upload/4f3119e2f27654646b00000a/L4_show.pdf%E2%80%8B.

[24] *Detecting and understanding correlations among data - ScienceDirect*. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/B9780323955768000064 (visited on 29th Apr. 2024).

[25] S. I. Julius O., *Mathematics of the Discrete Fourier Transform (DFT)*, 2nd Edition. Stanford, California 94305: Center for Computer Research in Music and Acoustics (CCRMA), Aug. 2002.

[26] A. V. Oppenheim and R. W. Schafler, *Discrete-time signal processing*, 2nd Edition. Upper Saddle River. New Jersey 07458: Pretice-Hall, inc, 1998, ISBN: 0-13-754920-2.

[27] *Passive geolocation and tracking of an unknown number of emitters — IEEE Journals & Magazine — IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/1642587 (visited on 4th Apr. 2024).

[28] *Object Tracking Using Time Difference of Arrival (TDOA) - MATLAB & Simulink - MathWorks Nordic*. [Online]. Available: https://se.mathworks.com/help/fusion/ug/object-tracking-using-time-difference-of-arrival.html (visited on 14th Mar. 2024).

[29] *Microphone Array Analysis Methods Using Cross-Correlations — IMECE — ASME Digital Collection*. [Online]. Available: https://asmedigitalcollection.asme.org/IMECE/proceedings/IMECE2009/43888/281/343968 (visited on 29th Apr. 2024).

[30] Norman R. Draper and Harry Smith, *Applied Regression Analysis*, 3rd Edition. New Jersey, USA, Apr. 1998, ISBN: 978-0-471-17082-2.

[31] P. Stoica and R. Moses, 'Chapter 1 & 2', in *Spectral Analysis of Signals*, Prentice Hall, Jan. 2005, ISBN: 978-0-13-113956-5.

[32] *Forskrift om forebygging av sammenstøt på sjøen (Sjøveisreglene) - Lovdata*. [Online]. Available: https://lovdata.no/dokument/SF/forskrift/1975-12-01-5 (visited on 8th Apr. 2024).

[33] *Fig.6. Pitch detection using Harmonic Product Spectrum HPS method...* en. [Online]. Available: https://www.researchgate.net/figure/Pitch-detection-using-Harmonic-Product-Spectrum-HPS-method-measures-the-maximum_fig5_336946611 (visited on 11th Apr. 2024).

[34] N. US Department of Commerce, *Speed of Sound Calculator*, EN-US, Publisher: NOAA's National Weather Service. [Online]. Available: https://www.weather.gov/epz/wxcalc_speedofsound (visited on 30th Apr. 2024).

[35] NTNU, *Emne - Signalbehandling - AIS2201 - NTNU*. [Online]. Available: https://www.ntnu.no/studier/emner/AIS2201#tab=omEmnet (visited on 3rd May 2024).

[36] *NUCLEO-F446RE - STM32 Nucleo-64 development board with STM32F446RE MCU, supports Arduino and ST morpho connectivity - STMicroelectronics*, en. [Online]. Available: https://www.st.com/en/evaluation-tools/nucleo-f446re.html (visited on 12th Apr. 2024).

[37] *STEVAL-MIC002V1 - Microphone coupon board based on the MP34DT06J digital MEMS - STMicroelectronics*, en. [Online]. Available: https://www.st.com/en/evaluation-tools/steval-mic002v1.html (visited on 12th Apr. 2024).

[38] *X-NUCLEO-CCA02M2 - Digital MEMS microphone expansion board based on MP34DT06J for STM32 Nucleo - STMicroelectronics*, en. [Online]. Available: https://www.st.com/en/ecosystems/x-nucleo-cca02m2.html (visited on 12th Apr. 2024).

[39] STMicroelectronics, *FP-AUD-SMARTMIC1 - STM32Cube function pack for MEMS microphone acquisition, advanced audio processing and audio output - STMicroelectronics*, en. [Online]. Available: https://www.st.com/en/embedded-software/fp-aud-smartmic1.html (visited on 15th Feb. 2024).

[40] STMicroelectronics, *STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics*, en. [Online]. Available: https://www.st.com/en/development-tools/stm32cubeide.html (visited on 7th May 2024).

[41] *Jetson Nano Developer Kit*, en-US. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit (visited on 12th Apr. 2024).

[42] *GPT-4*, en-US. [Online]. Available: https://openai.com/index/gpt-4/ (visited on 20th May 2024).

[43] lumenlearning.com, *17.2 Speed of Sound — University Physics Volume 1*. [Online]. Available: https://courses.lumenlearning.com/suny-osuniversityphysics/chapter/17-2-speed-of-sound/ (visited on 13th May 2024).

[44] *Course - Cybernetics - AIS2202 - NTNU*. [Online]. Available: https://www.ntnu.edu/studies/courses/AIS2202#tab=omEmnet (visited on 9th May 2024).