# Sound Source Distance Estimation Using Sound Frequency Attenuation

FRIDA JOHNSSON
GILLLIS OLDFELDT
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

# Sound Source Distance Estimation Using Sound Frequency Attenuation

## Distansapproximering med hjälp av frekvensattenuering

**Frida Johnsson, Gilllis Oldfeldt**

# Sound Source Distance Estimation Using Sound Frequency Attenuation

Frida Johnsson
fr2182jo-s@student.lu.se

Gilllis Oldfeldt
gi2802ol-s@student.lu.se

December 29, 2022

Master's thesis work carried out at AXIS Communication.

Supervisors: Amir Aminifar, amir.aminifar@eit.lth.se
Azra Abtahi Fahliani, azra.abtahi_fahliani@eit.lth.se
Magnus Rosell, Magnus.Rosell@axis.com

Examiner: Erik Larsson, erik.larsson@eit.lth.se

**Abstract**

Estimating the distance between a sound source and receiver is an important problem. This thesis explores the possibility of estimating distance from a sound source using the attenuations regarding different frequencies over a distance. This method can be useful in cases where currently existing distance approximation methods are not an option. We aimed to use an image recognition convolutional neural network, alongside a widely used feature extraction tool in sound analysis called Mel Frequency Cepstral Coefficients to create a model that can use frequency attenuation for sound source distance estimation. Our goal is to perform a pilot study, hoping for an accuracy within a few meters. The final model in this project has an accuracy within a few meters when tested indoors with little ambient noise. The model is limited in scope and has limited data, so the results may not be reliable beyond the scope of this thesis.

# Contents

# Chapter 1

# Introduction

## 1.1 Background

The process of measuring the spatial distance between a sound source and a receiver is called sound source distance estimation. This process can be used in several applications, such as determining the distance to a gunshot for surveillance purposes [5], robot human interaction like speech recognition[16] and service robots [3].

Humans have a good sense of spatial distance and there have therefore been studies on the accuracy of human perceptions of sound signals [4] [20]. When a machine is trying to perform sound source distance estimation it is usually done by transmitting data from an array of microphones or a binary sound source [29] [6]. However, there are situations where data from only one source is accessible. In these scenarios, there are several features of the signal that varies with distance [7].

The topic of this project, using frequency attenuation for sound source distance estimation, appears to be an unexplored area. Since there is no previous work to draw upon, we will be using standard sound analysis methods commonly used in machine learning.

The correlation between the features of a signal and the distance traveled by that signal is not straightforward. The acoustics of the room, the temperature, and more have an impact on the signal. Due to this, there is reason to perform sound source distance estimation with the help of machine learning. A machine learning approach that shows promising results in sound analysis is using convolutional neural networks in combination with mel based spectral features [28] [30]. This is discussed in sections 2.2.3 and 2.2.4

## 1.2 Problem Statement

The goal of this project is to estimate the distance between a sound source and a microphone inside a system of network devices. When attempting to communicate with a specific device on a network, some way of identifying that device is needed. One solution to this is to use the fact that the devices on the network have different physical locations. If a user could estimate the distances between a handheld device and the devices on the network, they could place themself close to the device they wish to communicate with, and would then be able to identify that device on the network using their distance to the user.

Measuring the distance between two electronic devices is usually done with common types of wireless communication such as Bluetooth or WiFi. Using sound signals is another common solution for distance measurement. The travel time of the signal or the fallof of its volume are the most common ways of achieving this. However, sometimes none of these solutions can be used. For example; when the devices do not have wireless communication, or when the internal clocks of the different devices are not in sync, or when the output volume of the devices is not known.

The approach to distance estimation we explored in this project came to be during a summer work project where the installation process of network speakers were to be streamlined. We needed some way of identifying the speaker, and estimating the distance between a user and a speaker seemed to be a good solution to this. The idea of using frequency attenuation came from the constrained nature of these systems, where previous established methods of distance, volume, and time offsets could not be used

## 1.3 Purpose of the Study

The aim of the project is to attempt to exploit a property of sound called frequency attenuation to estimate the distance between a speaker and a microphone. The idea is to use a convolutional neural network with mel frequency cepstral coefficients for this task. We are hoping to achieve predictions with errors less than a few meters.

## 1.4 Disposition

This report will contain relevant theory for the experiments that have been done during the project. This includes some basic information about sound analysis and the machine learning technologies that have been used. The report will also cover the process of the data gathering, the sound analyses, and the creation of the machine learning models. The results of which will be presented with graphs and followed up with a discussion of reasons behind the results, and future improvements.

# Chapter 2

# Theory

## 2.1 Sound Analysis

Using sound as a mapping tool is not a new thing. Different animals like bats and dolphins use differences in sound frequency, intensity and duration for tracking and navigation. Sound waves are pressure waves created by vibrations and when acoustic waves moves through a medium like air or water we say that it propagates through that medium. During propagation, the waves can be reflected, refracted, and attenuated. The medium affects the propagation by its density, pressure, temperature, motion, and viscosity [23]. The propagation of sound is what makes it possible to determine properties like distance and direction.

### 2.1.1 Frequency Dependent Attenuation

The attenuation rate of the sound wave is dependent on the wave's frequency, higher frequency resulting in more attenuation. Higher frequencies are also highly reflective, resulting in more inference [9]. The frequency dependency of acoustic attenuation can be expressed through the power law:

$$\begin{aligned} S(x + \Delta x) &= S(x)e^{-\alpha(\omega)\Delta x} \\ \alpha(\omega) &= \alpha_0|\omega|^y \end{aligned} \tag{2.1}$$

where $S$ represent the amplitude of a acoustic field variable such as pressure or velocity, and $\omega$ represent the angular frequency, and where $\Delta x$ is the propagation distance. There are two material dependent variables $\alpha_0$ and $y$ where $y$ lies between 0 and 4. For example, in water, the $y$ coefficient is 2 and the attenuation becomes frequency squared dependent [2].

### 2.1.2 Stokes Law

Stokes law of sound attenuation illustrates the loss of amplitude of the sound wave when it travels through a Newtonian fluid, because of the fluid's viscosity [21]. A Newtonian fluid is a fluid with some specific properties, air is such a fluid [17]. Stokes law states that the amplitude of the sound wave decreases with a rate of

$$\alpha = \frac{2\eta\omega^2}{3\rho V^3} \tag{2.2}$$

where $\eta$ is the dynamic viscosity coefficient, $\omega$ is the angular frequency, $\rho$ is the density and $V$ is the speed of sound [21]. There are more factors of sound attenuation that is not accounted for within the expression.

The viscosity, density, and speed of sound in air are affected by temperature. For viscosity, the most important factor is the temperature. For liquids the higher the temperature the lower the viscosity. The relationship can be expressed in a function that takes the temperature as input and outputs the viscosity as follows [22]:

$$\eta_{air}(T) = 2.791\dot{1}0^{-7}T^{0.7355} \tag{2.3}$$

Temperature affects the speed of sound in air by heating the air molecules and giving them more energy to vibrate faster, this results in the sound waves traveling faster. The difference in speed in room temperature versus freezing temperatures is 15 meters per second. The formula for the speed of sound in air is [25]:

$$v = 331m/s + 0.6\frac{m/s}{C}T \tag{2.4}$$

### 2.1.3 Frequency Response

In the context of audio, the frequency response denotes how the speaker system will replicate an input signal. The human ear can hear frequencies between 20 Hz to 20 kHz but not all speakers can replicate these frequencies. Different frequencies will also be amplified differently, changing the volume of the input signal. In most speakers' technical specifications, there is therefore a frequency band and a deviation denoted in ± db [13].

### 2.1.4 Anechoic Chamber

Anechoic means echo-free and an anechoic chamber is a room with the purpose of eliminating echos. It is frequently used by product manufacturers to test the sound of components. Orfield Laboratories was in 2014 the quietest place on earth with a chamber that had a sound level of -9 decibels. For comparison a quiet library has a sound level of around 30 decibels [14]. An anechoic chamber usually has the walls and the ceiling covered with foam triangles, an example of which can be seen in Figure 2.1.

**Figure 2.1:** A example of a anechoic chamber [24].

### 2.1.5 Signals

#### Impulse

A transmission system is defined as an enclosed space that sound travels through. The properties of which can be found by looking at the impulse response. The impulse response is defined as the output signal of a system that has been fed an impulse [11]. This process as well as the characteristics of the signal can be seen in Figure 2.2.

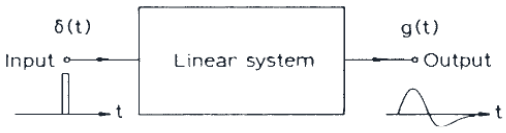



**Figure 2.2:** Figure of an impulse response [11].

An impulse signal includes all frequencies in equal amounts, likewise, the impulse response represents the response to all frequencies [26].

#### Spectrum

Any signal $x(t)$ can be represented by a sum of sinusoidal waves over a period of time $t$ as follows [27]:

$$x(t) = A_0 + \sum_{k=0}^{N} A_k cos(2\pi f_k t + \phi_k) \tag{2.5}$$

If the amplitudes $A_k$ and the phases $\phi_k$ is written as a function of the frequencies $f_k$ we can instead describe the signal with respect to frequency. This is defined as the spectrum $X(f)$. The spectrum uses the amplitudes and the phases to represent $x(t)$ as a function of the frequencies in $x(t)$ [27].

A magnitude spectrum is used to find the distribution of energy over a signal's frequencies. A Fourier transform can be used to get the specific magnitudes for this [27].

## 2.2 Machine Learning

### 2.2.1 The Basics of Neural Networks

A neural network is a machine learning method that tries to emulate biological brains.

Put simply, a brain, or a real neural network, works by having a web of interconnected neuron cells, that send electrical signals to each other. Each neuron will send out a signal depending on the strength of its input signals. The structure of the network, along with the specific signal criteria each neuron has to send a signal of their own, gives rise to the complex behavior.

The way a neural network in a machine learning context tries to emulate this is by having a network of interconnected nodes. Each node has inputs and outputs, and an activation function that takes its inputs, and decides what its output will be. Each neuron has parameters, also called weights, that can change to modify the behavior of its activation function and output. Each node also has a bias, a constant value that is added to its input. The bias is used to help the models to shift the activation function towards the positive or negative side.

These networks are generally structured in layers, with an input on the top layer, and an output at the bottom layer.

The weights of the connections and the biases are the trainable parameters of a network.

#### Training a network

For a neural network to perform the task one wants it to perform, it needs to be trained.

Training a network requires some training data. These data are fed through the network, and the output is then evaluated according to a loss function. The loss function is in essence a way of measuring how well the network performed the task one wanted it to perform. There are a plethora of loss functions, depending on the nature of the data one is using.

To move a network towards the behavior that is wanted, an optimizer is used. This optimizer is trying to minimize the result of the loss function, by updating the values of the trainable parameters after all the data has been fed through the network. The size of the steps the optimizer is using is called the learning rate.

Each full cycle of feeding the data through the network, and then updating its trainable parameters is called an epoch.

#### Supervised learning

The type of training we use in this project is called supervised learning. This means the data we use to train consists of some input data ($X$), and their corresponding output values

($y$). The value for the loss function for each data point in $\boldsymbol{X}$ is calculated by comparing the network's output value with the known $y$ value. This comparison can be done in several different ways.

### Overfitting

When training a machine learning model, it is important to try to avoid overfitting the model. Overfitting means that the model has been fitted too well to the training data. This can happen because the training data are not general enough, or has some hidden pattern that the model was able to pick up on, along with a lot of other reasons. The result of this is that the model will perform very well with data it was trained on, but not with data it has never seen before.

### Validation set

To be able to check the performance of a model, looking at the loss for the data the model was trained on is not enough. Since the loss tells one nothing about how the model performs with data outside of the training set, another set of data is needed. This other set of data is called the validation set.

The validation set can be fed through the network every epoch, without updating the weights, to give information about how well the model is actually performing each epoch.

### Hyperparameters

There are a lot of variables that can be changed about a neural network to impact its performance, like how well it will be able to be trained, and how prone to overfitting it will be. Things like the network's structure, usage of regulizers, the learning rate of the optimizer, and even how the training data is preprocessed are all things that can have a large impact on the final performance of the model.

All of the mentioned variables are called hyperparameters. Finding the best values for these hyperparameters is one of the main challenges of a neural network.

### Mean squared error

The Mean squared error is a loss function that takes the average of the square of each predicted values error. This means that large errors will have an exponentially larger impact than small errors.

The Mean squared error $\boldsymbol{MSE}$ is defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

where $\boldsymbol{n}$ is the number of predicted values, $y$ is the real values, and $\hat{y}$ is the predicted values.

### Mean absolute error

The Mean absolute error shows how large the error is in general. This is a useful tool to give an idea of how accurate a model is.

The Mean absolute error $MAE$ is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.7}$$

where $n$ is the number of predicted values, $y$ is the real values, and $\hat{y}$ is the predicted values.

### Regulizers

The purpose of using a regulizer in a neural network is to reduce overfitting. A regulizer will stop a network from getting too specialized, by punishing excessive growth of certain trainable parameters given the model's performance on the validation set [12].

Activity regulizers will penalize the output of the layer, while Bias regulizers will penalize the bias of the layer [12].

### Batch Normalization

Batch normalization is a method that is used to stabilize the training of a network. A small change in one layer might give very large effects on the layers beneath it. Batch normalization works by normalizing the connections between layers, which can reduce this effect [8].

## 2.2.2 Tuning Pretrained Networks

A good strategy when we have a limited amount of data is to use a pretrained network and retrain it with the new data. This significantly cuts down the amount of data needed. A strategy for doing this is to add layers to the end of the network, and either train only the added part of the network, or the entire network. This process is called tuning.

### MobileNetV2

MobileNetV2 is a light-weight image classification network. The authors, Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen, describe the arcitechture of the network as "The MobileNetV2 architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer." [19].

### 2.2.3 Convolutional Neural Networks

A very popular way of tackling the problem of analyzing audio with machine learning is to borrow from the image analysis side of machine learning. There, the most widely used type of model is called a Convolutional Neural Network (CNN). By converting the sound to an "image", we can utilize the power of CNN:s in audio research.

A CNN uses layers called convolutional layers in its network. These layers apply sliding filters over its input. These types of layers are very useful for image analysis, because the filters can be multidimensional, and can therefore use the spatial context of an image.

### 2.2.4 Sound Feature Extraction to Create an Image

When training a Neural Network, it is common to transform the training data before use. This transformation can be done simply to format the data to a usable format for training, but also commonly includes steps to extract the features of interest in the data. A common feature extraction method for sound analysis is called the Mel Frequency Cepstral Coefficients (MFCCs). This method turns the waveform of the sound into an image, which is exactly what we need when using an image-based network. An example of how an image created with this method can be seen in figure 2.3

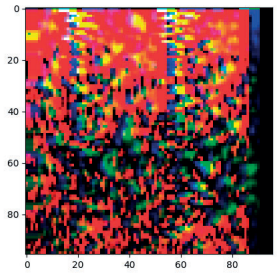



**Figure 2.3:** Extracted MFCCs of a pulse signal

#### Mel frequency scale

The mel scale is an attempt at mapping the normal linear frequency spectrum to a more "human-centric" scale. Humans do not perceive frequency linearly, so the mel scale is logarithmic, and tries to closely match what a person would perceive as a linear frequency scale [15].

A general formula for converting a frequency $f$ to mels $m$ is [15]:

$$m = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.8}$$

### Cepstrum

In general, a cepstrum is obtained by transforming the logarithm of a spectrum of a signal in a way to highlight periodic structures in the spectrum. This transform is generally the inverse Fourier transform, but other transforms such as the discrete cosine transform are also widely used. Highlighting these structures can be a very useful tool when analyzing a signal, especially sound, which has a lot of periodic harmonic frequencies to analyze [1].

### Mel frequency cepstral coeficcents

Mel frequency cepstral coefficients, shortened as MFCCs, is a widely used feature extraction tool used in sound analysis. It is calculated by first using a short time fourier transform on a signal to obtain a spectrum; then, mapping this spectrum to the mel scale, using triangular overlapping windows in the process. Then, the discrete cosine transform is applied to the logs for the powers of each mel frequency band obtained from the triangular filters. This results in a mel frequency cepstrum. The amplitudes of this spectrum are what are called the mel frequency cepstral coefficients [18].

A useful variation to the basic MFCCs process is to also calculate the first and second derivatives of the cepstrum, to give further information about the signal.

## 2.2.5 Alternatives to Using MFFCs and CNNs

### MFCCs vs spectrums

Since the mel scale is based on human perception, it is very counter-intuitive that using any features based on this scale would perform better than using the normal frequency scale. However, MFFCs have been widely used in machine learning research that is not related to human perception, or sound, and have been shown to yield better results than using the raw spectrum [31] [28].

### Waveform based networks

There are more modern solutions that instead of using MFCCs and image recognition networks use the raw waveform as input to the networks. These solutions are in many ways superior to the older MFCCs-based solutions [10]. The reason for us choosing to use the often inferior image-based networks can be seen in section 3.7.1

# Chapter 3

# Methodology

## 3.1 Simple Data Synthesis

For the first simplest proof of concept models, we used completely synthesized data. We did this for a couple of reasons. First to check if the concept would work. And secondly to see how the attenuated sound data would work with machine learning.

Using the Formula 2.2 for stokes law, we simulated how a signal consisting of a selection of frequencies with equal amplitudes would attenuate over a distance in an ideal setting. To get the attenuation from stokes law we calculated the viscosity of the air as a function of temperature per Formula 2.3. And for the density term, we used a lookup table with temperature as the input. There are many attenuation factors that were not considered at this state. Such as air humidity, frequency response, absorption, and reflection.

We trained all simple regression models in the scikit learn library on the synthesized data. The best performing model types were linear and Gaussian process regression. These regression model types were used for our four final machine learning models with the synthetic data. For the models to not use the general volume as a feature the general amplitude was randomized. First, we created Linear and Gaussian process regression models for the pure data. Then we also added noise to the amplitudes of the different frequencies, which gave us two more regression models. The best performing model was also tested with real sound data.

## 3.2 Type of Signals Used

During our experiment, we used five different kinds of signals.

- A sweep, a signal that linearly changes from 0 to 20 000 Hz

- A short pulse
- Multiple sine waves
- A sine wave of one frequency followed by a sine wave of another frequency.
- Multiple sine waves of some frequencies, followed by multiple sine waves of some other frequencies

The reason behind using a sweep was that it includes every single frequency, which would give the most accurate data on how the amplitudes of every single usable frequency have changed. It however does not have a lot of usable data on how the reverberations in the room behave.

The logic behind using a pulse is similar to a sweep since it includes every frequency. However, since the pulse is only recorded with a few samples, a lot of frequency information is lost. The short length of the signal can still be favorable since it makes it easy to cut a lot of the reverberations from the data.

The last two signals we would call step-sounds in the project. The logic behind using step-sounds is to get a good compromise between the two previous signal types. Long stretches of constant frequencies, with a sharp "pulse-like" change in the signal.

## 3.3 Ideal Data Gathering

In our collaboration with AXIS, we got access to an anechoic chamber. For these experiments, we used two different signals: One with multiple sine waves between 0-20000 Hz, and one consisting of pulses. The collected sound data was very similar to the data we synthesized since the chamber eliminated all noise and reverberations.

We performed this experiment as an extension of the last one. We theorized that the interaction with the environment and noise would be the most hindering elements to the machine learning results. This experiment was made to see if the theory worked without these factors.

We collected a few hundred sound recordings ranging between 0 to 3 meters. The data collected was used to create linear and Gaussian regression models, these type of models was chosen because they performed the best on the data. The mean squared error for the validation data was calculated and a scatter plot of the relationship between the real value and the predicted value was made. The predictions of the resulting models were tested inside, and outside the anechoic chamber.

## 3.4 Attempting to Create a Robot for Automated Data Gathering

We knew that we were facing a complex problem and that a simple machine learning model as in previous experiments would not suffice. We were also considering the effect different environmental settings would have on the sound, and wished to gather data from as many

different settings as possible. The data would also have to be gathered in a large range of distances and positions relative to the speaker. Doing this manually would be labor-intensive, slow, and prone to human errors.

In order to address this challenge, we attempted to create a robot that could position itself accurately within a room and gather data for us. This automated data gathering would address all of our concerns and eliminate many of the problems associated with manual data collection. By using a robot, we hoped to streamline the data gathering process and improve the accuracy of our results.

The robot ultimately could not be used due to time constraints not allowing us enough time to finalize it. Even though we were unable to fully operationalize the robot, our efforts on the project may still be valuable for others who want to build upon our work in the future. The robot can be modified with a relatively small amount of work to perform automatic sound recording, and it was a major part of the project. By sharing our findings, we hope to provide a helpful starting point for others who may be interested in continuing our work.

### 3.4.1 Main working principle of robot

The principle we decided to use for the positioning of the robot was to suspend the microphone with strings from three anchor points in the ceiling and have the robot vary the length of these strings; creating three axes of movement, and therefore letting the robot move the microphone anywhere within a triangular prism bound by the three anchor points in the ceiling. The illustration of the setup and the boundaries can be seen in Figure 3.1.

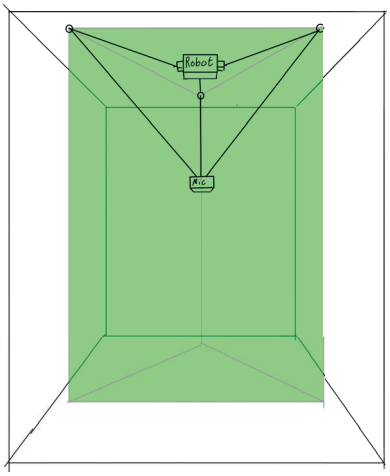
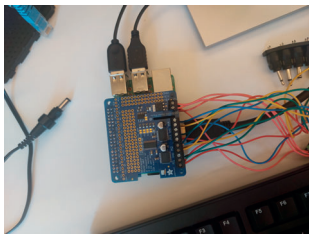


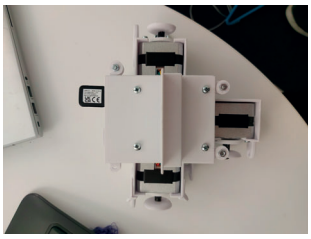

**Figure 3.1:** Bounding prism for robot movement.

### 3.4.2 Build and Hardware

The robot brain is a raspberry pi 3, with two Adafruit motor HATs to control three stepper motors. The stepper motors in turn move the strings that held the microphone. The unit can



**(a)** Raspberry pi



**(b)** Robot case

**Figure 3.2:** Images of the robot enclosure and the raspberry pi.

be seen connected to a screen and keyboard in Figure 3.2. As the separate parts had to hang from the ceiling we had to make a custom-made case that we then 3d printed.

The ceilings we planned to hang the robot from were located at an AXIS location which had ceilings with panels. The panels were held up by a 1 cm metal ledge and we used this to our advantage when designing the robot. The top part of the robot could slide between the ledge and the panel, keeping the robot up. Following the blueprints of the step motors and the raspberry, we could make a snug fit with outlets for cables, that used a minimal amount of plastic. The robot was designed with the purpose of moving it around, an idea that was later changed. There was therefore a perceived need for minimal weight and for the internal parts to stay in place. The resulting case can be seen in Figure 3.2.

This enclosure was mounted in the ceiling, and the strings were fed through holes in the three 3d-printed mounting points in the ceiling. These are then tied to the microphone, which was connected with a cable to a laptop. The illustration of this can be seen in Figure 3.1.

### 3.4.3 Connecting the Robot to WiFi

The raspberry pi 3 that we are using for the robot had a defective wireless module. This meant we had no way of connecting the robot to both WiFi and the Bluetooth speaker. The Bluetooth issue was solved by making the laptop that controlled the robot also control the speaker. But the robot did need to have access to WiFi to be controlled remotely when it was mounted. Otherwise, we would need a keyboard, mouse, and screen to be plugged into the robot while it was hanging from the ceiling. This would make the calibrations and testing much harder.

When these experiments were made there were no other raspberries available to buy. So the issue had to be solved with a WiFi USB dongle. However, as raspberry Pi 3 is supposed to have WiFi built in there are not a lot of compatible dongles. There is only a small span of Linux versions that support the version of python that is needed to control the motors. And these Linux versions are not compatible with most dongles. We did manage to find one in the end.

The WiFi dongle would however not work without some more problem-solving. Firstly the

drivers for the dongle had to be found externally from an open-source project. And secondly, the dongle was recognized as a mass storage unit by the raspberry on boot and would need to be ejected to work as a network device. To circumvent this we had to write a script that correctly ejected the device automatically when the raspberry was plugged in.

### 3.4.4 Increasing the Strength of the Motors

The step motors that were available to us were weak. A fact we learned throughout the process. In our original plan for the robot, we wanted to move the whole robot, pi and motors and all, around the room. But since the motors could not lift more than a couple of hundred grams, we had to work around that in some way.

We tried many things that would make it possible to move the robot around. We attempted to use mechanical pulleys to lessen the load on the motors. We also changed the mode of the motors to a more powerful, but less accurate mode. Another thing we tried was to decrease the diameter of the spools mounted on the motor axle to increase the force it could output. But the motors were still too weak to lift the around 1 kilogram robot.

We then changed the approach to have a stationary robot and motors. Instead of having the microphone plugged into the robot we connected it with a cable and moved just it instead. Some design changes did have to be made as a result of this. In the end, the microphone and cable weighed around 150 grams.

This was still not enough to be able to move around the microphone. The diameter of the spool we had mounted on the axle was still too large, even at 3 cm. Removing the spool completely and having the string wrap around the naked axle was the only way we got enough power to actually lift the microphone.

### 3.4.5 Spooling the Thread

#### Using a spool

The original plan for controlling the length of the strings used to position the robot and later microphone was to use spools on the motor axles. The plan was for the thread to wind up on the spool. The illustration of this system can be seen in Figure 3.3.

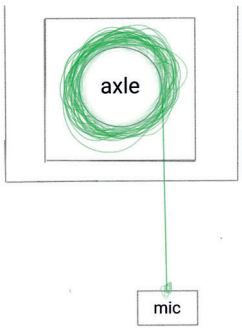



**Figure 3.3:** String spooling around the motor axle.

As we came to understand how weak the motors were we had to change our approach, we needed to reduce the diameter of the axle. The original plan was to use a large diameter of the spool to reduce the change in diameter the spooling of the thread would have. We started by investigating if the string would be wound up on the smaller spool in a predictable way, so we could model it into the positioning software. It turned out not to be predictable, and to mitigate this we would have to make the spool width very small. A narrower spool resulted in a large axle diameter when the string was completely wound up, which meant that the motor no longer could lift the microphone.

## Using counterweights

The solution for the spooling issue was to not use the motor axle as a spool, but instead, use it as a way to feed the string through at a constant rate. The original idea was to wind the string some turns around the axle, to give it enough friction to grip on the axle, and then unwind again by using a counterweight. It turned out that even if the string was just wound around the axle a single time, it would get tangled up and start winding up the string at both ends.

The way we tried to solve this was to switch to a thicker rougher rope and glue course sandpaper on the axle. This made it possible to have the string wound just half a turn around the axle, but still have enough grip to lift the microphone. The illustration of how this looked can be seen in Figure 3.4.

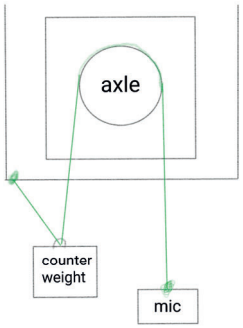



**Figure 3.4:** String with counterweight lying on the motor axle.

### 3.4.6 Calibration

Calibrating the positioning system was a challenge. The goal was to give the system enough information to position the microphone accurately, and to always know the distance to the speaker. To accomplish this there were three problems that needed to be solved.

#### The lengths of the strings

Knowing the length of the strings at all times was the first problem. Given that the diameter of the motor axle was known and that the motor could be spun in a predictable way, it was possible to calculate how much the length have changed. If we knew the starting length and saved it we could update the value every time we modified the length.

#### The positions of the anchor points

The second problem was to have knowledge about the positions of the anchor points. This required defining a coordinate system, and finding a way of measuring and calculating the position of the anchor points in this coordinate system.

The way we solved this was by defining the triangle created by the anchor point as seen in Figure 3.5. First, we defined one anchor point $A$ as having the coordinates $(0, 0)$. Then, we measured the distance $AB$ from anchor point $A$ to anchor point $B$, and defined the coordinates of anchor point $B$ as $(0, AB)$. Next we measured the distance $AC$ between anchor points $A$ and $C$, as well as the distance $BC$ between anchor points $B$ and $C$.

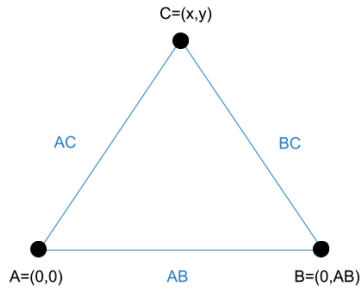



**Figure 3.5:** Triangle created by the anchor points.

We then found the coordinates of C using the distance formula $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ twice.

$$AC = \sqrt{(x-0)^2 + (y-0)^2)} \rightarrow$$

$$\rightarrow AC^2 = x^2 + y^2 \rightarrow$$

$$\rightarrow x = \pm\sqrt{AC^2 - y^2}$$

$$BC = \sqrt{(x-0)^2 + (y-AB)^2} \rightarrow$$

$$\rightarrow BC^2 = x^2 + y^2 + AB^2 - 2yAB \rightarrow$$

$$\rightarrow 2yAB = (\sqrt{AC^2 - y^2})^2 + y^2 + AB^2 - BC^2 \rightarrow$$

$$\rightarrow y = \frac{AB^2 + AC^2 - BC^2}{2AB}$$

There existed two possible x values for C and we would use $x = \sqrt{AC^2 - y^2}$ moving forward.

## The position of the speaker

The third problem was knowing the position of the speaker. This was done similarly to the previous problem, using the tetrahedron created by the three anchor points and the speaker as seen in Figure 3.6. We added a dimension to our plane with the anchor points. The anchor points coordinates was then $A = (0, 0, 0)$, $B = (0, AB, 0)$ and $C = (\sqrt{AC^2 - y^2}, \frac{AB^2+AC^2-BC^2}{2AB}, 0) = (x_C, y_C, 0)$.

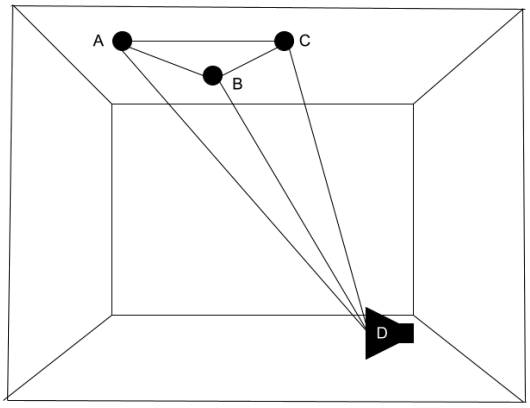



**Figure 3.6:** Speaker positioning.

We measured the distances $AD$, $BD$ and $CD$, between the anchor points $A$, $B$ and $C$ to the speaker $D$. This time we used the distance formula in three dimensions three times.

$$AD = \sqrt{(x-0)^2 + (y-0)^2 + (z-0)^2} \rightarrow$$
$$\rightarrow x = \pm\sqrt{AD^2 - y^2 - z^2}$$

$$BD = \sqrt{(x-0)^2 + (y-AB)^2 + (z-0)^2} \rightarrow$$
$$BD^2 = x^2 + y^2 + AB^2 - 2yAB + z^2 \rightarrow$$
$$\rightarrow 2yAB = (\sqrt{AD^2 - y^2 - z^2})^2 + y^2 + AB^2 + z^2 - BD^2 \rightarrow$$
$$\rightarrow y = \frac{AD^2 + AB^2 - BD^2}{2AB}$$

$$CD = \sqrt{(x-x_C)^2 + (y-y_C)^2 + (z-0)^2} \rightarrow$$
$$CD^2 = x^2 + x_C^2 - 2xx_C + y^2 + y_C^2 - 2yy_C + z^2 \rightarrow$$
$$\rightarrow CD^2 = (\sqrt{AD^2 - y^2 - z^2})^2 + x_C^2 - 2x_C(\sqrt{AD^2 - y^2 - z^2}) + y^2 + y_C^2 - 2yy_C + z^2 \rightarrow$$
$$\rightarrow CD^2 - AD^2 - x_C^2 - y_C^2 + 2yy_C = 2x_C\sqrt{AD^2 - y^2 - z^2} \rightarrow$$
$$\rightarrow AD^2 - y^2 - z^2 = (\frac{CD^2 - AD^2 - x_C^2 - y_C^2 + 2yy_C}{2x_C})^2 \rightarrow$$
$$\rightarrow z = \pm\sqrt{AD^2 - y^2 - (\frac{CD^2 - AD^2 - x_C^2 - y_C^2 + 2yy_C}{2x_C})^2}$$

We choose z to be $z = \sqrt{AD^2 - y^2 - (\frac{CD^2 - AD^2 - x_C^2 - y_C^2 + 2yy_C}{2x_C})^2}$.

### 3.4.7 Positioning of the Microphone

When positioning the microphone there were three things we had to consider. First generating a random point within the prism with the base made up of the anchor points. Secondly, we needed to translate the coordinates of the point to the lengths of the strings holding the microphone. Lastly, we needed to consider practical implementation of the microphone positioning.

#### Random point within a triangle

We accomplished this in two different ways. The first way was by finding a random point within a rectangle and then checking if it was within the bound of the triangle. The second way was by a method similar to the method of weighted average. We chose the second method because of its lower time complexity.

In the second method the random points within the triangle made up of the anchor points with vertices $A$, $B$, and $C$ were generated in four steps.

- We defined the vectors $\overrightarrow{AB} = B - A = (0, AB) - (0, 0) = (0, AB)$ and $\overrightarrow{AC} = C - A = (x_C, y_C) - (0, 0) = (x_C, y_C)$ , between point $A$ at the origin and points $B$ and $C$.
- We then generated two random uniform values $u1, u2 \sim U(0, 1)$.
- After this we checked if $u1 + u2 > 1$, if this was true we applied the transformations $u1 \rightarrow 1 - u1$ and $u2 = 1 - u2$.
- Lastly we found the random point by $w = u1 \times \overrightarrow{AB} + u2 \times \overrightarrow{AC}$

The $z$-coordinate was defined as a random value between zero and two meters.

#### Coordinates of the microphone

With the information from the previous steps, the method for getting the microphone to the desired position was simple. When positioning the microphone we had to calculate how long each of the strings suspending it should be. The positions of the anchor points were known and we could compute the vectors between the anchor points and the microphone. The lengths of the strings could then be calculated by taking the norm of those vectors.

#### Practical implementation

Because of the physical way we controlled the strings, we could never let a string slack. If the weight of one of the strings was ever too small, the counterweight would pull the string taught. We would then lose the information about the length of that string. Depending on the angles of the strings there was a risk of one motor not being able to keep up with the two other motors. The motor control code was not capable of variable speed, so we had to find another solution.

The solution was to move the microphone a couple of centimeters at a time. There would then be too small of a change for the different speeds to matter. This also had the added benefit of being able to move to more positions within a set time.

## 3.5 Manual Data Gathering

During the project, we also had to gather manual data. This was done in four iterations. Improving the data quality and decreasing the amount of labor at every step. The data was gathered at distances between 50cm and 7m, due to the inside environments that were available to us.

In our first attempts, the code and the practical component were very simple. The code started to record sound on a USB microphone and then played the sound from a Bluetooth speaker. When the recording was done the sound file was then named after the measured distance. The script was started with the input of the distance we wanted to record at. The method of gathering a data point was then to measure the distance to the speaker and starting the script. At this point, we used three different sounds. A pulse, a sweep, and a step-sound.

In the second iteration, we randomized the volume of the sound. The reason for this was to obscure the volume as a usable feature. We also streamlined the gathering process by using the setup seen in Figure 3.7. The microphone connected to the computer was moved linearly at the same height as the speaker.

**Figure 3.7:** First manual data gathering method using chairs and measuring tape.

The problem with the previous approach is that it created secondary features that a machine learning model could exploit. The most worrisome of which was the reverberation from the ceiling. The ceiling remained at a constant distance from the microphone, giving a possible predictable reverberation of the signal, which could be used to determine the distance. It's hard to see what features a machine learning model is actually using, so there was no way of knowing the features the model used. To solve this we held the microphone at different angles and different distances to the ceiling. Figure 3.8 shows how this was done. To be aware of the distance to the speaker we placed the microphone at the end of a measuring tape that was connected to the speaker. At this stage, we also changed one thing about the code. To eliminate the time feature that was present in the data we added a random amount of silence before the sound was played. This way the model didn't get information about the time it took for the sound to travel to the microphone.

**Figure 3.8:** Second manual data gathering using longer measuring tape.

The last iteration of the manual data gathering was the one that yielded the most data and had the least amount of unwanted features. The signal that in previous experiments resulted in the best performance was the impulse. So we decided to collect only impulse data to have a more accurate model as overfitting was still an issue. We also automated the code more to have less manual input.

An additional session of data gathering was performed towards the end of the project, gathering 701 pulses to be used as the test set. The final amount of pulses recorded between 50cm and 7m was 9532.

## 3.6 Data Preprocessing

The data was gathered as raw sound. To be able to effectively use the data in our machine learning models it had to be preprocessed. This process had four parts. Isolating the part of the sound that we wanted to use, removing faulty data, normalizing the data, and creating MFCCs.

First, we had to cut out the part of the sound file that was useful for us. The sweep and step-sound did not need to be changed but the impulse did. Due to the nature of an impulse the direct sound and reflected sound was separated on the time axis. We compared the performance of using the entire signal, or cutting out most of the reflected sound. We concluded that the second option worked better. To isolate the pulse, our first approach was to find the maximum amplitude of the signal and use a set window size portion around it. When looking at the waveform of the data, some of it had noise that had a higher amplitude than the actual pulse. This noise was in every instance we saw shorter than the pulse so this problem was minimized by using a sliding window maximum to find the pulse instead.

Another part of the preprocessing was to remove faulty data. Some of the recordings were faulty, and had either not captured a signal at all or had a signal that was too quiet. The solution for this was simple, we checked that the recording had a maximum amplitude above a reasonable value. Another source of faulty data was sound that peaked the microphone. The microphone had a maximum amplitude it could collect, the data that was close to this

maximum was therefore removed. After removing the faulty data points the final amount of pulses for the training and validation sets was 7413, and 700 for the test set.

To make sure any potential remaining correlation between volume and distance could not be used by the model, we normalized the amplitude of the data. We also trained some machine learning models with the volume as input, to see if the previous efforts of obscuring the volume as a feature were successful.

Lastly, we created MFCCs. Before using MFCCs we used the spectrum of the sound as the input for the machine learning model, but this yielded bad results. We used the librosa library in python to extract the MFCCs, along with their first and second derivatives. The MFCCs and the derivatives were then stacked into a three-channel image. The pretrained machine learning model we tuned with this data had a minimum input size of a three-channel image of size 96 times 96, so we padded the data to fit this minimum size. We tried using 96, 48, and 12 mel frequency bands.

## 3.7 Machine Learning

At the very start of the project, we used simple statistical regression models for solving our problem. Due to the complexity of our problem, these models did not perform well in a real setting, and this confirmed that we had to use something more advanced.

### 3.7.1 Type of Machine Learning Model

In our project, we were choosing between using an image-based CNN and using a waveform-based model. The information we found about the waveform-based models was very promising, but the problem we were attempting to solve was not very complex compared to the types of problems these networks are typically used for. Since there were a lot more resources available for creating or tuning an image-based model, we made the choice to focus our efforts to train an image recognition-based model for this project.

#### Using a pretrained network

Keras, the machine learning framework that we used in this project, has a plethora of pretrained image recognition networks that are free to use. Because of the limited computational power we had for training our models, a desktop with a single GTX 980, we chose the mobilenetv2 model as a base for this project. This is because it had the smallest amount of trainable parameters of the easily available Keras models, and would therefore be the least computationally intense to train.

The first networks we attempted to train used the structure of mobilenetv2, but without any pretrained weights. This was done because the shape of our MFCCs did not match the set input sizes required to use the pretrained weights for the network.

These models never performed well, so we decided to pad our training data to enable the use of pretrained weights. This yielded significantly better results.

### 3.7.2 Optimizing Hyperparameters

#### Evaluating models

The models are compared via the validation loss. We use the mean squared error as our loss function. For a more intuitive understanding we also calculate the absolute error and make a scatter plot, the plot included the predicted values versus the actual values. We also create plots that show the percentage of predicted values that have an error less than a threshold value, to visualize the accuracy of each model. When using the WANDB tool we also visualize the best hyperparameters with a WANDB sweep summary plot.

#### The WANDB tool

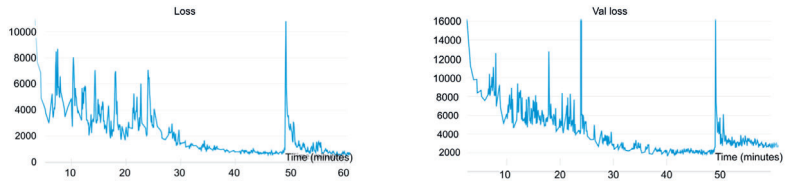
To visualize and test different hyperparameters for the network we use a tool named WANDB. WANDB is a tool that can be used for doing large automated searches for the best hyperparameters for a model. By defining a range of values, it can search the hyperparameter space for good values. It also saves the best model for each run, logs all the relevant info needed to diagnose and compare the performance of models, and calculates statistical significance for each hyperparameter.

#### Tests with all signals

Before using WANDB we made manual runs that tested different hyperparameters and signals. We started with testing models for impulse, sweep, and step-sound signals. The hyperparameters we tested were the learning rate as well as the size and amount of hidden layers. After testing a couple of models for each of the signals we realized that the step-sound performed way worse than any other signal. Moving forward, we therefore didn't use step-sound anymore.

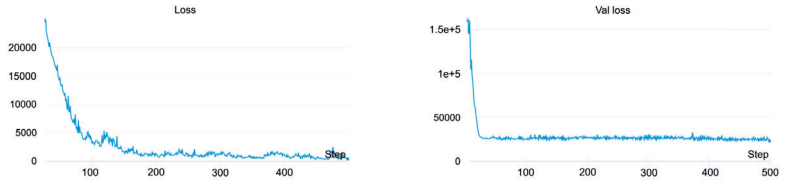
#### Tests with pulse and sweep signals

After starting to use WANDB, we could more efficiently run our tests. In the last experiments, we noticed overfitting so we added activity and bias regularizers. Including regularizers generally resulted in less overfitting. We also continued to try different learning rates and hidden layer sizes. We made around ten models for each signal, the loss and validation loss (val loss) of the best models can be seen in Figures 3.9 and 3.10. The best model for the sweep signal had a smaller loss but a ten times bigger validation loss than the best model for the pulse. The absolute error was around 100 cm for the sweep and around 30 cm for the pulse. With this in mind, we decided to focus solely on creating the best model for the pulse signal.




**(a)** Loss **(b)** Val loss

**Figure 3.9:** Loss and val loss early pulse model.



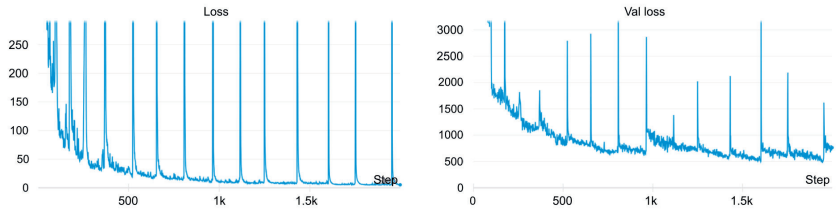

**(a)** Loss **(b)** Val Loss

**Figure 3.10:** Loss and val loss early sweep model.

## First candidate model

To reach our final results, we tried to optimize a selection of six hyperparameters. These were activity and bias regularizes, batch normalization, the number of nodes in each added hidden layer, the number of hidden layers, and the learning rate. The 7413 remaining pulse recordings were split into a training data set of 5461 data points and a validation data set of 1952 points.

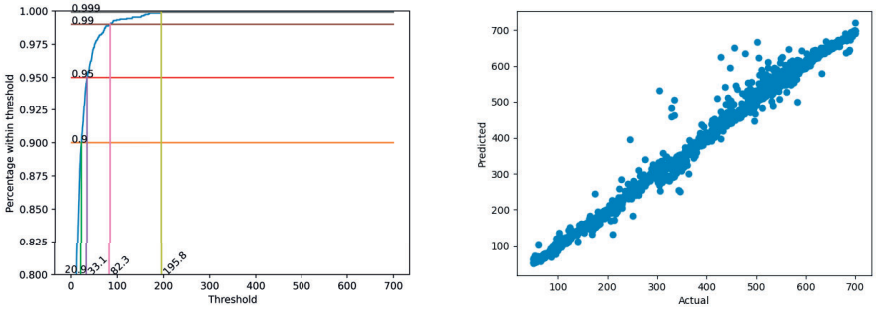
We trained and evaluated over 100 models with the pulse data, while continually adding more data and trying to combat the issues that arose during training. We arrived at a very promising combination of hyperparameters that led to an extremely low loss and val loss which can be seen in Figures 3.11.




**(a)** loss **(b)** val loss

**Figure 3.11:** Loss and val loss for first final candidate model.

The scatter plot for the model shows a very clear relationship between the predicted values and real values, as seen in Figure 3.12(b). The threshold plot, seen in Figure 3.12(a), plots the percentage of predicted values that have an error less than the threshold value. This model seemed to be 95% accurate within about 33 cm.



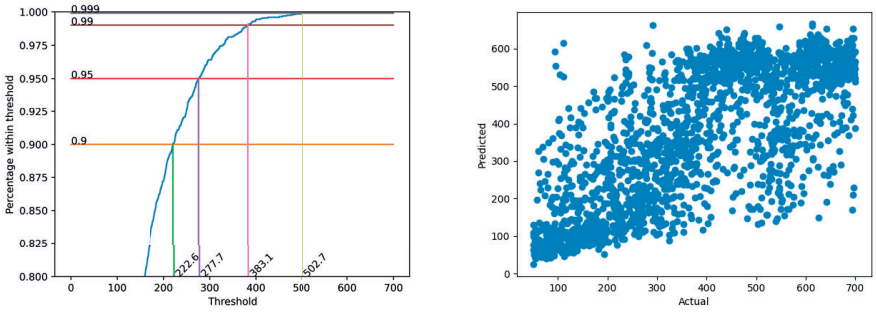

**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 3.12:** First final candidate model results.

When we tried this model in real life, however, it more or less predicted random values. The real-life tests were performed in the same rooms as the data was recorded in, with the same speaker and microphone setup.

We suspected that the issue lied in the training and validation data split. We theorized that the method we had used to split the data resulted in two too similar datasets. To verify this, we chose a different split of the data, that isolated one of the data collection sessions as the validation data, and retrained a model using the previously found hyperparameters. This resulted in a model with significantly lower performance than before, as seen in Table 3.1 and Figure 3.13. This model did perform better in real life than the previous model candidate. Hence we moved forward with this data split.




**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 3.13:** First model with new split.

| Hyperparameters | first model |
|---|---|
| activity regularizer | 0.0001 |
| bias regularizer | 0.0074 |
| batch normalization | false |
| hidden layer size | 1024 |
| no. hidden layers | 3 |
| learning rate | 0.00037 |
| number of mel bands | 96 |
| best val loss | 19956.2 |

**Table 3.1:** Table of hyperparameters used for the first model with new split.

After another search of about 10 combinations of hyperparameters, seen in Figure 3.14, we decided to attempt to reduce the complexity of the training data. We were at this moment using 96 mel frequency bands, while none of the research papers we had read used more than 15.

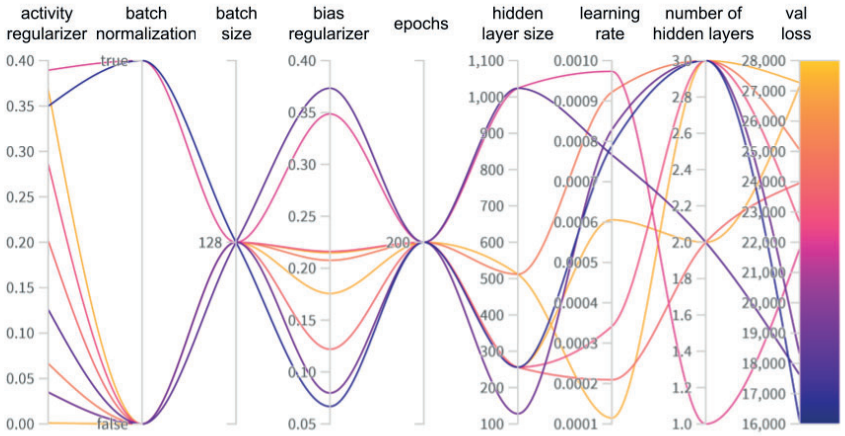



**Figure 3.14:** WANDB sweep summary for new data split.

## Amount of mel frequency bands

We created two more preprocessed data sets, one with 48 mel frequency bands, and one with 12 mel frequency bands. We then evaluated which of these performed the best by trying a range of hyperparameters for each of the new data sets. The results of this can be seen in Figure 3.15, where the "train_data" category represents which of the data sets are being used. The data set with 12 frequency bands seemed to yield the best results, so a more extensive search for the best hyperparameters for this data set was conducted. The results of this search can be seen in Figure 3.16

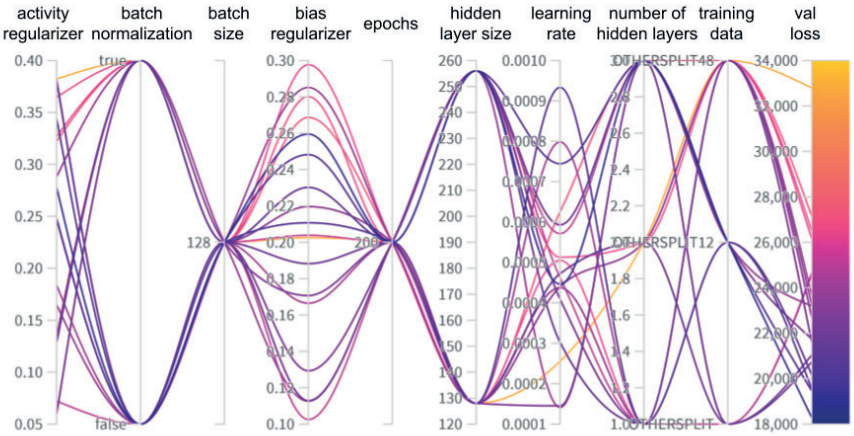



**Figure 3.15:** WANDB sweep summary for finding optimal Mel frequency band count

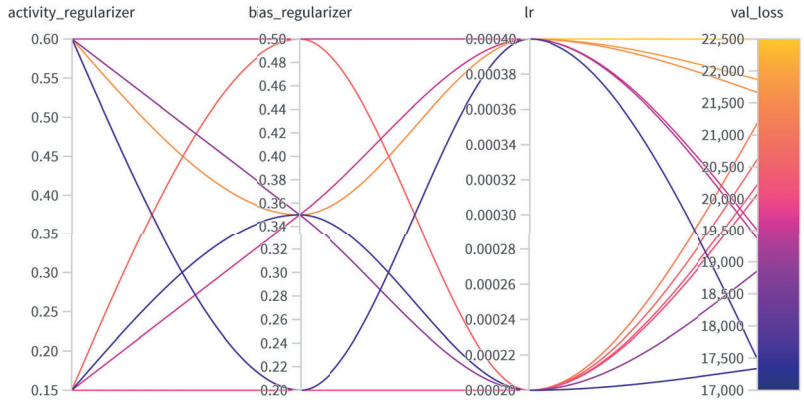



**Figure 3.16:** WANDB sweep summary for 12 Mel frequency bands

Using the best found hyperparameters we trained a final model.

### 3.7.3 Data Generator

As our amount of gathered data increased we had to create a data generator. The vRAM of the GTX 980 we used for training was quickly overwhelmed by the amount of data we were training on. This led us to have to implement a way to dynamically load the data from the disc.

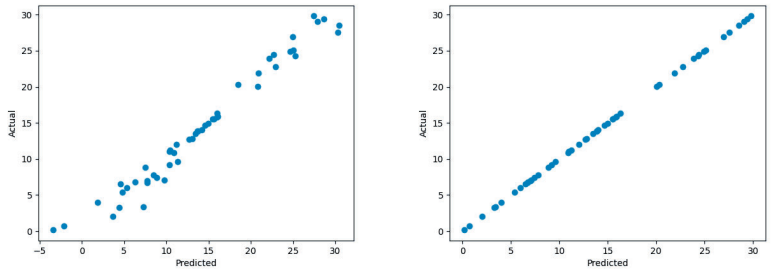
The way we did this was by using Keras' DataGenerator class. This is generally used for dynamic data augmentation and dynamic loading of data from the disc. Since it is already compatible with the Keras framework, we extended the class and created our own data generator that would make sure only one batch of the data was loaded in memory at once.

# Chapter 4

# Results

## 4.1 Synthetic Data Models

Linear and Gaussian process regression models were created with the synthetic data. The scatter plot of the actual values compared to the predicted values for the linear regression and Gaussian process regression models can be seen in Figures 4.1.

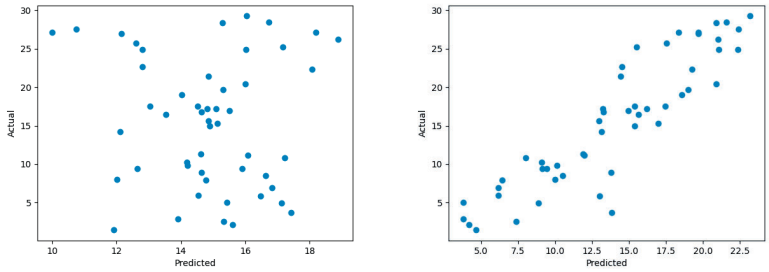



**(a)** linear regression

**(b)** Gaussian process regressor

**Figure 4.1:** Scatter plots (in cm) for synthetic data models without noise.

Linear and Gaussian process regression models were also made with noisy data, the scatter plots of the predicted distance values of these models versus the actual values can be seen in Figure 4.2.




**(a)** linear regression

**(b)** Gaussian process regressor

**Figure 4.2:** Scatter plots (in cm) for synthetic data models with noise.

The mean squared error for all four models can be seen in Table 4.3.

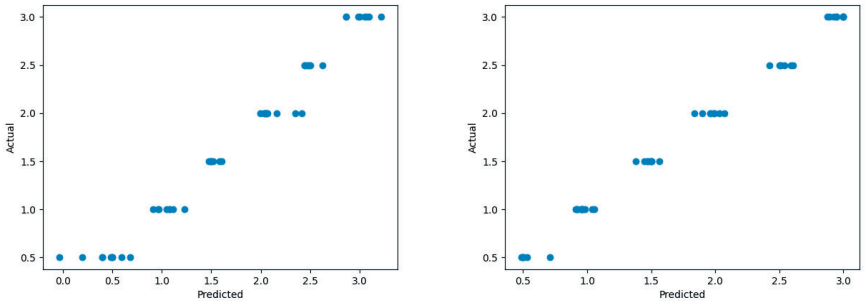
| | Linear regressor | Gaussian process regressor |
|---|---|---|
| Data without noise | 2.076 | $4.759\ e^{-10}$ |
| Noisy data | 19.773 | 75.678 |

**Table 4.1:** Mean square error for the models using synthetic data.

When the model was tested with real sound recordings it had predictions that were several meters off in both directions.

## 4.2 Ideal Data Models

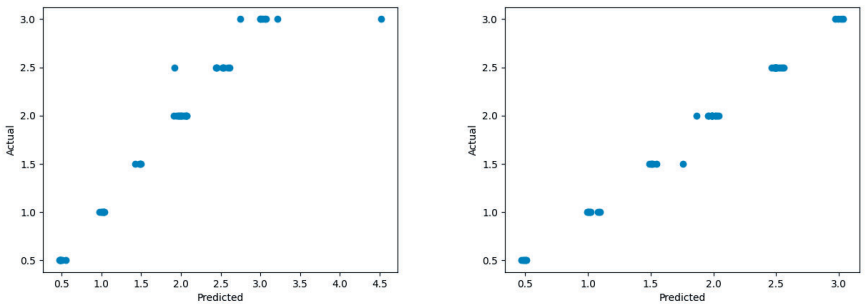
The data for the ideal data models were collected in an anechoic chamber. Gaussian process and linear regression models were created with pulses and signals consisting of multiple sinus waves. The resulting scatter plots of the actual values compared to the predicted values for the different signals can be seen in Figures 4.3 and 4.4.




**(a)** linear regressor

**(b)** Gaussian process regressor

**Figure 4.3:** Scatter plots (in cm) for multiple sine waves.




**(a)** linear regressor

**(b)** Gaussian process regressor

**Figure 4.4:** Scatter plots (in cm) for pulses.

The mean squared errors for the Linear and Gaussian regression models for both signal types can be seen in Table 4.2.

| | Linear regressor | Gaussian process regressor |
|---|---|---|
| Pulse | 0.057 | 0.003 |
| Multiple sinus waves | 0.020 | 0.008 |

**Table 4.2:** Table of mean square error for the models using ideal data.

Testing the best model inside the anechoic chamber gave predictions within a few cm of the actual values. When testing the model outside the anechoic chamber in a regular room the predicted distances were several meters off in both directions.

### 4.2.1 Failed Automated Data Collection

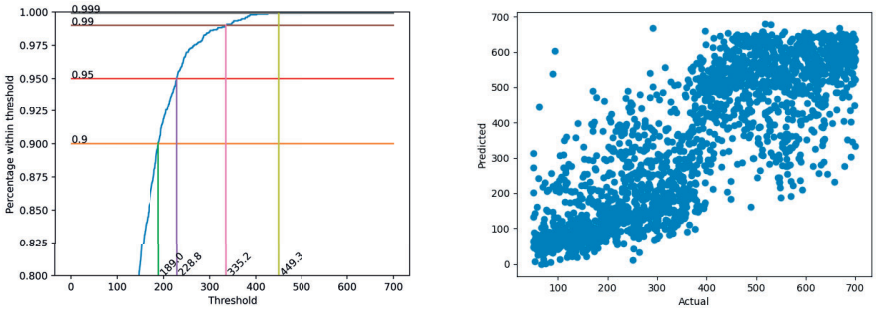
The finished robot could not move the microphone accurately enough. Even after using a thick rope, adding counterweights, and covering the axles in sandpaper the rope would still slip. The force on the rope would change depending on the angle of the rope. Therefore, it would also change the amount it was slipping. The added error of moving the microphone thousands of times was too excessive for the robot to be useful.

### 4.2.2 Final Neural Network Models

The best model, along with its hyperparameters, can be seen in column one in Table 4.3. Figure 4.5 shows the accuracy and a scatter plot of its performance on the validation data. A clear relationship between the predicted and real values can be seen in the scatter plot. If one would use this model they could be 95% certain that the error was not larger than 230 cm.

| Hyperparameters | Best model | Second best model |
|---|---|---|
| activity regularizer | 0.15 | 0.13469 |
| bias regularizer | 0.2 | 0.2038 |
| batch normalization | true | true |
| hidden layer size | 128 | 256 |
| no. hidden layers | 1 | 1 |
| learning rate | 0.0004 | 0.00046 |
| number of mel bands | 12 | 96 |
| Best val loss | 15086.0 | 15287.7 |

**Table 4.3:** Hyperparameters used for each model and their results.

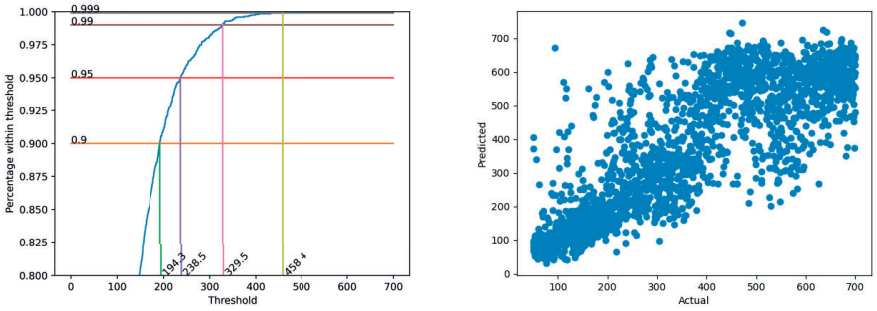



**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 4.5:** Best model parameter result.

The second best model, along with its hyperparameters, which can be seen in column two in Table 4.3, performed slightly worse than the best model. The most significant difference between this model and the best model was the larger amount of mel frequency bands in its training data. As seen in 4.6, it still shows similar performance as the best model, just slightly worse.



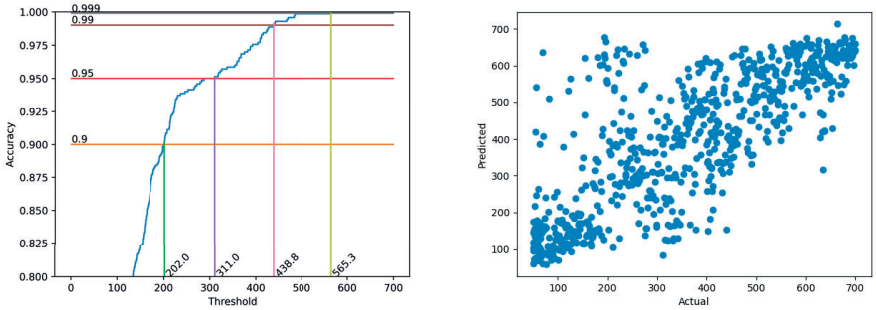

**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 4.6:** Second best model parameter result.

### 4.2.3 Performance On Unseen Test Set

The best performing model was evaluated on the test set. The loss was 72665, and as can be seen in Table 4.3, the model performed significantly worse than on the validation set. But the threshold plot seen in Figure 4.7, seems to show similar performance for the majority of the set, with 90% of errors being less than 202 cm. The scatter plot seen in 4.7 also shows a clear correlation between the predicted and actual values of the test set.



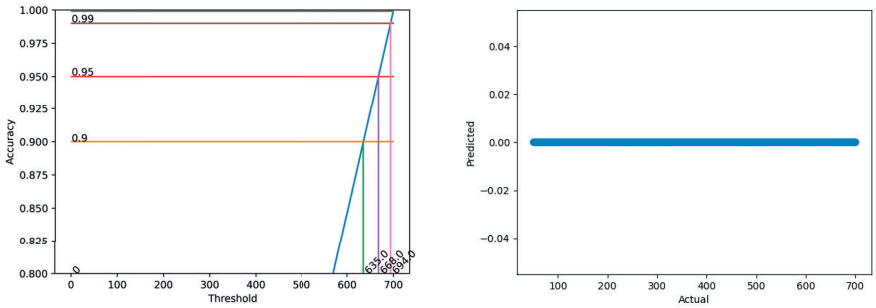

**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 4.7:** Best model performance on test set.

## 4.3 Simple Models With the Real Data

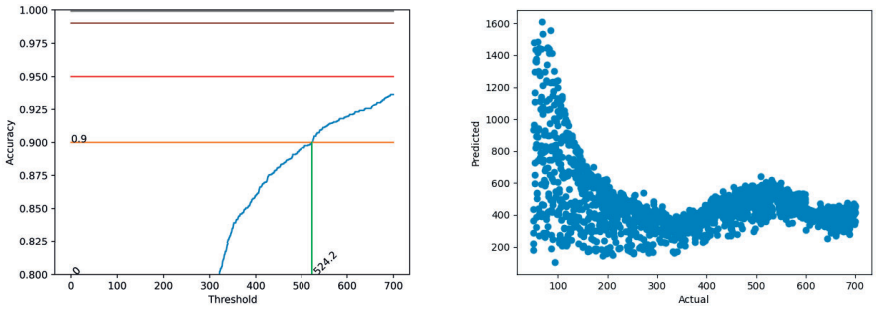
Linear and Gaussian regression models were made with the full data set of pulses. The preprocessing of the data was a simple fast Fourier transform and we used the same split as the final convolutional neural networks used. The Gaussian mean squared error was 175847.989 and the Linear regression mean squared error was 108446.424. As seen in the accuracy and scatter plots of the validation data for both models in Figures 4.8 and 4.10, these models were completely unusable.




**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 4.8:** Gaussian process regressor.




**(a)** percentage of errors that are less than a threshold (cm)

**(b)** correlation between predicted values (cm) and actual values (cm)

**Figure 4.9:** Linear regressor.

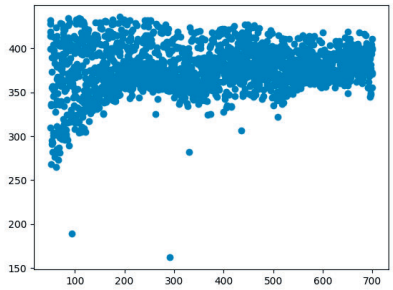
## 4.4 Volume Based Model

Even though we tried to control for the volume as a feature, we wanted to make sure that this was a success. This was explored by training regression models and our final convolutional neural network with just the volume of the signal. The validation loss for the three different models can be seen in Table 4.4. These models did not perform well.

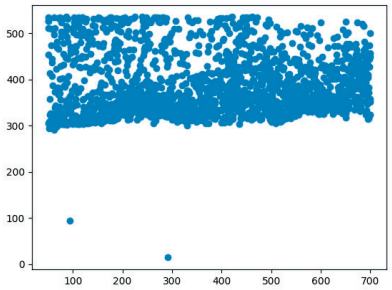
| Model type | Val loss |
|---|---|
| Linear regression | 34324.09 |
| Gaussian process regression | 39583.56 |
| CNN | 33867.42 |

**Table 4.4:** Validation loss for models based on volume.

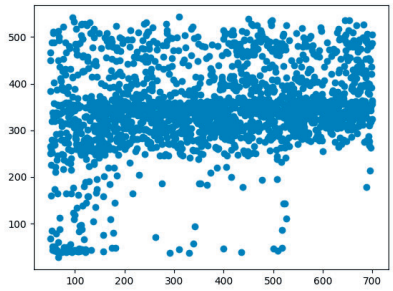
The relationships between the actual and predicted values can be seen in the scatter plots in Figure 4.10.




**(a)** linear regression




**(b)** Gaussian process regressor




**(c)** neural network

**Figure 4.10:** Scatter plots (in cm) for models based on volume.

# Chapter 5

# Discussion

## 5.1 Simple Models

Gaussian process and linear regression models were made with synthetic data and ideal data. The models performed well with their validation data but not with data from a more realistic setting. The Gaussian regression gave better results for both data types, which one would expect because of the non-linear properties of frequency attenuation.

### 5.1.1 Synthetic Data Models

The models created with the synthetic data were a first exploration of the concept of using frequency attenuation for distance estimation. These models were also made to test if a simple solution would be enough to solve this problem. The first models performing well was not very surprising, since they were simple and mathematically perfect models. Adding the noise was an attempt to approximate the complex changes that the geometry and materials in a real-world setting do to a sound signal. Both with and without noise Gaussian process regression performed remarkably better than linear regression.

However, the way sound interacts with the real world turned out to be way more complex than the simple noise model. This explains the very poor performance the first models had when using real-life data.

### 5.1.2 Ideal Data Models

The models created in the anechoic chamber validated the concept as something possibly doable. Frequency attenuation has a relativity small effect on a sound signal over short dis-

tances. But the model created in the anechoic chamber showed that this effect was measurable and could be used for distance approximation.

The Gaussian regression performed the best for both signals used. The best models tested in a real room setting had errors so large that they were unusable.

## 5.2 Creating the Robot

Given more time to complete the robot, it would have been a very useful tool for this project. In hindsight, given how many issues arose with the robot, we should have focused more on gathering data manually.

With small tweaks, the robot we built could have been useful. Using ball chains and a feeding gear instead of a rope or string would remove any slipping issues. This was the only known major issue that stopped the robot from being usable. This means the robot could become a useful tool for future research with just a small amount of work.

## 5.3 Final Neural Network model

### 5.3.1 Validation Threats

The results of the final model showed promise and should be enough to prompt further exploration of using frequency attenuation as a distance estimation tool. However our method includes sources of unreliability.

#### Final Data Set

The data set in this project was small, gathered in few different locations and with limited positions relative to the speaker. The test set was small and gathered in the same environment as the rest of the data. These are some of the reasons the results of the final model might not be reliable. More data would be needed to give a fuller picture of the applicability of the solution we have proposed.

#### Performance in a Noisy Environment

The data we gathered for the final models in this project did not have any significant environmental noise in them. This was done to limit the scope of the project, since we did not have the resources to gather the necessary amount of data, nor augment realistic environmental noise. This means that the performance of our proposed solution in a noisy environment is unknown.

#### Scope of Hardware

The data gathered in this project was collected with the same microphone and speaker. This limits the scope of the project, as the frequency responses of hardware might differ wildly.

This means that the machine learning models in this project might only perform similarly if the same models of the microphone and speaker are being used.

This variance in frequency response given different hardware would have to be taken into account if this method were to be implemented in a real-world solution.

### 5.3.2 Using Pulses for the Final Model

Our final model used pulses, as it was the signal with the consistently best performance. We found three different possible reasons for this. Firstly it could be because it was the shortest signal, which also meant the smallest input. All models had problems with overfitting, but it affected the pulse less. Secondly, it could be because we could limit the reverberations of the signal. The impulse was cut to mostly only include direct sound, which minimized reverberations and interference. The last reason could have been that the other signals we used might have been unsuitable for the purpose. There could possibly be some great signals that have been overlooked.

Some negatives of using only pulses are multichannel communication and scope. When communicating with waves one can use different frequency bands for different devices. This is however not possible with pulses as they include the full frequency spectra. If several signals were used as input to the model instead of just one it would expand the scope of possible implementations. A more general model could be trained on a lot of different sound signals, finding a general correlation between frequency attenuation and distance. However, this would require much more data and would be a big undertaking.

### 5.3.3 Number of Mel Frequency Bands

The best models we created used 12 Mel frequency bands, slightly outperforming the models created with 96 Mel frequency bands. The reason we used 96 Mel frequency bands in the beginning was simply because that was the required dimensions for the pretrained network we used. Testing a smaller number of frequency bands was done because previous research using MFFCs generally did not use many frequency bands. Reducing the number of frequency bands also reduced the complexity of the data, and this was generally helpful for training a more general model. Our results confirmed this, since using 12 frequency bands consistently outperformed 96 frequency bands.

## 5.4 Other Models With the Final Data Set

### 5.4.1 Models Based on Volume

The models using pure volume were created to see if we had properly isolated the frequency attenuation as the only usable feature for the models. Even though we normalized the data before feeding it to the models, we wanted to make sure that the preemptive measure of randomizing the volume in the data-gathering stage was enough to obscure the volume as a usable feature. Making sure the volume was properly obscured was important, because there

might have been some secondary feature based on the volume that we were not aware of. The results of the models based only on the volume did show that we succeeded with this.

### 5.4.2 Simple Regression Models

The results of the simple regression models with the final data sets showed that the distance approximation problem is too complex for simpler statistical models. Linear regression could not fit the data well, and the Gaussian process regression failed completely. This validates the use of a more complex machine learning model.

# Chapter 6

# Conclusion

The aim of this project was to use frequency attenuation to estimate the distance between a speaker and a microphone, with an error of less than a couple of meters. The final results were achieved with a convolutional neural network trained on a relatively small data set with limited real-world factors. Even with this limited data, these results showed promise that sound source estimation with an error of less than a few meters was possible with this method. The results of these experiments should be enough to prompt further research into this method.

## 6.1 Future Work

Future work based on the research in this thesis should include a lot more data gathering. Preferably the data should be gathered from many different environments, and a wide variety of speakers and microphones should be used. One way to accomplish this would be by finalizing the robot made in this project, as it only needs a few tweaks. Another solution would be to create more advanced synthetic data. Using a waveform-based model instead of an image-based CNN might also be a suitable focus for future work.

# References


[1] J. R. Healy B. P. Bogert and J. W Tukey. ""The Quefrency Analysis of Time Series for Echoes: Cepstrum, Pseudo-Autocovariance, Cross-Cepstrum, and Saphe Cracking," Proceedings of the Symposium on Time Series Analysis". In: 1963, pp. 209–243.

[2] Wen Chen and Sverre Holm. "Modified Szabo's wave equation models for lossy media obeying frequency power law". In: *The Journal of the Acoustical Society of America* 114 (Dec. 2003), pp. 2570–4. DOI: 10.1121/1.1621392.

[3] Ha Do, Weihua Sheng, and Meiqin Liu. "Human-assisted sound event recognition for home service robots". In: *Robotics and Biomimetics* 3 (Dec. 2016). DOI: 10.1186/s40638-016-0042-2.

[4] Kim Fluitt, Timothy Mermagen, and Tomasz Letowski. "Auditory Distance Estimation in an Open Space". In: *Soundscape Semiotics*. Ed. by Herve Glotin. Rijeka: IntechOpen, 2014. Chap. 7. DOI: 10.5772/56137. URL: https://doi.org/10.5772/56137.

[5] Dariusz Frejlichowski et al. ""SmartMonitor"— An Intelligent Security System for the Protection of Individuals and Small Properties with the Possibility of Home Automation". In: *Sensors* 14.6 (2014), pp. 9922–9948. ISSN: 1424-8220. DOI: 10.3390/s140609922. URL: https://www.mdpi.com/1424-8220/14/6/9922.

[6] Eleftheria Georganti et al. "Sound Source Distance Estimation in Rooms based on Statistical Properties of Binaural Signals". In: *IEEE Audio, Speech, Language Process.* 21 (Aug. 2013), pp. 1727–1741. DOI: 10.1109/TASL.2013.2260155.

[7] Eleftheria Georganti et al. "Speaker Distance Detection Using a Single Microphone". In: *IEEE Transactions on Acoustics Speech and Signal Processing* 19 (Sept. 2011). DOI: 10.1109/TASL.2011.2104953.

[8] Johann Huber. *Batch normalization in 3 levels of understanding*. URL: https://towardsdatascience.com/batch-normalization-in-3-levels-of-understanding-14c2da90a338#b93c. (accessed: 15.10.2022).

[9] Rohan Kapoor et al. "Acoustic Sensors for Air and Surface Navigation Applications". In: *Sensors* 18 (Feb. 2018), p. 499. DOI: 10.3390/s18020499.

[10] Taejun Kim, Jongpil Lee, and Juhan Nam. "Comparison and Analysis of SampleCNN Architectures for Audio Classification". In: *IEEE Journal of Selected Topics in Signal Processing* 13.2 (2019), pp. 285–297. DOI: 10.1109/JSTSP.2019.2909479.

[11] Heinrich Kuttruff. "Room Acoustics". In: Aachen, Germany: Taylor Francis e-Library, 2001, p. 19. ISBN: 0-419-24580-4.

[12] Vijaysinh Lendave. *What is Activity Regularization in Neural Networks?* URL: https://analyticsindiamag.com/what-is-activity-regularization-in-neural-networks/. (accessed: 15.10.2022).

[13] James Longman. *Understanding Speaker Frequency Response*. URL: https://www.audioreputation.com/understanding-speaker-frequency-response/. (accessed: 27.09.2022).

[14] Ella Morton. *How Long Could You Endure the World's Quietest Place?* URL: http://www.slate.com/blogs/atlas_obscura/2014/05/05/orfield_laboratories_in_minneapolis_is_the_world_s_quietest_place.html?via=gdpr-consent. (accessed: 22.10.2022).

[15] Douglas O'Shaughnessy. "Speech communication: human and machine". In: Addison-Wesley, 1987, p. 150. ISBN: 978-0-201-16520-3.

[16] M. Omologo, P. Svaizer, and M. Matassoni. "Environmental conditions and acoustic transduction in hands-free speech recognition". In: *Speech Communication* 25.1 (1998), pp. 75–95. ISSN: 0167-6393. DOI: https://doi.org/10.1016/S0167-6393(98)00030-2. URL: https://www.sciencedirect.com/science/article/pii/S0167639398000302.

[17] Ronald L. Panton. "Incompressible Flow". In: Hoboken, New Jersey, USA: John Wiley Sons, Inc., 2013, p. 114. ISBN: 978-1-118-01343-4.

[18] Md. Sahidullah and Goutam Saha. "Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition". In: *Speech Communication* 54.4 (2012), pp. 543–565. ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2011.11.004. URL: https://www.sciencedirect.com/science/article/pii/S0167639311001622.

[19] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: (2018). DOI: 10.48550/ARXIV.1801.04381. URL: https://arxiv.org/abs/1801.04381.

[20] Ignacio Spiousas et al. "Sound Spectrum Influences Auditory Distance Perception of Sound Sources Located in a Room Environment". In: *Frontiers in Psychology* 8 (May 2017). DOI: 10.3389/fpsyg.2017.00969.

[21] George Gabriel Stokes. "On the Theories of the Internal Friction of Fluids in Motion, and of the Equilibrium and Motion of Elastic Solids". In: *Mathematical and Physical Papers*. Vol. 1. Cambridge Library Collection - Mathematics. Cambridge University Press, 2009, pp. 287–342. DOI: 10.1017/CBO9780511702242.005.

[22] tec-science.com. *Viscosity of liquids and gases*. URL: https://www.tec-science.com/mechanics/gases-and-liquids/viscosity-of-liquids-and-gases/#Temperature_dependence_of_viscosity. (accessed: 23.10.2022).

[23] *The Propagation of sound*. URL: https://pages.jh.edu/virtlab/ray/acoustic.htm. (accessed: 26.09.2022).

[24] ThisIsEngineering. Pexels. July 2019.

[25] Iowa state university. *Temperature and the Speed of Sound*. URL: https://www.nde-ed.org/Physics/Sound/speedinmaterials.xhtml. (accessed: 23.10.2022).

[26] Berkeley University of California. *Impulse response*. URL: https://ptolemy.berkeley.edu/eecs20/week11/response.html. (accessed: 22.10.2022).

[27] Barry Van Veen. *The Spectrum: Representing Signals as a Function of Frequency*. Youtube. 2018. URL: https://www.youtube.com/watch?v=xh5pLY3f6H8.

[28] Costas Yiallourides et al. "Acoustic Analysis and Assessment of the Knee in Osteoarthritis During Walking". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 281–285. DOI: 10.1109/ICASSP.2018.8461622.

[29] M. Yiwere and E.J. Rhee. "Distance estimation and localization of sound sources in reverberant conditions using deep neural networks". In: *International Journal of Applied Engineering Research* 12 (Jan. 2017), pp. 12384–12389.

[30] Mariam Yiwere and Eun Rhee. "Sound Source Distance Estimation Using Deep Learning: An Image Classification Approach". In: *Sensors* 20 (Dec. 2019), p. 172. DOI: 10.3390/s20010172.

[31] Li Zhiming and Miao Sheng. "Heart sound recognition method of congenital heart disease based on improved cepstrum coefficient features". In: *2021 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI)*. 2021, pp. 319–324. DOI: 10.1109/ICCEAI52939.2021.00064.

Printed by Tryckeriet i E-huset, Lund 2023


LUND
UNIVERSITY

Series of Master's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2023-906
http://www.eit.lth.se