Theodoros Papaiakovou
Matriculation number S1720541
BEng Hons Project Report
<Deep Neural Networks for Microphone
Array Direction of Arrival Estimation>
28 April 2021

Mission Statement

Project Definition

The aim of this project is to design a deep neural network model that can estimate the direction of arrival (DoA) of multiple sound sources in real time. A 6-microphone array (ReSpeaker Core v2.0) is to be used that can do deep beamforming to increase selectivity of the sound sources and therefore potentially improve the accuracy. Then the network should be tested, optimized and implemented on an ARM based embedded computer.

Preparatory Tasks

- Learn more about machine learning.
- Learn how to use TensorFlow or PyTorch better.
- Research how beamforming and microphone arrays work.
- Find what type of neural network and what pre-processing of inputs would be most suitable.
- Study the structure of ReSpeaker Core v2.0 6-microphone array and its embedded ARM processor.

Main Tasks

- Test different deep neural networks to predict the DoA of multiple sound sources (more than one).
- Design a deep beamformer to increase the accuracy of the DoA estimation.
- Implement the network on an ARM based embedded computer.
- Achieve real time (or close to real time) performance.
- Build and test the network using the ReSpeaker Core v2.0 6-microphone array.
- Find or create accurate training data for the neural networks.

Scope for Extension

- Further increase performance.
- Increase the number of sound sources.

Abstract

The scope of this project is to implement a real-time deep neural network based direction of arrival (**DOA**) estimation algorithm, that can localize multiple sound sources, in an embedded computer. The symmetry of a 6 microphone uniform circular array (**UCA**) was exploited by finding a reference microphone closest to a source using beamforming and inputting the generalized cross correlation with phase transform (**GCC-PHAT**) matrix, for all microphone pair combinations, in a multilayer perceptron (**MLP**) that was trained to predict the DOA in 60 degrees. This way all 360 degrees are covered but the classification task of the MLP is reduced to 60 classes relaxing the training data requirement. The algorithm achieved reasonable results for one source, was able to run in real time and outperformed **MUSIC**, a conventional DOA estimation algorithm, in terms of accuracy. It was shown that this method has the potential to perform fast and robustly in low SNR environments with a minimal amount of training data. Ideas for extension that could improve the performance as well as insights on the multiple source scenario are given. A Python implementation of this algorithm is included.

Declaration of Originality

I declare that this thesis is my
original work except where stated.
Theodoros Papaiakovou

Statement of Achievement

While working towards this project, despite the limited conditions of having no access to the labs due to the COVID-19 pandemic, I considerably improved my skillset in a variety of electrical engineering fields. I learned about machine learning, I learned how to apply signal processing techniques such as beamforming on microphone arrays, I learned how to use Linux environments and significantly improved my skills in using Python.

More specifically, I created and trained two different types of neural networks. I simulated artificial audio data and learned how to extract the direction of arrival information from them. I created code that does beamforming to a 6 microphone circular array and developed an algorithm that exploits the symmetry of the array to make training a neural network possible with a limited dataset. Python scripts with this algorithm were implemented in an embedded ARM computer. The algorithm can run in real time and achieved reasonable results in localizing a single source.

Contents

Glossary

AI Artificial intelligence.

AOA Angle of arrival.

API Application programming interface.

CNN Convolutional Neural Network.

DOA Direction of arrival.

DSB Delay and sum beamformer.

FFT Fast Fourier transform.

GCC Generalised cross-correlation.

GCC-PHAT Generalised cross-correlation with phase transform.

HL Hidden layer.

HRI Human robot interaction.

ISM Image source method.

MLP Multilayer perceptron.

MPEG Moving Picture Experts Group.

MUSIC Multiple signal classification.

NN Neural Network.

PHAT Phase transform.

PSB Phase shift beamformer.

SAC Spatial audio coding.

SDB Superdirective beamformer.

SNR Signal to noise ratio.

TDOA Time delay of arrival.

UCA Uniform circular array.

VAD Voice activity detection.

Chapter 1

Introduction and Background

1.1 Motivation

The cocktail party effect is the term given to the problem of filtering out a particular auditory stimulus, like the voice of a speaker, from multiple stimuli and background noise, like the conditions present in a cocktail party. Human brains can naturally make that separation by adjusting the acoustic capabilities of the two ears depending on the target location [1].
Using microphone arrays, machines can do beamforming a technique that increases the spatial selectivity of the array and the received signal to noise ratio (**SNR**). However, to do beamforming machines need to know the **DOA** of the sound sources in advance which, unlike for humans, is not trivial. Multiple techniques that perform DOA estimation have been researched with the most popular being beamforming methods, subspace-based methods [2] and time difference of arrival (**TDOA**), many times referred to as time delay of arrival, based methods but tend to suffer in low SNR and reverberant conditions. More modern approaches attempt to use neural networks (**NN**) to extract the DOA information of single [3, 4, 5] and multiple [6] sound sources from the audio inputs of a microphone array with interesting results but, the challenge of using NN is the numerous training data that they require. An important consideration for which algorithm to use is the geometry of the microphone array with common options being linear, rectangular, circular and spherical. In this paper a **UCA** geometry is used because it was shown to perform well for DOA estimation tasks [3, 6, 5, 7] and because when mounted on a cylindrical baffle outperformed all other array geometries in terms of resolution, robustness to environmental noise and directivity [8]. Even though research on the effect of the number of microphones on the number of detectable sources has not been made to my knowledge, it was suggested that the number of sources should be smaller than that of the microphones [7] and that the number of microphones determines the frequency response of the beamformer, with more microphones increasing the upper frequency limit [9]. In addition, the circular symmetry of the UCA allowed for the novel 60-degree implementation proposed that reduces the number of training data required by the neural network. DOA estimation can include both the azimuth and elevation angles but in this work only the azimuth was considered. The multiple sources scenario was attempted and insights on its implementation is given in the section 3.5. The potential

impact as well as real world applications of a complete multi-source DOA estimation algorithm is given in chapter 2.

1.2 Beamforming

Beamforming is achieved by combining the inputs of an array in a way that signals arriving from a particular direction interfere constructively while interference from the other directions interferes destructively. This can work with any signal input to a sensor array but this work focused on acoustic signals. Some common methods to do beamforming are the delay and sum (**DSB**) beamformer, the superdirective beamformer (**SDB**), the least-squares beamformer (LSB) and the minimum variance distortionless response (MVDR) beamformer. Out of these methods, SDB has the best performance in terms of SNR improvement with the LSB coming second [8] and the DSB being more robust in noisy environments [9].

1.2.1 The Delay and Sum Beamformer

The DSB also known as the conventional beamformer and as the Bartlett beamformer, is based on delaying the input of each microphone by an appropriate amount of time, determined by the steering angle of the array, and summing all of the inputs together as seen in equation 1.1.

$$y_{DSB} = \sum_{i=1}^{M} x_i(t - t_i) \tag{1.1}$$

Where y_{DSB} is the beamformed signal, x_i is the input of microphone i and t_i is the time delay applied to microphone i. A narrowband approximation of the DSB is called the phase shift beamformer (PSB), which utilizes the fact that a time delay in the time domain is a phase shift in the frequency domain, and simplifies the DSB implementation by multiplying the frequency domain representation of the input signals with complex beamforming weights as shown in equation 1.2.

$$Y_{PSB} = \sum_{i=1}^{M} w_i X_i(f) \tag{1.2}$$

Where Y_{PSB} is the **FFT** of the beamformed signal, X_i is the FFT of the i_{th} microphone input and w_i is the complex beamforming weight. This achieves a spatial selectivity in 360 degrees like the one seen in figure 1.1.

1.3 Conventional DOA Estimation

1.3.1 Beamforming Methods

Beamforming methods obtain a spatial power spectrum by beamforming in multiple directions and localizing a source based on which direction holds the maximum power. To estimate the DOA for multiple sources the largest peaks in the spatial power spectrum, that correspond to the number of sources, are

found. Any beamforming technique could be used for DOA estimation with a popular one being the MVDR beamformer. The sidelobes of the beampattern limit the resolution and the estimation accuracy that can be achieved from a beamforming technique [2].

1.3.2 Subspace Methods

Subspace methods decompose the signals covariance matrix to its eigenvectors and eigenvalues. The N largest eigenvalues, where N is the number of sources, correspond to the eigenvectors of the signal that form the signal subspace which is orthogonal to the rest of the eigenvectors that correspond to the noise subspace. Popular techniques include the multiple signal classification (MUSIC) and the estimation of signal parameters via rotational invariance techniques (ESPRIT). Their limitations for the multi-source scenario is that they require the number of sources to be known in advance so that the correct number of signal eigenvalues are chosen and that they work only for uncorrelated and incoherent signals [2].

1.3.3 TDOA Methods

TDOA methods are based on finding the TDOA of the source signal for the different microphone pairs of the array as an incident sound signal will reach each microphone at a slightly different time depending on the angle of arrival (**AOA**). By using techniques such as the **GCC**, it is possible to find the TDOA for each microphone pair and then combine this information with the position of the microphones to locate the DOA. Figure 1.2 helps to illustrate that. By knowing the speed of sound c and the TDOA $T_{i,j}$ we can use equation 1.5

$$t_i = \frac{d_i}{c} \tag{1.3}$$

and

$$T_{i,j} = t_i - t_j \tag{1.4}$$

$$= \frac{d_i - d_j}{c} \tag{1.5}$$

If we solve for d_i using all the microphone pairs we can find the distance of every microphone from the source which with the positions of microphones m_i, m_j and therefore the relative distances between them $dm_{i,j}$, maps to a particular DOA.

The GCC is effectively the cross-correlation function of the input signal between two channels calculated using the inverse Fourier transform of the cross power spectrum. Mathematically this is expressed as:

$$GCC_{i,j} = \int_{\infty}^{\infty} \psi X_i(f) X_j(f) e^{j2\pi ft} df \tag{1.6}$$

$X_i(f)$ and $X_j(f)$ are the frequency spectra of the inputs of m_i and m_j respectively. ψ is a weighting function that is used to boost the SNR at a frequency of the power spectrum. A popular weighting

function is the **PHAT**

$$\psi = \frac{1}{|X_i(f)X_j(f)|} \tag{1.7}$$

whose effect on the GCC is discussed in section 3.2.2. The peak of this function occurs at the time delay where the signals $x_i(t)$ and $x_j(t)$ are the same which is the TDOA [10]. The GCC function shifted by the TDOA is effectively the autocorrelation function of the source signal plus noise. When the noise is uncorrelated to the signal clear peaks are seen, however when this noise origins from reverberations of the source signal, large peaks that mask the true TDOA may be present and retrieving the true TDOA from the location of the largest peak is not possible. A technique that combines finding the TDOA with the GCC and iteratively searching space for multi- source DOA estimation, called steered response power with phase transform (SRP-PHAT) as well as a proposed variation of it called singular value decomposition (SVD-PHAT) is presented in [11]. Kwon, Youngjin Park and Youn-sik Park [10] showed that for two sources, another peak in the **GCC-PHAT** function corresponds to the TDOA of the second source and its ratio with the main peak correlates to the power ratio of the two sources. In addition, the probability that the second largest peak corresponds to the second source was found to be about 50% and criteria like the ratio of the 1_{st} and 2_{nd} peaks and the 2_{nd} and 3_{rd} peaks can give an evaluation of how valid this assumption is. Another technique based on deep learning uses the GCC-PHAT separated in Mel scale filterbanks, an audio transformation that applies a triangular Mel filter at different frequency ranges to simulate human hearing, because signals from different sources usually do not have the same GCC patterns in different filter banks and accurately estimates the DOA for up to two sources [6].

1.4 Neural Networks

1.4.1 Overview

Artificial Intelligence (**AI**) is one of the most thriving areas of research of the 21st century and is starting to be deployed in an increasing amount of applications. Newer approaches to AI called machine learning use models composed of many layers that can extract information or features from raw data. These layers are made up of nodes that were inspired by neurons in the human brain which is one of the reasons that led to the name artificial neural networks [12, chapter 1]. Each of these nodes operates by getting an input, multiplying it by a weight and adding a bias to it, then performing a transformation by a nonlinear activation function to be able to map nonlinear outputs [12, chapter 5]. By connecting many of these nodes in a layer, and stacking many of these layers, hence the name Deep Learning, very complex functions can be created. The layers found between the input and output layers are called hidden layers (HL) and extract more abstract features from the input data [12, chapter 1]. Using a technique called back-propagation a NN can find the loss between the parameters of the model and the output value and alter the weights applied to each node to minimize this loss and learn [13]. The Universal approximation theorem states that a NN can learn any possible function regardless of complexity if they have enough parameters, but they sometimes perfectly learn how to represent the training data without being able

to generalize their predictions to the real data, a problem that is named overfitting. By providing a validation dataset the model can have a reference of overfitting and try to update its hyper-parameters to better generalize to non-training data [12, chapter 5]. Deep NN composed of layers with nodes fully connected to each other are named **MLP**s and are the most basic type of feedforward deep NN that can learn to output values from the input data. Other types of deep NN include convolutional neural networks (**CNN**s), that accept multiple arrays of data and apply weights called filters, with a discrete convolution operation, to different areas of the input to extract its features. Recurrent neural networks (RNN) are good at solving sequential tasks as the previous model outputs are saved in a **HL** and are taken into account in the next iteration [13].
Machine learning can be separated in supervised learning, unsupervised learning and reinforcement learning. Supervised learning models are given the input data as well as labels that map them to an output and are trained to predict values of the output mapping. Unsupervised learning models are fed the dataset and try to learn the underlying features or cluster the inputs based on similarity. Reinforcement learning involves putting a model called agent in an environment, observing the state of the environment and taking an action. This action produces a change in the state of the environment which is rewarded positively or negatively depending on the desired outcome and learns by trying to take actions that maximize the rewards [12, chapter 1, 5].

1.4.2 Training NNs for DOA Estimation

NNs for DOA estimation have been used with many successful examples. Most researchers use CNNs to extract the spatial features from GCC or discrete Fourier transform (DFT) inputs [4, 14, 5, 6] while some use MLPs [3, 6, 15] with both types being able to achieve similar results. All of these papers mention that the greatest limitation is acquiring enough training data especially in multi-source approaches as the number of sources increases. Assuming that we are trying to achieve a 1 degree resolution, each source could be in 360 different degrees. The way that the sources can be distributed in the 360 degrees is a combination with 360 elements and k selections, where k is the number of sources. The equation of the number of combinations with respect to k is

$$360Ck = \frac{360!}{k!(360-k)!} \tag{1.8}$$

whose growth rate depends on the k! factorial and the diminishing (360-k)! factorial. As shown in figure A.1 of appendix A the initial scaling of the number of combinations is almost exponential for small values and decreases as it gets closer to 180. Since the number of sources should be smaller than the number of microphones of the array [7], the number of different source combinations that would be required to train the AI for up to 5 sources are:

- $k_1 = 360$
- $k_2 = 64620$
- $k_3 = 7711320$

- $k_4 = 688 * 10^6$
- $k_5 = 4.9 * 10^{10}$

Moreover, as the number of combinations increases the complexity of the NN increases and more data per combination will be required for training. Even with modern sophisticated model architectures that have the required capacity, the computing power required to train such a network and obtain this number of training data limits the scalability of the arbitrary number of sources scenario in practice.

1.5 60-Degree Implementation

A technique was proposed that exploits the symmetry of the 6 microphone UCA and combines beamforming with a NN to estimate the DOA. By beamforming on the direction of each microphone from the center of the array and finding the power held at each direction the microphone closer to the source is estimated, which is how the beamforming DOA estimation methods mentioned above work. Even though beamforming methods have limited resolution they should be able to locate which microphone is closer to the source in 60 degrees. The NN can be trained to predict the DOA to any desired resolution within the 60 degrees region covered by a microphone that will be used as reference for training. Combining the information of which is the reference microphone obtained by beamforming with the DOA estimation of the NN for the region covered by that reference microphone, the location of a sound source can be found in 360 degrees. Figure 1.3 shows how this symmetrical approach works. Considering source 1, by beamforming in directions bd_i, the power of bd_1 is found to be greater as source 1 lies closer to that direction. This indicates that the reference microphone is m_1 and the NN makes a prediction in the 60-degree region R_1. If source 2 is present instead, the direction bd_2 has the most power, m_2 is classified as the reference microphone and the NN predicts the DOA for R_2. Even if the adjacent microphone is falsely selected when the direction of the sound source is between two microphones, the NN will predict a direction close to their boundary resulting in an error of only a few degrees. This technique has the advantage of greatly relaxing the training data and capacity requirement of the NN as only 60 degrees have to be classified instead of 360 to get a 1 degree resolution.

1.6 Multiple Sources Scenario

Localizing multiple sound sources has been achieved with three different methods. One implementation is to use a sigmoid activation function for the output layer of the NN, which in DOA classification tasks outputs the probability that the input belongs to each direction, and set a probability threshold above which we consider that there exists an active source in that direction [14]. Another implementation is to find the DOA of the dominant sound source using the GCC-PHAT input and then null the time delay regions of the GCC-PHAT function that correspond to that predicted direction. Then the DOA is estimated for the modified GCC-PHAT input that has the effect of the first source removed and by iteratively following this method the DOA of any number of sources can be found [11]. However, this

method must be complemented by an accurate source counting technique and the validity of considering the second peak of the GCC-PHAT function is discussed in subsection 1.3.3. The other possible method is to use a special output encoding for the output labels of the NN that uses Gaussian functions with mean the direction of a source and train the network for all different combinations of the number of sources [6]. Their applicability in the 60-degree implementation mentioned above is investigated in section 3.5.

1.7 Specific Objectives

The aim of this project is to develop a DOA estimation algorithm for multiple sources using deep NN. Firstly, a voice activity of detection (VAD) algorithm is a prerequisite block to DOA estimation and is discussed in section 3.1. Then, the representation of the training data is considered in subsection 3.2.2 and the data simulation and recording are covered in subsections 3.2.3 and 3.2.4. The architecture and the training procedure of the NNs that estimate the DOA in 60 degree regions is considered in section 3.3.2. The beamforming algorithm that makes the 60-degree implementation possible is presented in section 3.4. Insight on the limitations of the 60-degree implementation for the multi-source scenario is given in section 3.5. Experiments on the performance of the algorithm for different conditions and their results are shown in chapter 4. Finally, the experimental results are analyzed, the algorithm is reviewed and potential steps for extension are given in chapter 5.

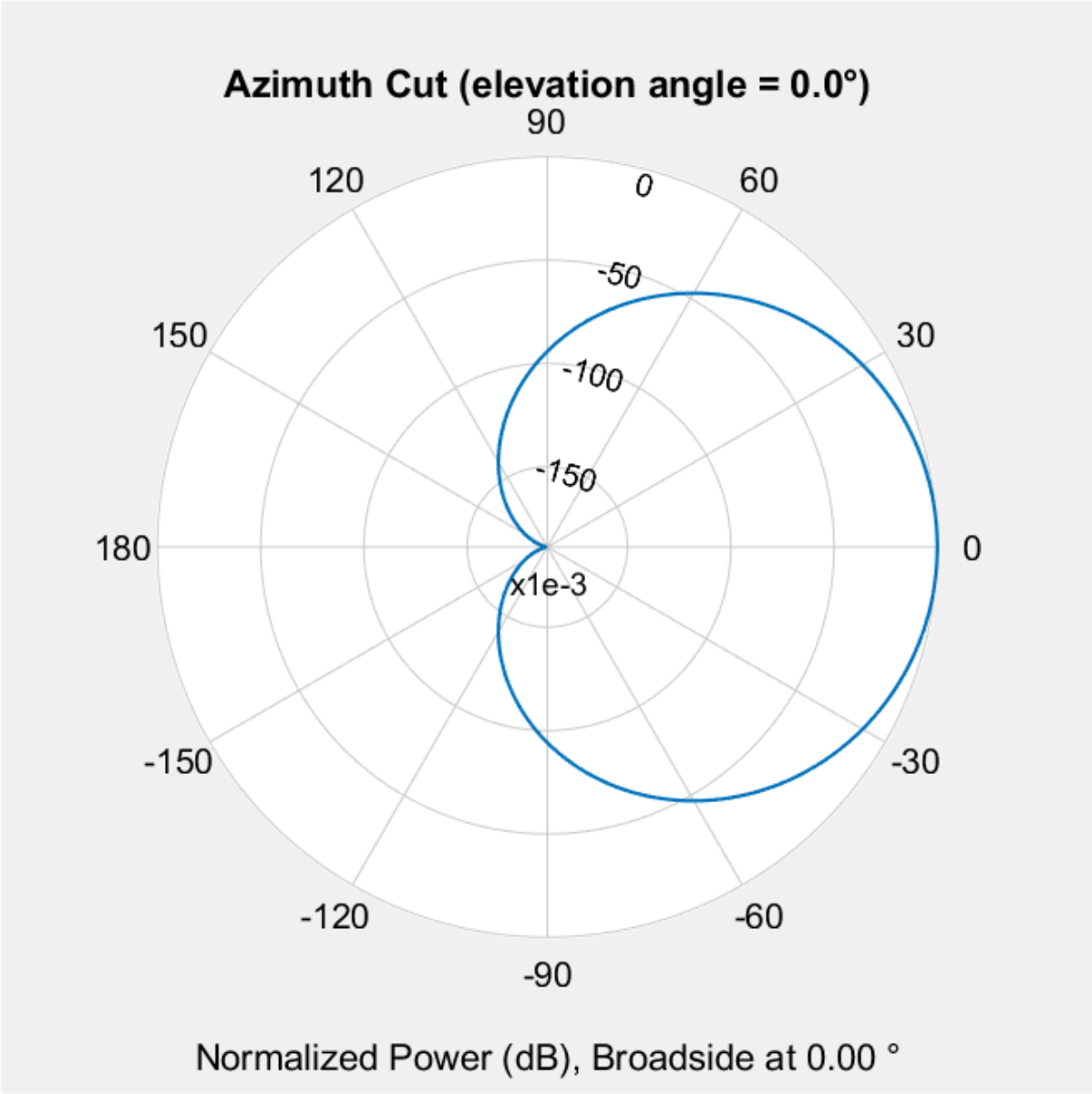


Figure 1.1: Beampattern for phase shift beamformer at direction 0 degrees

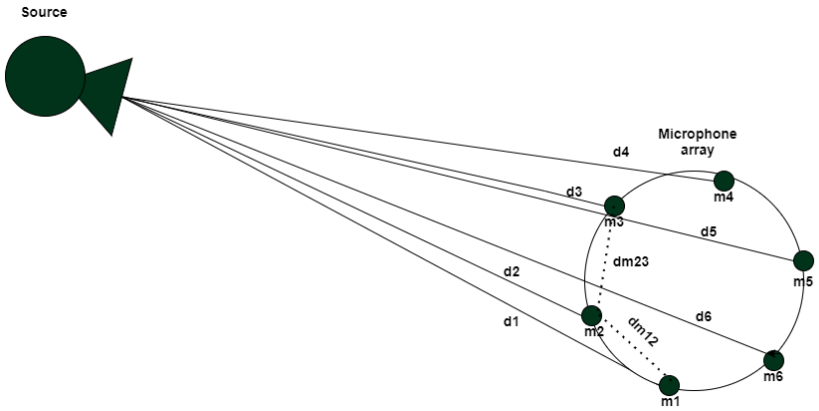


Figure 1.2: Diagram that illustrates the relationship of the DOA of a sound source to the TDOA and the microphone UCA geometry

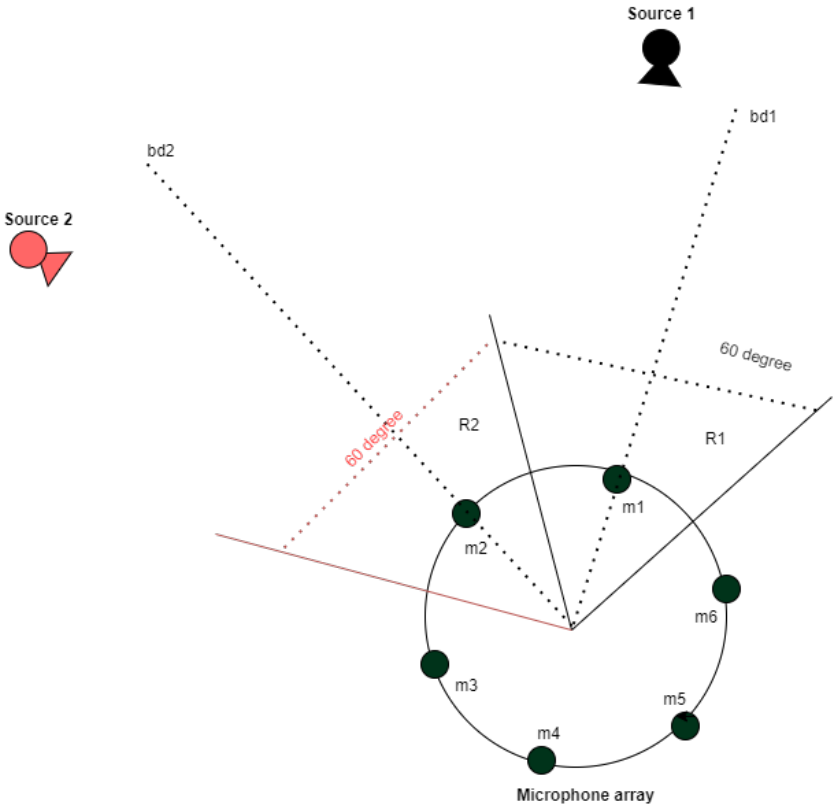


Figure 1.3: Diagram that illustrates how the 60 degree implementation is able to localize sources in 360 degrees

Chapter 2

Impact and Exploitation

2.1 Introduction

With modern technological advancements such as increased computing power and easy access to enormous amounts of data it has become feasible to implement AI algorithms in many of the devices that shape our daily lives. Deep neural networks are making breakthroughs in tasks like computer vision and web searches [13] and can act as an alternative to older algorithms that are limited by slow performance and assumptions on the operating environment. In this work an algorithm that uses signal processing and a neural network to determine the DOA for multiple sound sources was researched, developed and implemented in an embedded computer. Such a method will immediately have an impact in video teleconferencing, home assistants, human-robot interaction (**HRI**) [16], speech enhancement and spatial audio coding (**SAC**) [17] while the same idea can be extended to antennas and be used in the defense industry. This chapter investigates how this project fits in the research and development chain of a commercial product and how its completion could affect the aforementioned industrial sectors.

2.2 Project Development

The technology behind DOA estimation is saturated with conventional algorithms covered in section 1.3 of the introduction, that suffer from susceptibility to noise and reverberation and high computational complexity preventing them from achieving accurate results in real time implementations. The idea of using neural networks to map the location of sound sources has been attempted by multiple researchers whose findings prove that it is a viable improvement for real time applications in noisy environments. This work builds on this idea by researching the possible methods to extract the DOA information from sound data with deep learning models and demonstrates with a prototype an algorithm that estimates the DOA of a single source in practice. Developing a method that can accurately and reliably localize multiple sound sources and can easily be inserted to an embedded device could be directly used in devices such as home assistants and would enhance many processes that rely on accurate spatial localization.

2.3 Areas of Impact

The nature of this project makes its utilization flexible as the algorithm that was developed could be used for the development of a product or the actual device could be used to obtain the spatial information to realize another process. For example, home assistants and robots that operate with speech recognition rely on DOA estimation to find the direction of the speaker and do beamforming to suppress noise or interference coming from other directions. This way the speech signal of the end-user is enhanced making speech recognition possible [15]. In addition, accurate DOA information was noted to improve HRI as the robot can use the spatial information to make interactions feel more natural (e.g. the robot facing the speaker) and the authors explicitly stated their interest in improving the multi-source DOA estimation [16]. An extension of the algorithm that uses an antenna UCA could be used to localize and enhance signals having applications in wireless communications, search and rescue missions and national defense. For instance in defense, the radio waves transmitted by intruding aerial vehicles could be used to map their location. These applications will obviously require training of the neural network with the corresponding input radio data of the antenna array. Another field that benefits from this work is multiparty video teleconferencing where the DOA information acquired from a device could be used in the camera steering controls so that the video is automatically focused on the active speaker. Moreover, SAC techniques like **MPEG** that compress multichannel audio data need to estimate the spatial features of the original audio so that it can be used when reconstructing the multichannel signal [17]. The findings of this project could become a standardized tool for doing SAC which can improve multiple growing sectors like film, music, home cinemas and gaming. Finally, the multichannel audio dataset that must be recorded for training the neural network could be made available for future researchers to use.

2.4 Amount of Impact

The Areas of Impact section showed that a refined multi-source DOA algorithm could become a fundamental tool, adopted across a wide variety of the largest industrial sectors. In the short term enhancing the quality of speech recognition of home assistants, an industry that had a market value of 11.8 billion US$ in 2018 [18], would be of interest to the leading AI assistant companies. In addition, the completed device could facilitate SAC techniques essential for audio data compression in film, music and gaming that account for the largest part of the growing entertainment industry and were valued at approximately 200 billion US$ revenue in 2020 [19, 20, 21]. In the long term, the algorithm could contribute to the development of the robotics and automation industry and be found inside any robot that interacts verbally with humans. Successful results of this work could result in superior digital audio commodities that would improve social welfare.

2.5 Summary

To summarize, this project researches and develops an algorithm that uses AI to estimate the DOA of a single source and a prototype is made to demonstrate its functionality. A continuation of this algorithm

that could dependably and accurately localize multiple sources could become a standardized tool that aids companies in a variety of industrial sectors while improving social welfare. Meanwhile, this research could lay the foundation for more advanced academic research. Exploiting this scientific knowledge could be an enrichment to the technological arsenal of humanity.

Chapter 3

Methods

3.1 Voice Activity of Detection

To achieve real time application, accurate **VAD** is needed to feed only voiced segments to the DOA algorithm and to prepare the dataset for simulations. The WebRTCvad Python **API** was selected for this task, that was developed by Google for the Webrtc project which aids developers create real time web communication services. The VAD can work for 8, 16, 32 and 48 kHz inputs, can accept time frames of 10,20 and 30 ms and has an "aggresiveness" parameter that determines the VADs sensitivity. By inspecting the source code [22], the probability that an audio segment is voice or background noise is found by Gaussian mixture models and a hypothesis test classifies it as one of the two. A function was created that takes an audio segment and returns the voiced parts for the data generation (Appendix B) and used the API to predict the DOA of a source only when the input signal contains voice in the experiments of chapter 4.

3.2 Data generation

3.2.1 Overview

In order to train the neural network to perform DOA estimation, a substantial amount of training data is required. Goodfellow [12, chapter 1, p. 20-21] suggests that supervised learning requires approximately 5000 training data per class and that exceptional performance can be achieved with 10 million. The three possible choices to get access to these data were to simulate them, to record real data or to find an existing dataset. Recording the data has the benefit that the model will be trained to data similar to the ones that it will need to classify, which should give the model the best performance, but has two main limitations. Firstly, recording voiced data requires a lot of time especially as the number of classes, that in this case correspond to the resolution, increases. Secondly, a decrease in performance would be expected if the microphone array was placed in a different room or even position in the same room as the one where the training data were recorded. This would occur because the rooms impulse response would be different

and the model might not generalize well to the new input. Another option would be to find a suitable dataset of prerecorded data to use that would provide immediate access to enough data for training. The limitations of this approach are that the prerecorded dataset needs to be recorded in a microphone array like the one used, in this case the Respeaker core v2.0 [23] (more information in Appendix C) and that performance could be bad depending on the conditions of the room that they were recorded in. It is important that the recording equipment is a UCA with 6 microphones and radius of 0.0463 meters, so that the TDOAs between different microphone combinations are the same. I was not able to find any available prerecorded audio data that are compatible with the Respeaker. A middle solution is to simulate the training data using software which offers the ease of being able to create a bulk of audio data that are compatible with the Respeaker and gives the freedom to select different parameters such as the amount of reverberation, the number and position of sources and microphones and the received SNR. However, the simulations still require an amount of time to be made and may still not perform well in the actual room that the device is deployed.

3.2.2 Feature extraction

The audio data of the channels of the microphone array need to be processed in a form that represents the DOA information before being fed to the NN. A research paper [14] inputs the discrete Fourier transform (DFT) of every channel to a CNN to extract the spatial features. This method however does not directly show the DOA information and will make it more difficult for the CNN to match the DFT features to spatial information, requiring more training data and a more complex NN with increased capacity. Going one step further, research papers use the GCC [4] and the GCC-PHAT [3, 5, 6, 11, 15] functions whose peaks are the TDOA of the input signal for the different microphone combinations that hold the information to localize the source. The PHAT weighting makes the TDOA peaks much more prominent which aids the neural network with the feature extraction and performs better in reverberant conditions. To illustrate this point figure 3.1 shows a plot of both the GCC and the GCC-PHAT functions of the input of one microphone combination for a simulated audio signal.

They both have a maximum at a time delay of -0.109375 ms but the TDOA in the GCC-PHAT (orange) is much more prominent rendering it a more suitable choice to use as input of the NN. Python code to calculate the GCC and the GCC-PHAT can be found in Appendix B. In order to reduce the dimensionality of the GCC-PHAT input, the maximum TDOA that an audio signal could have between two microphones was found. The Respeaker microphone array has a diameter d of 0.0926 m which is the maximum distance between two microphones and the speed of sound in air c, was assumed to be 340 m/s. The max TDOA can be calculated by equation 3.1

$$TDOAmax = \frac{d}{c} \tag{3.1}$$

which in this case is equal to 0.272 ms, approximately equal to 0.3 ms. To find how many time delay values to keep, the sampling frequency (fs) chosen for the microphones as well as the resolution of the inverse FFT need to be taken into account. The FFT is computed faster for input sequences that are a power of 2, and zero padding can be used to increase the resolution of the GCC-PHAT function where a

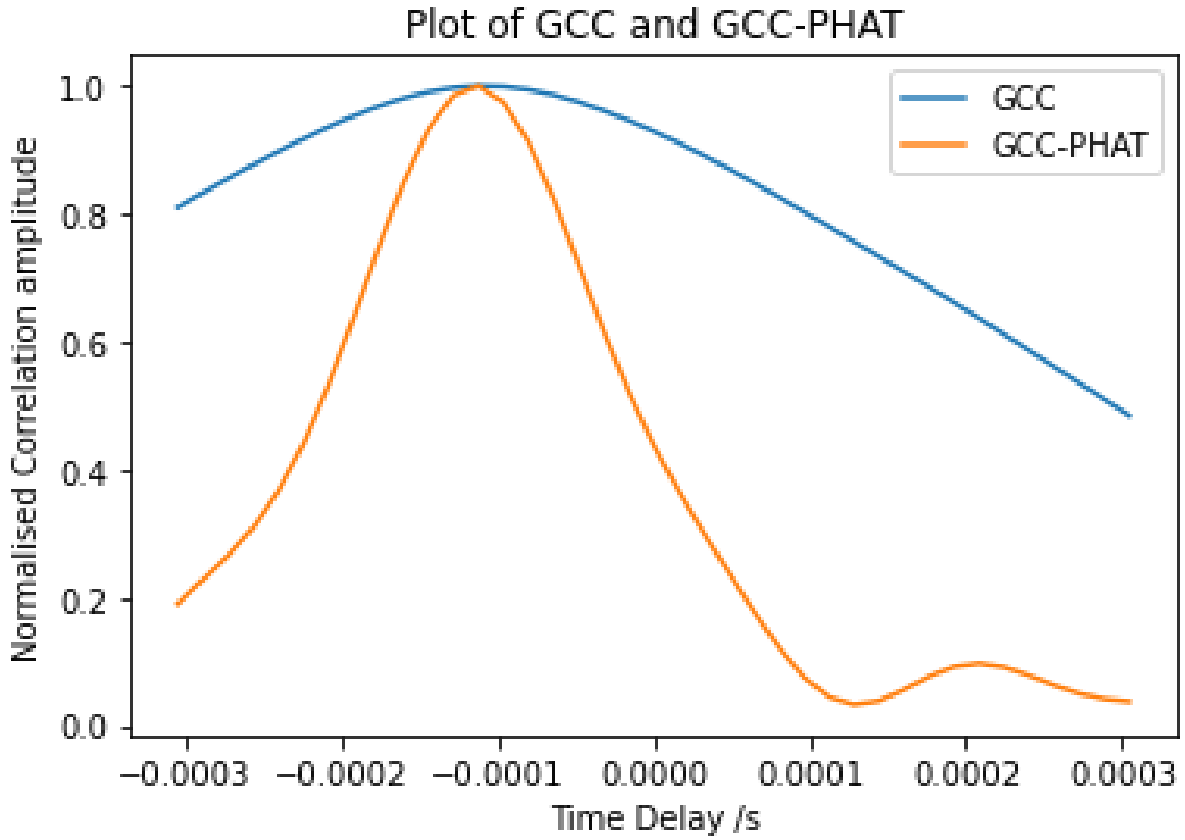


Figure 3.1: GCC and GCC-PHAT functions

higher resolution should make the DOA estimation more accurate in reverberant conditions as the actual TDOA peak might be masked by noise and the extra information will help the NN to make a correct prediction. The fs of the input is equal to 16000 Hz and the FFT resolution is increased by a factor of 4 so it requires $16000 * 0.3 * 10^3 4 = 19.2$ values for the 0.3 ms time delay. The actual dimension for the GCC-PHAT is $19 * 2 + 1 = 39$ because 19.2 rounds to 19 since we cannot have a decimal amount of values, we multiply by 2 since the value can be negative 0.3 ms as well and add 1 to make it odd as the time delay has to be symmetric about 0.
To extract the DOA, the TDOA for all different pairs of M microphones is needed and the number of different pair combinations can be found from equation 3.2.

$$Microphonepaircombinations = \frac{M(M-1)}{2} \tag{3.2}$$

Since we have 6 microphones the total number of combinations is 15. By combining all the GCC-PHAT vectors in 39x15 matrix form we represent them in an image like the one shown in figure 3.2. The y axis is the time delay and the x axis shows the different microphone pairs with the yellow spots representing the

TDOA. These matrices were used as inputs to the neural network because they hold all the information required to localize one source. Python code to generate these images is found in Appendix B.

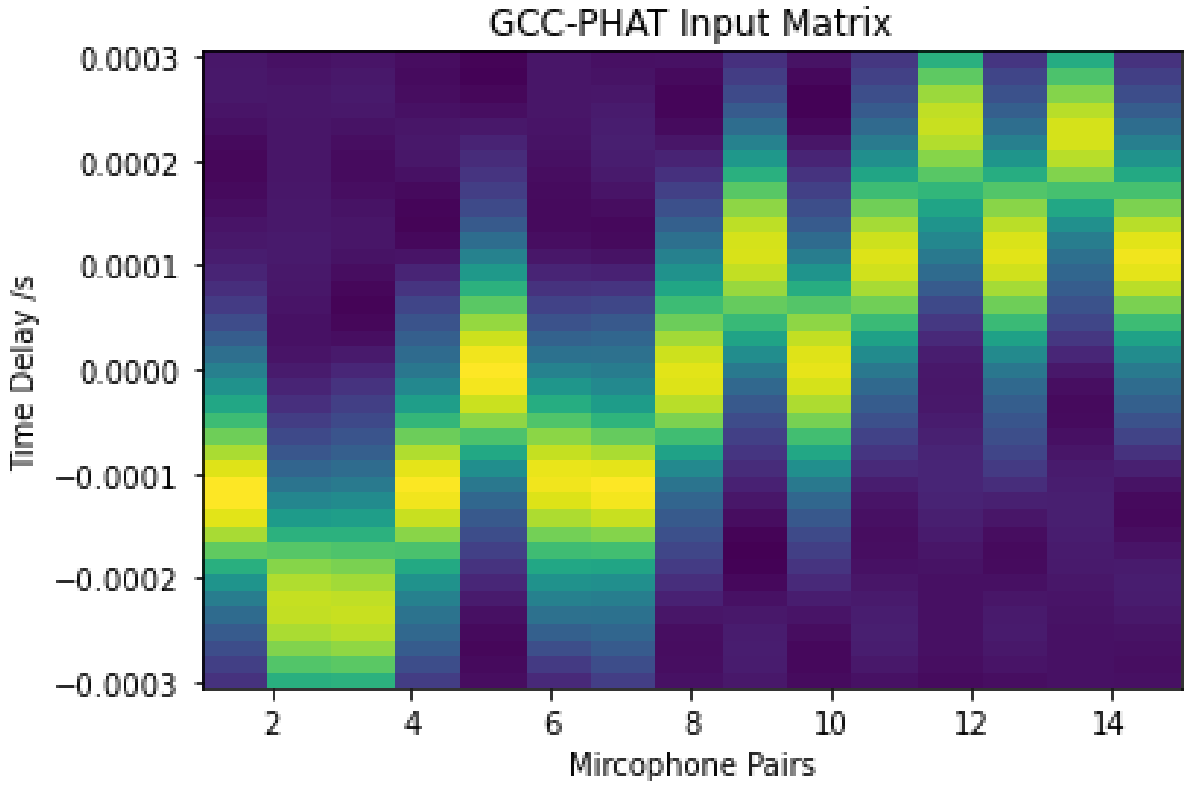


Figure 3.2: 39 x 15 GCC-PHAT inputs (yellow stripes indicate TDOA peaks for different microphone pairs)

3.2.3 Data Simulation

To generate audio data captured by a 6 microphone UCA similar to the Respeaker, a dataset of recorded utterances is needed as well as software that can estimate the rooms impulse response (RIR) for the 6 microphones. Then the utterance is convolved with the RIR to obtain the audio input of each microphone. Any dataset with recorded voices would work but in this experiment the AMI Corpus dataset [24] was used which is hosted by the University of Edinburgh and has audio recorded from meetings in office conditions. A Python code that does VAD to the WAV file of the audio recording was used to separate the audio in 2 second voiced parts. It is not important to have an entire utterance in the voiced segments as any audio of similar frequency content to the expected real input, that has at least the acceptable frame length for the VAD, is sufficient.

To find the RIR at each microphone the Image source method (**ISM**) was chosen. The ISM models the pressure waves emitted by the sound source in a rectangular room and creates an image of that wave

with decayed energy after each wall reflection. A transfer function of the summation of all the pressure waves at any space coordinate in that rectangular room can be found and its FFT is the time domain RIR [25]. The ISM can estimate an accurate RIR over the audio band frequencies for rectangular rooms where the source and the microphone are not close to the walls, which are acceptable assumptions for the initial testing of this project. To simulate the RIR of figure 3.3 the pyroomacoustics library was used [26] that allows for tuning of the rooms to polyhedral shapes, inputting the location of an arbitrary number of sources and microphones, using ray tracing to complement the ISM etc. Pyroomacoustics has a function that convolves the RIR with the audio of the sound source and adds additive white Gaussian noise (AWGN) to create the simulated microphone inputs with the selected SNR. This way 2350 data for every degree in the 60-degree implementation were generated and the parameters used in the room simulation are found in Appendix A

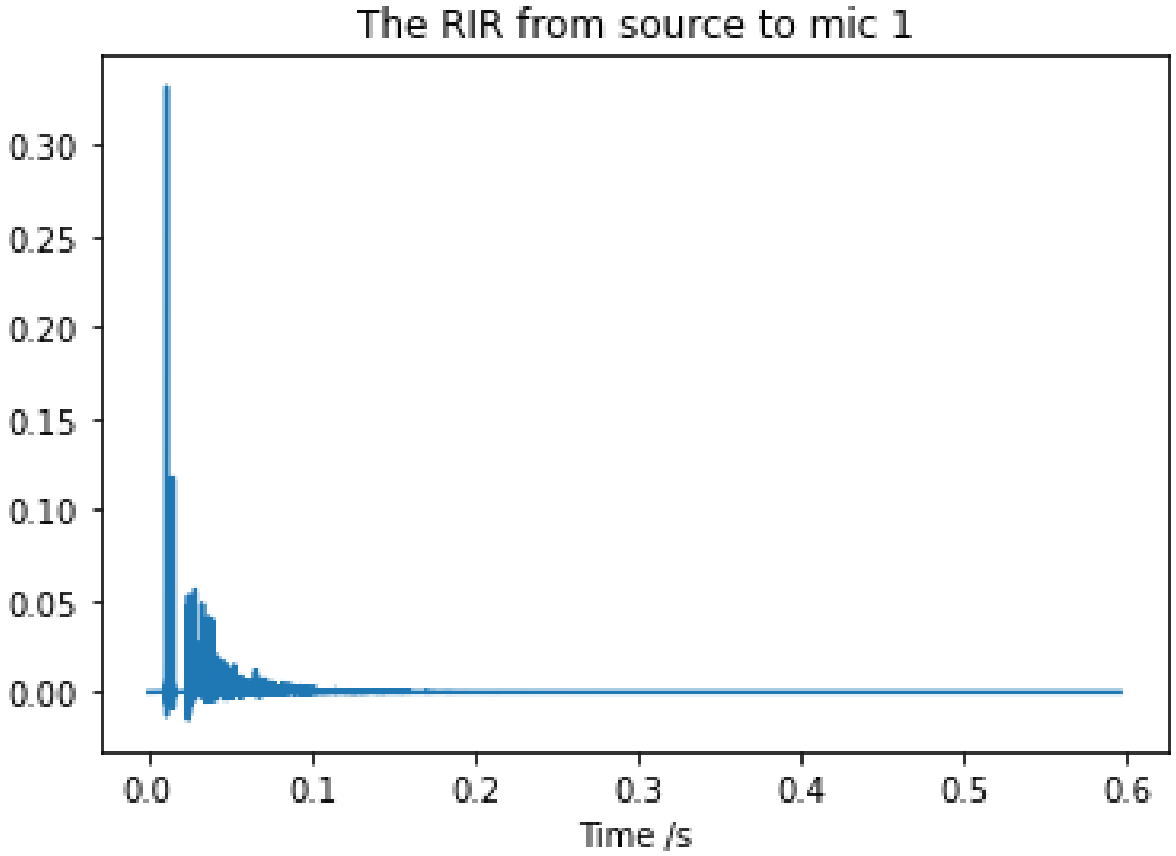


Figure 3.3: Room Impulse Response for a simulated room

3.2.4 Recorded Data

Some real audio data were recorded using the Respeaker for fine tuning the NN model and for evaluating its performance on real data. Ideally this should done in an audio lab, that has soundproof foam on the walls to reduce reverberation, by placing the microphone array on a rotating platform that can be electronically pointed at exactly the desired DOA relative to the source. Since access to an audio lab was not possible, the following experiment was conducted in a 4x6x3 (width, length, height) room with hard wall surfaces. The microphone array was placed on top of a 3D printed protractor that had a string attached to its center, so that the center of the protractor and the UCA matched. The other end of the string was tied to a mobile phone that played recorded audio of people talking. Both the protractor and the phone were placed on top of two speaker stands close to the center of the room and by rotating the stand that had the microphone array on the desired DOA, indicated by the string, was selected. Figure 3.4 shows this setup. 50 samples were recorded for every 10 degrees to get a 10-degree resolution for the 60-degree algorithm implementation.

3.3 Neural Network

3.3.1 Model Architecture

In this work, the NN must be able to estimate the DOA within 60 degrees from the 39x15 GCC-PHAT inputs. The model should be trained in a supervised learning manner where a set of the GCC-PHAT inputs are assigned to output labels that represent the degrees. Supervised learning is chosen as we want to map the inputs to the DOA outputs and not just cluster them. The model could perform a regression task that outputs a value of the DOA. This theoretically has infinite resolution because the output will be a decimal value between 0 and 60 but the accuracy of the degrees that were not in the training dataset would be questionable and the regression is not expected to perform as well as classification in unseen inputs. Otherwise, the NN could do a classification task where each class represents a different direction and the resolution of the algorithm is determined by the number of classes. In this project, the number of classes was chosen to be 60 and 6 for a 60-degree prediction of resolution 1 and 10 degrees respectively. For the output labels, three different variations are one that uses one-hot encoded labels with a 1 in the ground truth direction, one that has a 1 in the ground truth and its adjacent degrees to effectively reduce the resolution and one that uses a Gaussian function centered around the ground truth [6] which is usually found in evidential learning and can provide a confidence estimate for the models' prediction. The first method of standard one-hot encoding was used for both the 1- and 10-degree resolution, but it would be interesting to attempt training with another label encoding technique.
Both a MLP and a CNN were considered. The MLP, inspired by [6], shown in figure 3.5 has the parameters shown in table 3.1. By changing the number of HL it was observed that the model did not have enough capacity with two HL and that four HL did not improve the accuracy on the test data but added extra parameters. The number of nodes was tweaked and although they did not affect the accuracy or loss as much, the values shown in figure 3.5 were chosen. The activation functions of the HL were chosen to be rectified linear units (relu) which is the most common choice for deep learning. The output layer had

60 or 6 outputs depending on the resolution (1 or 10 degrees) and a SoftMax activation function that pushes the model to classify a single class. The categorical cross entropy loss function should be used because the logarithm of the loss undoes the exponential term of the SoftMax inverting its saturation due to extremely positive or negative inputs preventing the vanishing gradient problem [12]. Categorical cross entropy is preferred over sparse categorical cross entropy when a vector of one-hot encoded labels is used. The adaptive moments (Adam) optimizer was used because it converged faster than rmsprop, sgd with a learning rate schedule and Adagrad, with a small initial learning rate of 0.00005 which was found by experimenting with different values. Goodfellow [12] suggests that no optimization algorithms outperform the other options so the choice depends on the users confidence of using it. Batch Normalization was used after every HL to facilitate optimization and reduce overfitting.

The CNN is depicted in figure 3.6 and its parameters are found in table 3.1. After the input layer, two convolutional layers are used because each layer is good at extracting one feature and in this case the main features of interest are the position of the TDOA peaks and their intensity. Pooling layers make the output invariant to translations of the input features, so as in this case the DOA depends on the vertical translations of the yellow peaks, pooling layers should not be used and the dimensionality of the feature maps was decreased by using a stride of 2x2. The kernel size in both layers was 3x3 and the number of filters was 12 for the first convolutional layer and 24 for the second. Afterwards, two fully connected also known as dense layers were used to map those features to the DOA classes.

Parameter	**Value**
Loss	categorical cross entropy
Optimizer	Adam
Learning rate	0.00005
Metrics	Accuracy
MLP number of parameters	73656
CNN number of parameters	87238

Table 3.1: MLP and CNN parameters.

3.3.2 Training

For training the 2350 simulated data were pre-processed to remove any NaN values and then separated in approximately 1400 (60%) training data, 470 (20%) test data and 470 (20%) validation data. The data were either separated in groups of 1 or 10 degrees depending on the resolution. By testing for different batch sizes of powers of 2, the batch size was chosen to be 32, because for my data gave the perfect balance of fast convergence and good generalization error. Training with batch sizes close to one were affected by noise in the training dataset and were very slow to converge while batch sizes greater than 64 did not generalize well judging from the validation loss. The models were run for 100 epochs with an early stopping if the validation loss did not decrease for more than 10 epochs and the model with the best validation loss was saved. The trained models were evaluated on the test dataset where the MLP achieved an accuracy of 99.6% and the CNN achieved an accuracy of 98.9%. The same pre-processing procedure

was done for the real recorded data but only for the 10-degree resolution. The data was separated in 30 training data, 10 validation data and 10 test data for every 10 degrees interval. Then, the best 10-degree MLP and CNN models were loaded and evaluated on the test dataset of the real data, but achieved an accuracy of only 30% and 20%. Even though this accuracy estimate is questionable because of the small number of tests, it shows that the models trained with only simulated data have limited performance on real data. To improve the accuracy of the models trained with artificial data, fine tuning was done with the real data and the models were reevaluated to give an accuracy of 70%. This highlights the importance of having a sufficient number of data that are also representative of the data distribution that the model is going to encounter. The MLP model was converted to TensorFlow lite [27], because it achieved better accuracy than the CNN on the test data and had fewer parameters so will run faster, to be implemented in a Python script on the Respeaker embedded computer.

3.4 Beamforming Implementation

For the scope of the 60-degree implementation, the beamformer needs to find the direction of the microphone that is closest to the source, so a 60-degree resolution and robustness to reverberations are the prioritized attributes. The **PSB** approximation of the DSB was picked for this work because it has robustness to reverberation and gives a 60-degree resolution for frequencies greater than approximately 1kHz, which covers most of the audio band [8, 9]. As most speech signals have harmonics of frequencies higher than 1kHz it should correctly find the direction closest to the source but might suffer with lower frequency signals. Tests using the SDB to find the reference microphone would be an immediate extension of this work, because SDB achieves the highest directivity and noise suppression compared to most common beamforming algorithms and works best with UCAs [8] but was left as a scope for extension due to time limitations.

After recording a voiced audio signal, we beamform iteratively on the direction of all microphones and find which direction has the most energy, which corresponds to the reference microphone. A logical approach would be to iteratively beamform across all 360 directions and directly find the DOA in a similar manner, but the accuracy achieved is not satisfactory because of the low directivity of PSB.

A 6x6 lookup table is made in MATLAB (Code found in A.3) which holds the complex weights that each of the 6 microphones must be multiplied with for the 6 different beamforming directions. This table is imported to Python where the FFT for each channel of a recorded audio signal is multiplied by the corresponding beamforming weight and the channels are summed together to give the FFT of the beamformed signal. Then according to Parseval's theorem, the energy of the signal can be found from the FFT of the beamformed signal $Y(f)$ by equation 3.3

$$Energy = \sum_{0}^{N-1} |y(n)|^2 = \frac{1}{N} \sum_{0}^{N-1} |Y(f)|^2 \tag{3.3}$$

and by iteratively doing this for all 6 directions the direction with the maximum energy is found. Then the index of the reference microphone is used to reorder the channels to effectively rotate the array

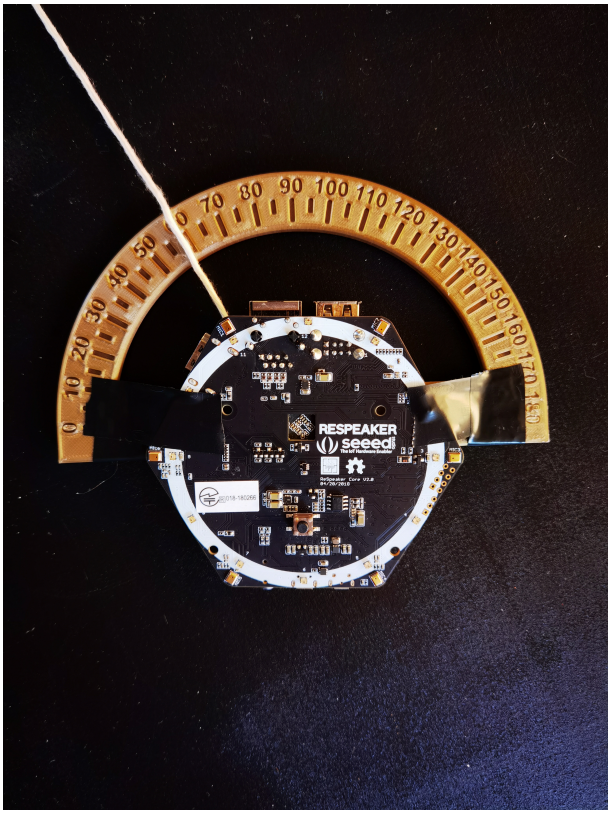
for the NN to predict the DOA for the 60 degrees corresponding to the reference microphone. Finally, the information of the NN and the reference microphone are combined to estimate the DOA for all 360 degrees.

3.5 Multiple Sources Insight

Different methods to address the multi-source scenario for the 60-degree implementation were attempted. Firstly, the activation function of the NN was changed to sigmoid and the model would predict the likelihood that a source exist in every direction for each microphone acting as the reference microphone of the 60 degree implementation. This gave many "accurate" random classifications as at least one source was identified in every 60 degree region while only one sound source was actually active. Erroneous classifications could be avoided by many more training data and some form of evidential learning to have a confidence measure of the models prediction. However, this technique is unlikely to work well with the 60- degree implementation as reordering the input channels to match the reference microphone for every microphone is prone to give false positives in some 60-degree regions.
The next method that requires a source counting technique to iteratively find the DOA in section 1.6 is investigated. The source counting technique that was attempted beamforms in 360 degrees and checks how many peaks there are in the power map but it did not have the required resolution to identify two sources that are closer than approximately 100 degrees to each other rendering it unreliable to use for more than 2 sources. In addition, the code that nulls the effect of the first DOA from the GCC-PHAT function works only for the microphone combinations that produce a second peak, but some combinations where the TDOA peaks of the two sources overlap are also nulled. This raises uncertainty on whether a NN trained for the single source scenario could handle the case where some of the TDOA peaks are missing. This is illustrated in figure 3.7, where 3.7a shows the GCC-PHAT matrix for two sources. The GCC-PHAT for microphone pair 8 is shown in 3.7b where when the time delay values of the dominant source are zeroed, the peak of the second source becomes dominant. However, for microphone pair 6 shown in 3.7c there is a single peak as the TDOA of the two sources coincide which when nulled destroys the DOA information.

The other method that required training for the 64620 different source combinations to localize two sources was not attempted as it was not practical with the available equipment.

(a) Microphone array and sound source mounted on top of speaker stands



(b) Microphone array on 3D printed protractor with string to indicate the direction

Figure 3.4: The experimental setup used to record the data and test the algorithm

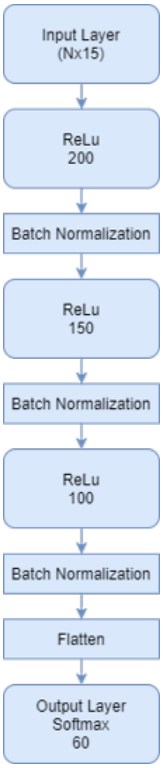


Figure 3.5: MLP model architecture graph

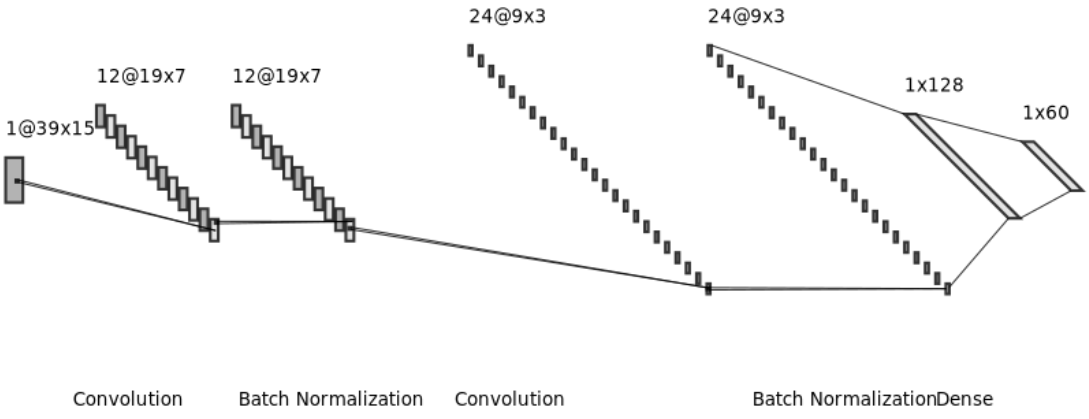
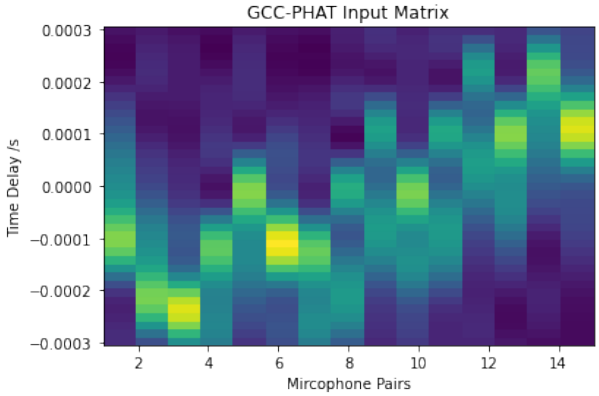
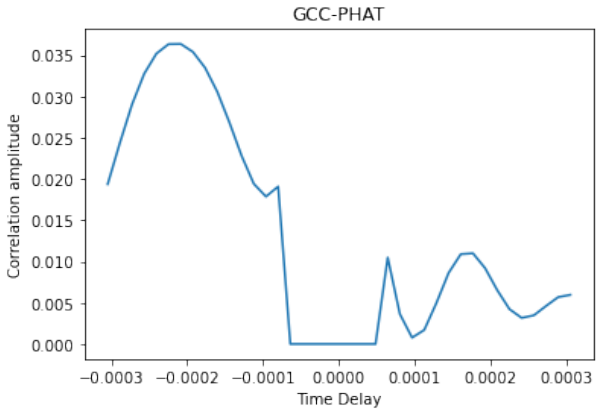


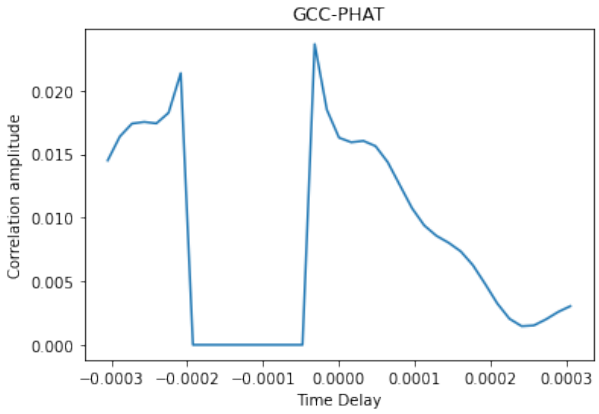
Figure 3.6: CNN model architecture graph



(a) GCC-PHAT input matrix for two sources



(b) GCC-PHAT for microphone pair 8



(c) GCC-PHAT for microphone pair 6

Figure 3.7: Illustration of nulling the GCC-PHAT time delay values of the dominant source

Chapter 4

Results

In order to check the performance of the DOA estimation algorithm, Python scripts were implemented in the Respeaker ARM embedded computer and the setup of figure 3.4 was recycled for testing. A series of experiments were conducted where the 60-degree implementation was tested for different resolutions and conditions. More specifically, fixed distance tests were done with the 1 degree resolution model that was not fine tuned and the 10 degree resolution fine tuned model with and without VAD as a prior step. A variable distance experiment to determine the effect of distance and SNR to the accuracy of the algorithm as well as a test where the Respeaker was placed in a corner of the room to determine the performance in reverberant conditions were done. The MUSIC algorithm of the pyargus Python API [28] was used to provide a performance reference for some of the tests. The average time it took to run the whole DOA algorithm was 0.71 seconds which can almost run in real time and the average time of running the NN model alone was 0.009 seconds which is fast enough for real time applications.

4.1 Non fine-tuned NN with 1 degree resolution

This test was done with a fixed distance between the microphone array and the mobile phone sound source of 1.7 m and a height from the ground of 1.2 m, where audio segments of 0.03s were recorded whenever the sound source was emitting sound. This NN is a MLP and was not trained with real data because the method used for collecting real data is not accurate to 1 degree and because of the time limitation of collecting enough data to train 60 different classes. This experiment aimed to determine how well the NN trained only on simulation data performed in real applications. The results for 450 predictions are shown in table 4.1.

It is seen that the accuracy of the beamforming algorithm that finds the reference microphone was correct 65% of time. The model predicted the correct degree only 0.9% of the times and performed better within 30 degrees with a 25% accuracy. The accuracy of the NN with 1 degree resolution when the beamforming predicted the correct source direction was 1.4%.

Process	Accuracy
Beamforming	65%
DOA in 1 degree	0.9%
DOA in 30 degrees	25%
NN prediction in 1 degree with correct reference microphone	1.4%

Table 4.1: Results for non fine tuned model.

4.2 Fine-tuned NN with 10 degrees resolution

This experiment had a fixed source to UCA distance of 1.7m and height of 1.2m, where 0.03s audio segments were recorded whenever sound was emitted. The fine tuned MLP model with real data was used with a 10 degree resolution. This experiment aims to test whether fine tuning the NN with real data will improve its performance in practise. The MUSIC algorithm was used as a reference and 900 DOA estimations were done for 6 different directions as seen in table 4.2..

Process	Accuracy
Beamforming	67%
DOA in 10 degree	27%
DOA in 30 degrees	51%
NN prediction in 10 degree with correct reference microphone	41%
MUSIC in 10 degrees	13%
MUSIC in 30 degrees	27%

Table 4.2: Results for fine tuned model.

The beamforming accuracy has not changed much as expected and is 67%. This time the DOA estimation in 10 degrees is at 27% accuracy and at 51% in 30 degrees, more than double that of the non fine tuned model. The NN predicted the correct direction 41% of time when the reference microphone was correctly identified. For comparison the MUSIC algorithm had a 13% accuracy in 10 degrees and a 27% accuracy in 30 degrees.
A bar plot containing the accuracy of the DOA estimation algorithm in 10 degrees for each individual direction tested is shown in figure 4.1.

4.3 Fine-tuned 10-degree NN with VAD preprocessing

This experiment was done in similar conditions to the previous section but the recorded frames are passed through VAD pre-processing to have only voiced data as inputs. The results shown in table 4.3. are drawn from 600 predictions for 4 different directions

The beamforming accuracy is 71% which is close to the previous experiments. The DOA estimation accuracy in 10 degrees this time is 50% and 63% in 30 degrees. The NN prediction for a correct reference

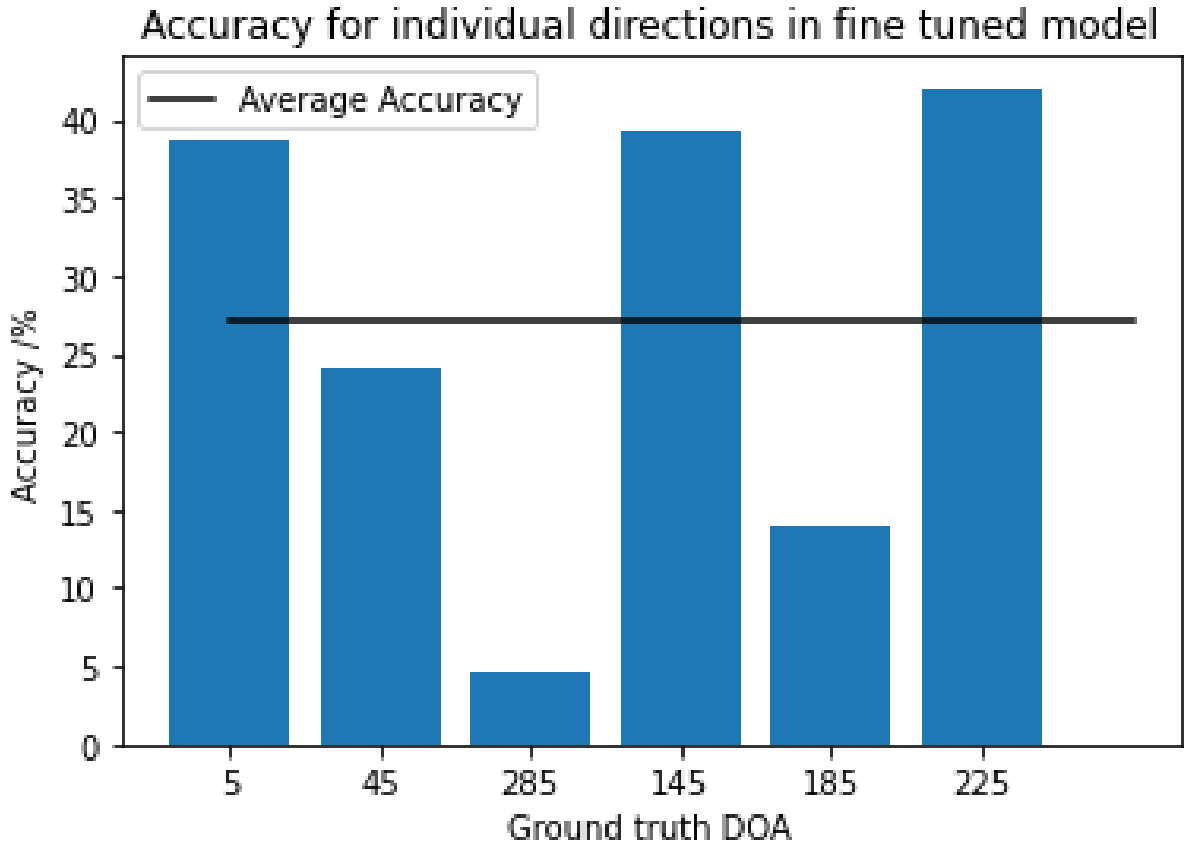


Figure 4.1: Bar plot of the DOA estimation algorithm in 10 degrees for each direction

Process	**Accuracy**
Beamforming	71%
DOA in 10 degree	50%
DOA in 30 degrees	63%
NN prediction in 10 degree with correct reference microphone	70%
MUSIC in 10 degrees	11%
MUSIC in 30 degrees	21%

Table 4.3: Results for fine tuned model with VAD.

microphone in 10 degrees is 70%.

A bar plot with the accuracy in 10 degrees of the DOA estimation algorithm for each individual direction is shown in figure 4.2

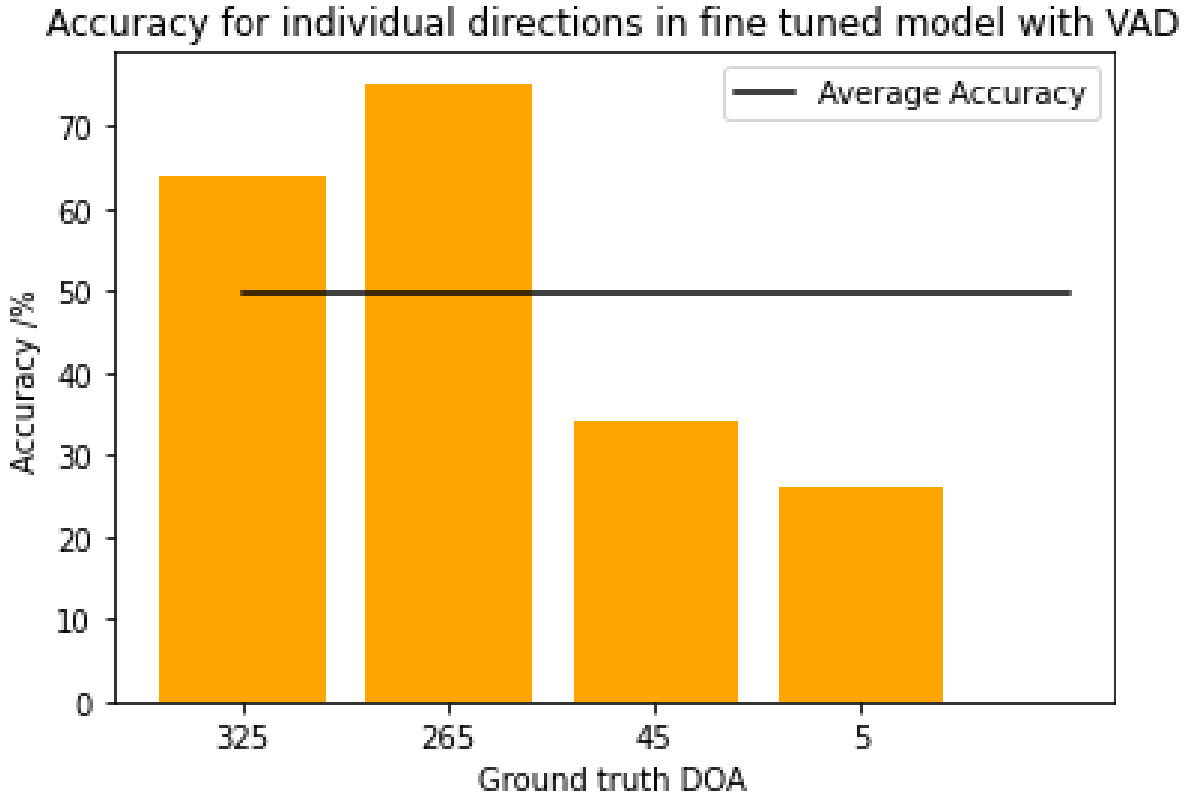


Figure 4.2: Bar plot of accuracy in 10 degrees of the DOA

4.4 Fine-tuned 10-degree NN with VAD for variable distance

The same setup as before was used but this time the distance to the microphone UCA was varied at 0.7m, 1.1m, 1.4m, 1.7m, 2.2m from source and the DOA was held constant at 45 degrees. The results for 750 predictions are shown in table 4.4.

Distance	0.7m	1.1m	1.4m	1.7m	2.2m
Beamforming accuracy	85%	57%	57%	55%	35%
DOA accuracy for 45 degrees	9%	0%	37%	23%	15%
DOA accuracy for 55 degrees	67%	49%	3%	24%	1%
DOA accuracy for 35-55 degrees	76%	49%	52%	50%	27%
Ratio of Beamforming to DOA accuracy in 30 degrees	89%	86%	92%	90%	78%

Table 4.4: Results for fine tuned model with VAD vs distance.

It can be seen that for the distances of 0.7m and 1.1 m the NN predicts the 55 degrees 67% and 49% of the times respectively. For the distance of 1.4m, 55 degrees are predicted only 3% and 45 degrees 52% of times, while for a distance of 1.7m they are both predicted approximately the same amount of time at

24%. For the distance of 2.2m the 45 degree is predicted 27% of times. This differences is likely caused by experimental error where the sound source was placed closer to 55 degrees for distances 0.7m and 1.1m, closer to 45 for distances 1.4m and 2.2m and approximately in the middle (50 degrees) for the distance of 1.7m. Overall if we consider a 30 degree resolution, to reduce experimental error, it is observed that the DOA estimation accuracy drops with increasing distance. Beamforming seems to perform better at closer distances with 85% accuracy and drops to 35%. The ratio of the beamforming accuracy to the NN accuracy in 30 degrees remains relatively constant with distance at an average of 89.25% for distances of 0.7-1.7m and drops to 78% in 2.2m.

Figure 4.3 shows the accuracy drop of the beamforming algorithm and the DOA for 45 and 55 degrees with distance.

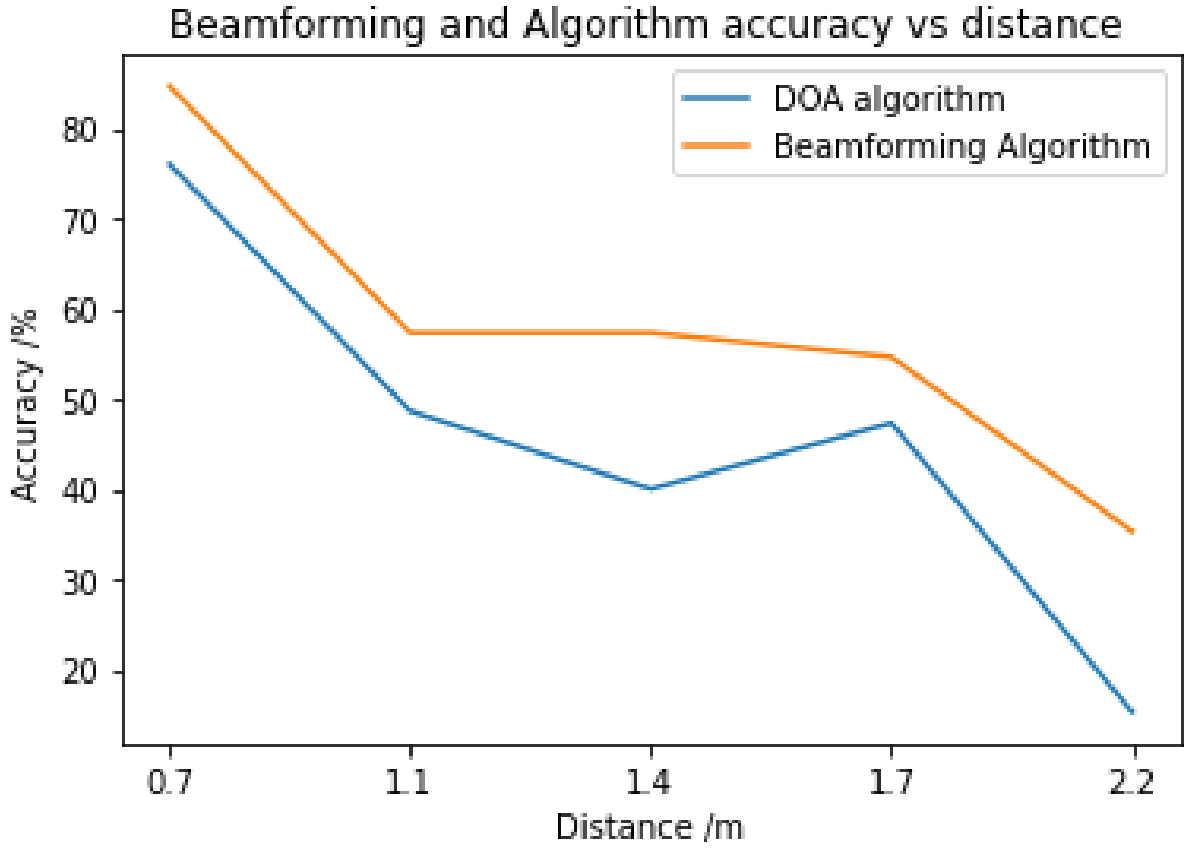


Figure 4.3: Accuracy of the DOA in 20 degrees and beamforming algorithms vs distance

4.5 Fine-tuned 10-degree NN with VAD in reverberant conditions

The microphone UCA was placed approximately 20 cm from a corner of the room at a distance of 1.7m from the source and predicted the DOA 450 times for three different directions as seen in table 4.5.

Process	**Accuracy**
Beamforming	45%
DOA in 10 degree	30%
DOA in 30 degrees	43%
NN prediction in 10 degree with correct reference microphone	66%

Table 4.5: Results for fine tuned model in reverberant environment.

In the reverberant scenario the accuracy of the beamformer drops to 45% which shows that the beamforming technique is not robust in noisy conditions. The model however achieves an accuracy of 30% and 43% for the 10 and 30 degree resolutions respectively and when the reference microphone is predicted correctly achieves an accuracy of 66%. This shows that the model is not affected as much from the reverberations of the environment.

Chapter 5

Discussion

5.1 Results analysis

First of all, we consider the experiment with the NN trained with only simulated data with a 1 degree resolution where table 4.1 shows that the model accuracy is at 0.9% which is 3 times better than random $\frac{1}{360}100 = 0.28\%$. The probability of a correct classification when the correct reference microphone is found is $\frac{1}{60}100 = 1.6\%$ and the NN accuracy with a correct reference microphone is at 1.4% which is slightly worse than random. This indicates that the simulated data are not representative of the real data and therefore the NN can not generalize to the real data and make a correct prediction. The experiment of section 4.2 uses the 10-degree resolution NN that was fine tuned with 30 real training data for each 10 degree interval. The accuracy of the algorithm in 10 degrees is 27% and 41% in 30 degrees resolution as seen in table 4.2. Compared to the model of table 4.1 which was 25% there has been a $\frac{51-25}{25}100 = 104\%$ increase in the accuracy of the overall algorithm. Even though the accuracy is low, the **MUSIC** algorithm which is known to have a good performance from the conventional DOA algorithms [2], for the exact same audio inputs had an accuracy of 13% which is $\frac{27-13}{27}100 = 51.8\%$ lower than the proposed algorithm. This is likely due to reverberation in the room that gives the characteristics of the signal to the noise. That way the noise subspace and the signal subspace that are used in MUSIC are not orthogonal anymore. Figure 4.1 shows that the accuracy percentage is not equal for all degrees, but some directions such as 225 are predicted correctly more frequently while directions such as 285 is found correctly only 5% of the time. This difference could result from three possible sources of error. One source could be the experimental error of the setup of page 22, as the resolution of the protractor is five degrees which is enough for the true DOA to be misplaced in the adjacent direction, and the accuracy of the string method depends on its tightness. Another source of error that is linked with the setup error mentioned above is the experimental error when recording the training data that might have given some of the NN classes a bias. For instance, if the true DOA was at 15 degrees when recording for the 5 degree direction then there would be a bias as the 5 degree direction is not properly represented in the training data and would be falsely predicted when the DOA is at 15 degrees. In addition, beamforming can produce an error of theoretically 10 degrees, especially when the DOA is in between two sources. As every source covers 60-degree regions

and beamforming has approximately 60 degrees resolution an error is more likely to occur at directions 30,90,150,210,270 and 330 which are the boundaries in between the microphones rather than at the center of the beamforming direction which is at 0,60,120,180,240 and 300 degrees. In the case of 285 degrees this should not be the main source of error as it is 15 degrees away from the boundary while degree 145 is only 5 degrees away but still has much better accuracy. The experiment in section 4.3 uses the same NN but the audio segments are first passed in the VAD algorithm and only those classified as voiced go into the DOA algorithm. The statistics of table 4.3 show that this time the DOA prediction in 10 degrees is 50% accurate and in 30 degrees it is 63% accurate while the performance of the MUSIC algorithm has slightly dropped. Using VAD the audio segments that did not include voiced data, such as those that could occur in the 30 milliseconds in between utterances, are now filtered. However, the increase in performance is not likely to be caused by the voiced segments otherwise the MUSIC algorithm should have also had an increase in performance. By examining figure 4.2 it is noticed that degrees 325 and 265 have a very good accuracy of approximately 65% and 75% while degrees 45 and 5 have an accuracy of approximately 35% and 25%. Compared to the previous experiment the accuracy for degree 5 dropped almost 15% while the accuracy of degree 45 increased by approximately 10%. This must be the effect of the experimental error of the setup. Degree 325 and 265 have in common that they are the upper 10 degree cluster of the 60 degree region which indicates that there might be a bias that predicts the upper 10 degrees more frequently. Degree 145 from figure 4.1 which also is in the higher 10 degree cluster of the 60 degrees, has achieved accuracy higher than average. The increase in performance is likely to be because overall this experiment had less setup experimental error and because the classes that the randomly chosen directions represented in the 60 degrees were favoured by the NN. Section 4.4 shows the experiment of the DOA algorithm versus distance. The main difference of increasing the distance is that the received SNR drops, but longer distances might also have more experimental error. The degree was held constant at 45 degrees but table 4.4 shows that for 0.7m and 1.1m distance of microphone array to source, degree 55 was predicted much more often than 45 degrees, then at distances 1.4m the 45 degree direction was predicted more often, at 1.7m they were both predicted an approximately equal amount of times and at 2.2m the 45 degree is again dominant, being predicted 15% of times with the 35 degree direction also being predicted 11% of times. This must be the result of the experimental error of the setup where the actual direction was closer to 55 degrees for distances of 0.7m and 1.1m, closer to 45 degrees for distances 1.4m and 2.2m and in the middle for the distance of 1.7m. To try to investigate the relationship of distance with the accuracy of the algorithm, a prediction is considered accurate for either a prediction of 45 or 55 degrees. Figure 4.3 shows that there is a clear decrease of the performance of the algorithm as the distance increases and hence the SNR drops. This trend mostly follows the decrease in performance of the beamforming algorithm which determines the maximum accuracy of the whole estimation. Beamforming predicts the correct reference microphone 85% of time when the source is at 0.7m and drops to 35% accuracy at 2.2m. This suggests that the beamforming algorithm is very dependent on SNR. The accuracy ratio of the beamforming algorithm to the NN prediction in 30 degrees remains almost constant with distance at an average of 89.25% from 0.7m-1.7m and drops to 78% at 2.2m. This suggests that the NN predictions are robust to the change in distance. Finally, section 4.5 has the results for the algorithms performance in the corner of the room where the reverberations are high. From

table 4.5 it is observed that the accuracy of the beamforming algorithm significantly drops to 45% in reverberant conditions where the SNR is lowered due to increased noise. Nonetheless, the predictions of the NN for the correct reference microphone are accurate 66% of times. This shows that the deep NN can perform well despite the reverberations. The results indicate that the beamforming method only performs well in high SNR and is not reliable to use in reverberant environments or in long distances. Moreover, it is observed that the setup with the protractor and the string has a too large experimental error and must not be used for accurate measurements or for collecting training data, as the error introduces bias to the resulting classes. Also, the simulated data are not representative of the real microphone input data in my room as it gave worse than random performance. Nonetheless, the NN trained with even a minimal amount of training data performs robustly in low SNR environments as its performance is not affected much by increasing reverberation or distance.

5.2 Scope for extension

An immediate extension would be to try SDB as the beamforming algorithm that has a better resolution in the low audio frequency [8]. Also, gathering a substantial amount of real training data, at least 5000 [12], is necessary to improve the performance of the NN. This data must be recorded in an audio lab where the microphone UCA is to be placed on a rotating platform that can accurately control the DOA of the sound source and the same setup should be used for testing the algorithms performance to reduce the experimental error. In the case that more data are gathered in varying conditions and the capacity of the model needs to be increased, the CNN introduced in section 3.3.1 could potentially be a more suitable NN.
Even though the 60-degree implementation relaxes the training data requirement and is suitable for localizing one source, it makes the multiple sources approach more difficult, as depending on the position of the additional sources the symmetry that would be observed if only one source was present is lost. So for future work, a NN would be trained for 360 degrees estimation and the methods presented in section 1.6 would be investigated. Testing the performance difference with a 3D printed cylindrical baffle [8, 9] would be interesting but was left outside the scope of this project. Finally, the algorithm could have extra functionality depending on the task that the DOA is used for. For example, if it is used for speech recognition or speech selection, the model could learn to output the beamforming weights for the predicted direction and be optimized for the speech recognition task [15].

5.3 Conclusion

This study investigated the different techniques that are traditionally used for DOA estimation and an AI algorithm was implemented in an embedded microphone array computer. Training data were simulated and recorded and a MLP and a CNN were trained and evaluated achieving similar performance on the test dataset. The proposed algorithm exploits the symmetry of the UCA by beamforming and the NN was able to predict the correct DOA in 10 degrees with reasonable accuracy while being trained

with a very small dataset. Experiments that tested the performance of this algorithm showed that the beamforming algorithm suffered in noisy conditions that sets an upper limit to the algorithms' accuracy. The accuracy of the NN was not affected by low SNR or reverberant environments so it can be a robust approach for DOA estimation. This 60-degree implementation for UCAs was functional for one source and could potentially be improved by augmenting the training dataset and trying a different beamforming algorithm. Even though the multiple source scenario is a limitation of this approach, insight on a possible multi-source implementation is given.

Acknowledgements

I want to thank my family and friends for bringing me to where I am, but most of all I want to thank my father for making all of this possible.

References

[1] Bronkhorst, A.: 'The Cocktail Party Phenomenon: A Review of Research on Speech Intelligibility in Multiple-Talker Conditions', in: *Acta Acustica united with Acustica* 86 (Jan. 2000), pp. 117–128.

[2] Chung, P.-J., Viberg, M. and Yu, J.: 'DOA Estimation Methods and Algorithms', in: vol. 3, pp. 599–650, ISBN: 9780124115972, DOI: 10.1016/B978-0-12-411597-2.00014-X.

[3] Xiao, X., Zhao, S., Zhong, X., Jones, D. L., Chng, E. S. and Li, H.: 'A learning-based approach to direction of arrival estimation in noisy and reverberant environments', in: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2814–2818, DOI: 10.1109/ICASSP.2015.7178484.

[4] Ferguson, E. L., Williams, S. B. and Jin, C. T.: 'Sound Source Localization in a Multipath Environment Using Convolutional Neural Networks', in: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2386–2390, DOI: 10.1109/ICASSP.2018.8462024.

[5] Qinglong, L., Zhang, X. and Li, H.: 'Online Direction of Arrival Estimation Based on Deep Learning', in: DOI: 10.1109/ICASSP.2018.8461386.

[6] He, W., Motlicek, P. and Odobez, J.-M.: 'Deep Neural Networks for Multiple Speaker Detection and Localization', in: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 74–79, DOI: 10.1109/ICRA.2018.8461267.

[7] Pavlidi, D., Griffin, A., Puigt, M. and Mouchtaris, A.: 'Real-Time Multiple Sound Source Localization and Counting Using a Circular Microphone Array', in: *IEEE Transactions on Audio, Speech, and Language Processing* 21.10 (2013), pp. 2193–2206, DOI: 10.1109/TASL.2013.2272524.

[8] Blanco Galindo, M., Coleman, P. and Jackson, P.: 'Microphone Array Geometries for Horizontal Spatial Audio Object Capture With Beamforming', in: *Journal of the Audio Engineering Society* 68 (June 2020), pp. 324–337, DOI: 10.17743/jaes.2020.0025.

[9] Tiana-Roig, E., Jacobsen, F. and Fernandez-Grande, E.: 'Beamforming with a circular microphone array for localization of environmental noise sources', in: *The Journal of the Acoustical Society of America* 128 (Dec. 2010), pp. 3535–42, DOI: 10.1121/1.3500669.

[10] Kwon, B., Park, Y. and Park, Y.-s.: 'Analysis of the GCC-PHAT technique for multiple sources', in: *ICCAS 2010*, pp. 2070–2073, DOI: 10.1109/ICCAS.2010.5670137.

[11] Grondin, F. and Glass, J.: *Multiple Sound Source Localization with SVD-PHAT*, June 2019.

[12] Goodfellow, I., Bengio, Y. and Courville, A.: 'Deep Learning', http://www.deeplearningbook.org, (MIT Press, 2016).

[13] LeCun, Y., Bengio, Y. and Hinton, G.: 'Deep Learning', in: *Nature* 521 (May 2015), pp. 436–44, DOI: 10.1038/nature14539.

[14] Adavanne, S., Politis, A. and Virtanen, T.: 'Direction of Arrival Estimation for Multiple Sound Sources Using Convolutional Recurrent Neural Network', in: *2018 26th European Signal Processing Conference (EUSIPCO)*, pp. 1462–1466, DOI: 10.23919/EUSIPCO.2018.8553182.

[15] Xiao, X., Watanabe, S., Erdogan, H. et al.: 'Deep beamforming networks for multi-channel speech recognition', in: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5745–5749, DOI: 10.1109/ICASSP.2016.7472778.

[16] Rascon, C., Meza, I., Fuentes-Pineda, G., Salinas, L. and Pineda, L.: 'Integration of the Multi-DOA Estimation Functionality to Human-Robot Interaction', in: *International Journal of Advanced Robotic Systems* 12 (Feb. 2015), DOI: 10.5772/59993.

[17] Elfitri, I. and Luthfi, A.: 'Reviews on Technology and Standard of Spatial Audio Coding', in: *JURNAL NASIONAL TEKNIK ELEKTRO* 6 (Mar. 2017), DOI: 10.20449/jnte.v6i1.372.

[18] Lionel Sujay Vailshery: 'Smart speaker market revenue worldwide in 2017, 2018 and 2022', Statista, 2021, https://www.statista.com/statistics/822511/worldwide-smart-speaker-market-revenue/, accessed Apr. 2021.

[19] 'GAMING MARKET - GROWTH, TRENDS, COVID-19 IMPACT, AND FORECASTS (2021 - 2026)', Mordor Intelligence, 2021, https://www.mordorintelligence.com/industry-reports/global-games-market, accessed Apr. 2021.

[20] '2020 Year-End Music Industry Revenue Report | RIAA', RIAA, 2021, https://www.riaa.com/reports/2020-year-end-music-industry-revenue-report/, accessed Apr. 2021.

[21] Julia Stoll: 'Global box office revenue from 2005 to 2020', Statista, 2021, https://www.statista.com/statistics/271856/global-box-office-revenue/, accessed Apr. 2021.

[22] John Wiseman: *WebRTCvad source code*, 2016, URL: https://github.com/wiseman/py-webrtcvad/blob/master/cbits/webrtc/common_audio/vad/vad_core.c, (visited on Apr. 2021), Github.

[23] 'ReSpeaker Core v2.0', Seeed, 2021, https://wiki.seeedstudio.com/ReSpeaker_Core_v2.0/, accessed Apr. 2021.

[24] *AMI Corpus*, License: CC BY 4.0, University of Edinburgh, URL: license : https://creativecommons.org/licenses/by/4.0/legalcode, (visited on Apr. 2021).

[25] Allen, J. and Berkley, D.: 'Image method for efficiently simulating small-room acoustics', in: *The Journal of the Acoustical Society of America* 65 (Apr. 1979), pp. 943–950, DOI: 10.1121/1.382599.

[26] Scheibler, R., Bezzam, E. and Dokmanić, I.: 'Pyroomacoustics: A Python Package for Audio Room Simulation and Array Processing Algorithms', in: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 351–355, DOI: 10.1109/ICASSP.2018.8461310.

[27] TensorFlow: *Deploy machine learning models on mobile and IoT devices*, TensorFlow, 2021, URL: https://www.tensorflow.org/lite, (visited on Apr. 2021).

[28] Tamás Pető: *pyArgus*, 2021, URL: https://pypi.org/project/pyargus/, (visited on Apr. 2021).

Appendix A

Supporting Information

A.1 Multi source rate of growth

Figure A.1 shows how the factorial growth compares to an exponential growth of $y = 360^x$ for up to 5 sources.

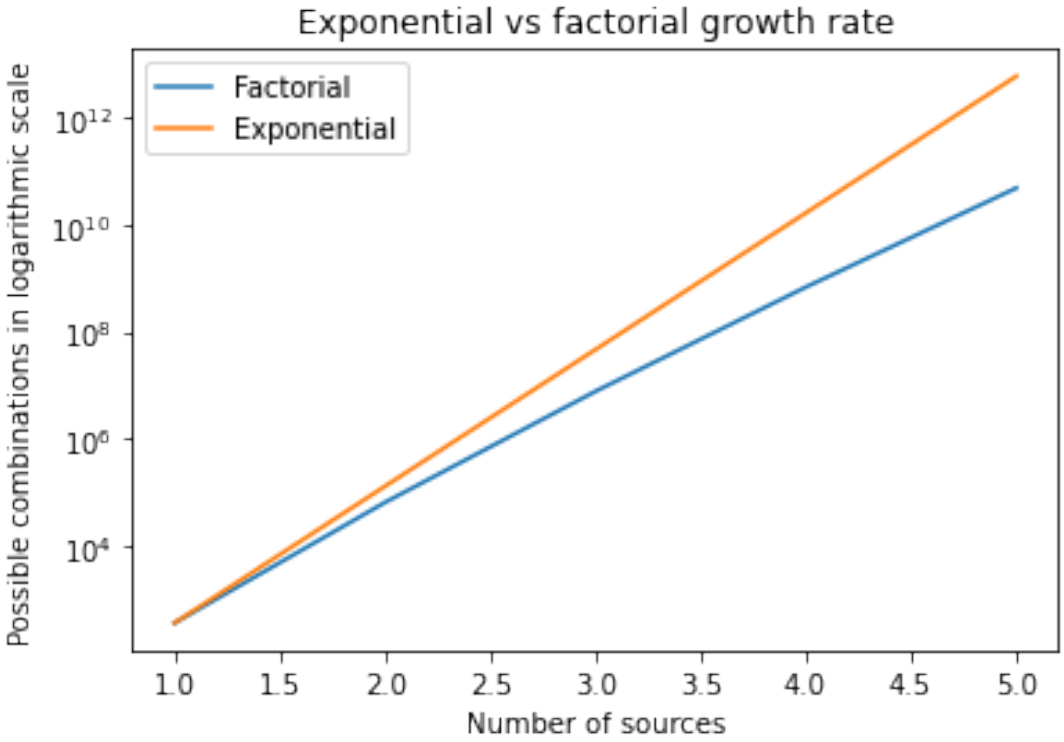


Figure A.1: Exponential vs factorial rate of growth

A.2 Simulation Parameters

These are the parameters that were used in the pyroomacoustics library to simulate audio input.

Parameter	Value
Room dimensions	4x6x3
Source height	1.2m
Microphone array height	1.2m
Microphone array center location	(2,2)
Source - UCA distance	1.5m and 3m
RT60	0.57s
SNR	10dB
Sampling frequency	16000 Hz

Table A.1: Simulation parameters.

A.3 MATLAB Beamforming Weights

This is the code used to generate the beamforming weights in MATLAB

```
1
2 mic = phased.CustomMicrophoneElement;
3 mic.FrequencyVector = [0 20*10^3];
4 micarray = phased.UCA('NumElements',6,'Radius',0.0463,'Element',mic);
5 pos = getElementPosition(micarray);
6 pos(3,:) = [];
7 ang = [30,90,150,-150,-80,-20];
8 wt = cbfweights(pos,ang);
9 save("beamformerwt1.mat",'wt')
```

Appendix B

Python code

```
1
2  import numpy as np
3  import webrtcvad
4  import sys
5  import pyaudio
6  import scipy.io
7  import tflite_runtime.interpreter as tflite
8  import math
9  import itertools
10
11
12
13
14 #Data simulation code
15
16 def npow2(x):
17     #finds the next power of 2 of the input x
18     return 1<<(x-1).bit_length()
19
20
21 def stftprep(x, fs, time):
22     #splits the input array in an array of frames of fixed length (to be used to prepare
       array for VAD or STFT)
23
24     #x = input array
25     #fs = sampling frequency
26     #time = time of the split frames (determines the frame length)
27
28     # returns the array of the split input data
29     framesamp = int(time*fs)
30     X = np.array([x[i:i+framesamp] for i in range(0, len(x)-framesamp, framesamp)])
31     return X
32
```

```
33
34 def VAD(aggr, data ,sr ,time):
35     #aggr how sensitive voice detection is
36     #data (numpy array) to be classified as voiced/unvoiced
37     #sr is the sampling rate
38     # the time of the frame in ms (10,20,30 ms)
39
40     #Sets the vad object
41     vad = webrtcvad.Vad(aggr)
42     #sets frame length
43     framesz= time/1000
44     #cuts the data in the frame size
45     Z = stftprep(data, sr, time/1000)
46     #returns indexes of voiced data
47     ind = [i for i in range(len(Z)) if vad.is_speech(np.int16(Z[i]*32768).tobytes(),sr)]
       # multiply by 32768
48     #creates the array of the voiced parts of the data
49     emin = [Z[k] for k in ind]
50     eminor = np.array([j for i in emin for j in i])
51     return eminor
52
53
54 def roomsimdata(rt60,dim ,fs,data,sloc,miccent,mich,deg,delay =1,snr = None , N =1):
55
56     #Function that simulates a room and returns the signals captured by the microphones
57
58     #rt60 is the time it takes for reverberations to fall below 60dB
59     #dim is a 3D array having the xyz dimensions of the room
60     #fs is the sampling frequency
61     #data is the length N array of the wav data of the source
62     #sloc is a Nx3 array of the source location in xyz
63     #delay is how much to wait before start playing the source
64     #miccent is a 2D array of the center of the mic array
65     #mich is the height at which the mic array is placed
66     #deg is an array with the degrees of each source
67     #snr is the SNR at the mic signals (default is set to None)
68     #N is the number of sources
69
70
71     e_absorption, max_order = pra.inverse_sabine(rt60, dim)
72     # creates room object
73     room = pra.ShoeBox(
74         dim, fs, materials=pra.Material(e_absorption), max_order=max_order
75     )
76
77
78     #adds sources to the room
79     for i in range(N):
80         room.add_source(sloc[:,deg[i]], signal= data[i], delay = delay)
81
```

```
82      #generates UCA mic lcoations, (change 6 with your number of microphones and 0.0463
        with the UCA radius)
83      G = pra.beamforming.circular_2D_array( miccent , 6 , 0 , 0.0463 )
84      mic = []
85
86
87      for i in range(0,6):
88          c = [ G[0][i],G[1][i], mich]
89          mic.append(c)
90
91      mic_locs = np.c_[mic[0],mic[1],mic[2],mic[3],mic[4],mic[5]]
92      #adds the microphone locations in the room
93      room.add_microphone_array(mic_locs)
94
95      # finds the room impulse responses
96      room.compute_rir()
97      #simulates the room for a given SNR
98      room.simulate(snr)
99      #returns the signals captured by the microphone array
100     return room.mic_array.signals
101
102
103
104 def pyrec(fs,duration,channels =1):
105     #Function that records multi channel audio using pyaudio
106
107     #fs = sampling frequency
108     #duration = recording duration
109     #channels = number of microphone inputs (defaults to 1)
110
111     CHUNKSIZE = int(fs*duration) # fixed chunk size depending on sampling frequency and
        recording duration
112
113     #open stream
114     p = pyaudio.PyAudio()
115     stream = p.open(format=pyaudio.paInt16, channels=channels, rate= fs, input=True,
        frames_per_buffer=CHUNKSIZE)
116
117     #read the microphone input calues and modify in an channelsXchunksize shape numpy
        array
118     data = stream.read(CHUNKSIZE)
119     numpydata = np.frombuffer(data, dtype=np.int16)
120     numpydata = np.reshape(numpydata, (CHUNKSIZE, channels))
121     numpydata = [numpydata[:,i] for i in range(channels)]
122
123     # close stream
124     stream.stop_stream()
125     stream.close()
126     p.terminate()
127     return numpydata
```

```
128
129
130 def gcc_phat(sig, refsig, fs, max_tau=None, interp=4):
131     '''
132     I modified this function from the original:
133
134     Copyright (c) 2017 Yihui Xiong
135      Licensed under the Apache License, Version 2.0 (the "License");
136     you may not use this file except in compliance with the License.
137     You may obtain a copy of the License at
138     http://www.apache.org/licenses/LICENSE-2.0
139
140
141     This function computes the offset between the signal sig and the reference signal
        refsig
142     using the Generalized Cross Correlation - Phase Transform (GCC-PHAT)method.
143     '''
144
145     # make sure the length for the FFT is larger or equal than len(sig) + len(refsig)
146     n = sig.shape[0] + refsig.shape[0]
147
148     #find next power of 2 to make FFT faster
149     n = npow2(n)
150
151     # Generalized Cross Correlation Phase Transform
152     #Uncomment to add window, multiply the signals sig and refsig with the window
153     #window = np.hamming(len(sig))
154
155     SIG = np.fft.rfft(sig, n=n)
156     REFSIG = np.fft.rfft(refsig, n=n)
157
158     R = SIG * np.conj(REFSIG)
159     #phat weighting
160     ab = np.abs(R)
161     ab[ab < 1e-10] = 1e-10
162     #obtains gcc
163     cc = np.fft.irfft(R / np.abs(ab), n=npow2(interp * n))
164
165     #Uncomment below to normalize
166     #cc = cc/np.max(cc)
167
168     max_shift = int(interp * n / 2)
169     if max_tau:
170         max_shift = np.minimum(int(interp * fs * max_tau), max_shift)
171
172     cc = np.concatenate((cc[-max_shift:], cc[:max_shift+1]))
173
174     # find max cross correlation index (TDOA)
175     shift = np.argmax(np.abs(cc)) - max_shift
176
```

```
177     tau = shift / float(interp * fs)
178 
179     return tau, cc
180 
181 
182 def gccfb(sig, refsig, fs, max_tau=None, interp=4,nmel = 40):
183     '''
184     I modified this function from the original:
185 
186     Copyright (c) 2017 Yihui Xiong
187     Licensed under the Apache License, Version 2.0 (the "License");
188     you may not use this file except in compliance with the License.
189     You may obtain a copy of the License at
190     http://www.apache.org/licenses/LICENSE-2.0
191 
192 
193     This function computes the GCC-PHAT function in mel filterbanks
194     returns the filterbanks as well as the X,Y to plot them with plt.pcolormesh(X,Y,GCCFB
    )
195     '''
196 
197     # make sure the length for the FFT is larger or equal than len(sig) + len(refsig)
198     n = sig.shape[0] + refsig.shape[0]
199 
200     #find next power of 2 to make FFT faster
201     n = npow2(n)
202 
203     # Generalized Cross Correlation Phase Transform
204     #Uncomment to add window, multiply the signals sig and refsig with the window
205     #window = np.hamming(len(sig))
206 
207     SIG = np.fft.rfft(sig, n=n)
208     REFSIG = np.fft.rfft(refsig, n=n)
209 
210     R = SIG * np.conj(REFSIG)
211     #phat weighting
212     ab = np.abs(R)
213     ab[ab < 1e-10] = 1e-10
214 
215     #gets the mel filters
216     M = librosa.filters.mel(fs, n_fft = n, n_mels= nmel, fmin= 100, fmax= 8000, htk=False
    , norm= None)
217     M[M < 1e-9] = 1e-9
218 
219 
220     max_shift = int(interp * n / 2)
221     if max_tau:
222         max_shift = np.minimum(int(interp * fs * max_tau), max_shift)
223 
224     GCCFB= []
```

```
225     #Finds the GCC-PHAT for every fiterbank
226     for l in range(40):
227         M[l] = M[l]/np.max(M[l])
228 
229         ccc = np.fft.irfft(M[l]*R / np.abs(ab), n=(interp * n))
230         cc = ccc/np.sum(M[l])
231         cc = np.concatenate((cc[-max_shift:], cc[:max_shift+1]))
232         GCCFB.append(cc)
233     GCCFB = np.array(GCCFB)
234     G = -len(cc)/(2*16*fs)
235     Y = np.linspace(G,-G,len(cc))
236     X = np.linspace(1,40,40)
237 
238     return X,Y,GCCFB
239 
240 
241 
242 def myinwhole(Dat,fs, mics = 6):
243     # function that generates the GCC-PHAT function for all different microphone pair
    combinations
244 
245     #Dat = input audio data (data must be of shape mics X any audio length)
246     ind = list(itertools.combinations(range(mics),2))
247     GCC= []
248 
249     for i,j in ind:
250         t,c =  gcc_phat(Dat[i] , Dat[j], fs , 0.0003)
251 
252         GCC.append(c)
253 
254     GCC = np.array(GCC)
255     GCC = np.transpose(GCC)
256     #Change the 4 below with the interpolation factor given in gccphat() function
257     G = -len(c)/(2*4*fs)
258     #X and Y are included to be able to print the input image with matplotlib.pyplot.
    colormesh(X,Y,GCC)
259     Y = np.linspace(G,-G,len(c)+1)
260     X = np.linspace(1,len(ind),len(ind)+1)
261     return X , Y , GCC
262 
263 #DOA estimation algorithm code
264 def power(a,fs):
265     #function that finds which element in an array has the max power and returns its
    index
266 
267     #a = array
268     #fs = sampling frequency
269     P = []
270     for i in range(len(a)):
271         x = sum(np.square(a[i]))*fs/len(a[i])
```

```
272         P.append(x)
273         ind = np.argmax(P)
274     return ind
275 
276 def mic_index(i,n =6):
277     #This function reorders the channels with respect to the reference microphone
278 
279     # i is the index of the microphone with the maximum power
280     # n is the number of microphones in my mic array
281     #ind has all the different possible mic combinations (the order matters as the neural
         network inputs are generated in this sequence)
282     ind = np.array(list(itertools.combinations(range(n),2)))
283     fixed = (ind + i)%n #modulus operation wiht base 6 returns the mic indexes rotated so
         that mic with max power is at position of 0
284     return fixed
285 
286 def beamform(a,k,mat):
287     #This function performs phase shift beamforming to a multichannel audio input
288 
289     #a = audio input
290     #k is the column that has the direction in the beamforming weights matrix
291     #mat = beamforming weights matrix of shape kXchannels can be obtained from MATLAB and
         imported as
292     #mat = scipy.io.loadmat('beamformerwt1.mat')
293     #mat = mat['wt']
294 
295     Y = 0
296     for i in range(len(a)):
297         #finds frequency spectrum of audio signal for each microhpone
298         A = np.fft.rfft(a[i])
299         #selects the weights for each microphone in direction k
300         w = mat[i,k]
301         #multiplies frequency spectrum by complex beamformign weights
302         x = A*w
303         #Sums to get the frequency spectrum of the beamformed signal
304         Y += x
305     #retrieve the beamformed signal in time
306     #(This step can be skipped and return Y instead to find the maximum energy according
         to Parsevals theorem)
307     y = np.fft.irfft(Y)
308     return y
309 
310 def order(a):
311     #reorders the microphone inputs to match the simulated data
312     #( This is inlcuded because the channel ordering of the Respeaker is clockwise
313     #while the simulated channel order by librosa is counter clockwise)
314     o = [a[0],a[5],a[4],a[3],a[2],a[1]]
315     return o
316 
317 def prep(fs,duration):
```

```
318     channels = 6
319     mat = scipy.io.loadmat('beamformerwt1.mat')
320     mat = mat['wt']
321     #records audio data
322     a = pyrec(fs,duration,channels)
323     #change the order of channels to match training data and beamforming weights
324     a = order(a)
325     #Do beamforming for the direction of the microphones
326     y = [beamform(a,i,mat) for i in range(6)]
327     #Finds which microphone is clsoer to the source
328     imax = power(y,fs)
329     #changes the order of the channels wrt to ref microphone to be used in the model
330     newind = mic_index(imax)
331     #finds the GCC values that are fed to the model
332     X,Y,GCC = myinwhole(a,fs,newind)
333     GCC = np.array([GCC], dtype=np.float32)
334     return X,Y,GCC,imax
335 
336 def setinterpreter(path, inp):
337     #sets the tflite interpreter so that the model can make its prediction, passes the
        input to the model and returns the prediction
338 
339     #path is the path to the .tflite model
340     #inp = input on which the prediction should be made (gcc-phat matrix of myinwhole()
        function)
341 
342     interpreter = tflite.Interpreter(model_path= path)
343     interpreter.allocate_tensors()
344     input_details = interpreter.get_input_details()
345     output_details = interpreter.get_output_details()
346     input_shape = input_details[0]['shape']
347     interpreter.set_tensor(input_details[0]['index'], inp)
348     interpreter.invoke()
349     output_data = interpreter.get_tensor(output_details[0]['index'])
350     return output_data
351 
352 def doaconv(i,j):
353     #Converts the output of the beamformer and the NN to 360 degrees with 10 degrees
        resolution
354 
355     #i is the output of the beamformer
356     #j is the output of the neural network
357     res = 10
358     doa  = np.arange(-25,25+res,res)
359     off = i * 60
360     deg = off + doa[j]
361     return deg
362 
363 def doaconv60(i,j):
364     #Converts the output of the beamformer and the NN to 360 degrees with 1 degree
```

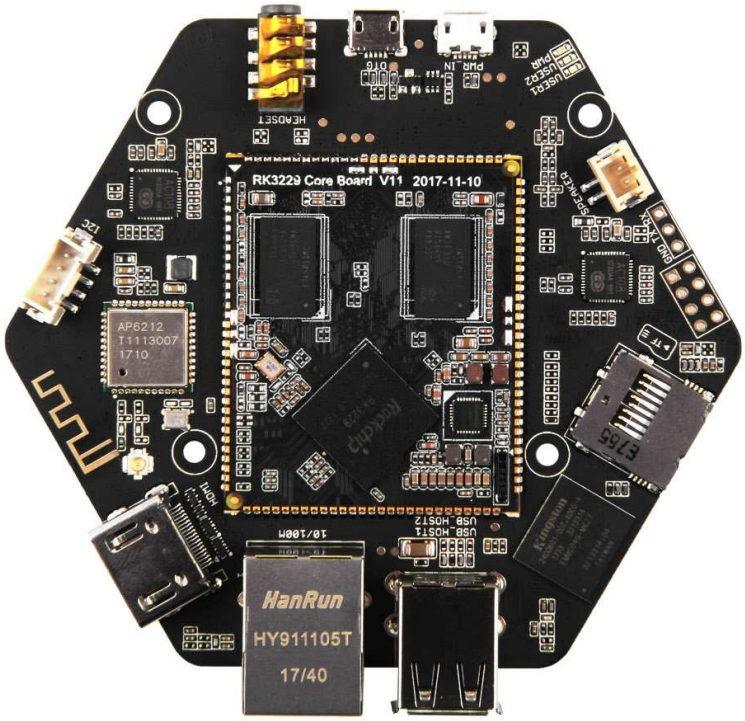
```
    resolution
365
366     #i is the output of the beamformer
367     #j is the output of the neural network
368     res = 1
369     doa  = np.arange(-30,30,res)
370     off = i * 60
371     deg = off + doa[j]
372     return deg
373 def PointsInCircum(r,n=100):
374     pi = math.pi
375     return [(math.cos(2*pi/n*x)*r,math.sin(2*pi/n*x)*r) for x in range(0,n+1)]
376
377
378 def doa(a,fs,path,mat):
379     #Example function on how beamforming can be done for audio data
380
381     #a = input audio data
382     # fs = sampling frequency
383     #mat = beamforming weights matrix of shape kXchannels can be obtained from MATLAB and
         imported as
384     #mat = scipy.io.loadmat('beamformerwt1.mat')
385     #mat = mat['wt']
386     #path = the path to the .tflite model
387
388     #change the order of channels to match training data and beamforming weights
389     a = order(a)
390     #Do beamforming for the direction of the microphones
391     y = [beamform(a,i,mat) for i in range(6)]
392     #Finds which microphone is clsoer to the source
393     imax = power(y,fs)
394     #changes the order of the channels wrt to ref microphone to be used in the model
395     newind = mic_index(imax)
396     #finds the GCC values that are fed to the model
397     X,Y,GCC = myinwhole(a,fs,newind)
398     GCC = np.array([GCC], dtype=np.float32)
399     nnres = np.argmax(setinterpreter(path,GCC))
400     doa = doaconv(imax,nnres)
401     return doa,imax
402
403 def MLP():
404     #MLP NN training code
405     #sets paths to save the model and the tensorboard logs
406     tbpath = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
407     mdpath = "logs/train/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S") +'model.{
        epoch:02d}.h5'
408
409     #sets tensorboard logs callback
410     tb_callback = tf.keras.callbacks.TensorBoard(tbpath,
411         histogram_freq=0,
```

```
412         write_graph=True,
413         write_images=True,
414         update_freq="epoch",
415         profile_batch=2
416     )
417 
418     #model callback that saves the model when validation lossdecreases
419     model_callback = tf.keras.callbacks.ModelCheckpoint(
420         mdpath,
421         monitor="val_loss",
422         verbose=1,
423         save_best_only= True,
424         save_weights_only=False,
425         mode="auto",
426         save_freq="epoch",
427         options=None
428     )
429 
430     #early stopping callback that stops training if validation loss does not decrease for
     more than 5 epochs
431     es_callback = tf.keras.callbacks.EarlyStopping(
432         monitor="val_loss",
433         min_delta=0,
434         patience=5,
435         verbose=1,
436         mode="auto",
437         baseline=None,
438         restore_best_weights=False,
439     )
440     #Sequential MLP model for 39x15 input
441     model = tf.keras.models.Sequential()
442     model.add(tf.keras.Input(shape=(39,15)))
443     model.add(tf.keras.layers.Dense(200, activation='relu'))
444     model.add(tf.keras.layers.BatchNormalization())
445     model.add(tf.keras.layers.Dense(150, activation='relu'))
446     model.add(tf.keras.layers.BatchNormalization())
447     model.add(tf.keras.layers.Dense(100, activation='relu'))
448     model.add(tf.keras.layers.BatchNormalization())
449     model.add(tf.keras.layers.Flatten())
450     model.add(tf.keras.layers.Dense(60, activation = 'softmax'))
451     model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'
    ])
452     model.optimizer.lr.assign(0.0005)
453 
454     #Train model
455     history = model.fit(Training_data, training_labels, epochs=100, batch_size=32,shuffle
     = True, validation_data=(validation_data, validation_labels),callbacks=[
    model_callback,tb_callback,es_callback])
```

Appendix C

Respeaker specifications

ReSpeaker Core v2



Seeed ReSpeaker Core v2 is designed for voice interactive applications. It is based on quad-core ARM Cortex-A7, up to 1.5Ghz, and 1GB RAM on-board. Besides, it features six microphone array with necessary speech algorithm, like DoA(Direction of arrival), BF(Beam-Forming), AEC(Acoustic echo cancellation) and etc.

ReSpeaker Core v2 runs GNU/Linux operation system. It benefits from powerful and active community, we can use lot of existing software/tools for development, testing and deploy, so that rapid product development become available.

ReSpeaker Core v2 is not only designed for makers/enthusiast, but also a turnkey solution for business company. The hardware consists of two parts, one is the minimized SoC module which is small and easy for manufacturing and ready for final product, the other is a bottom board which can be full customizable.

Get One Now

Features

- High performance SoC
- 1GB RAM & 4GB eMMC
- 6 Microphone Array
- USB OTG, USB device
- WiFi b/g/n and BLE 4.0
- Detect range: ~5 meters
- Grove socket for other sensor
- 3.5mm audio jack & JST2.0 connector
- 8 channel ADCs for 6 microphone array and 2 loopback (hardware loopback)

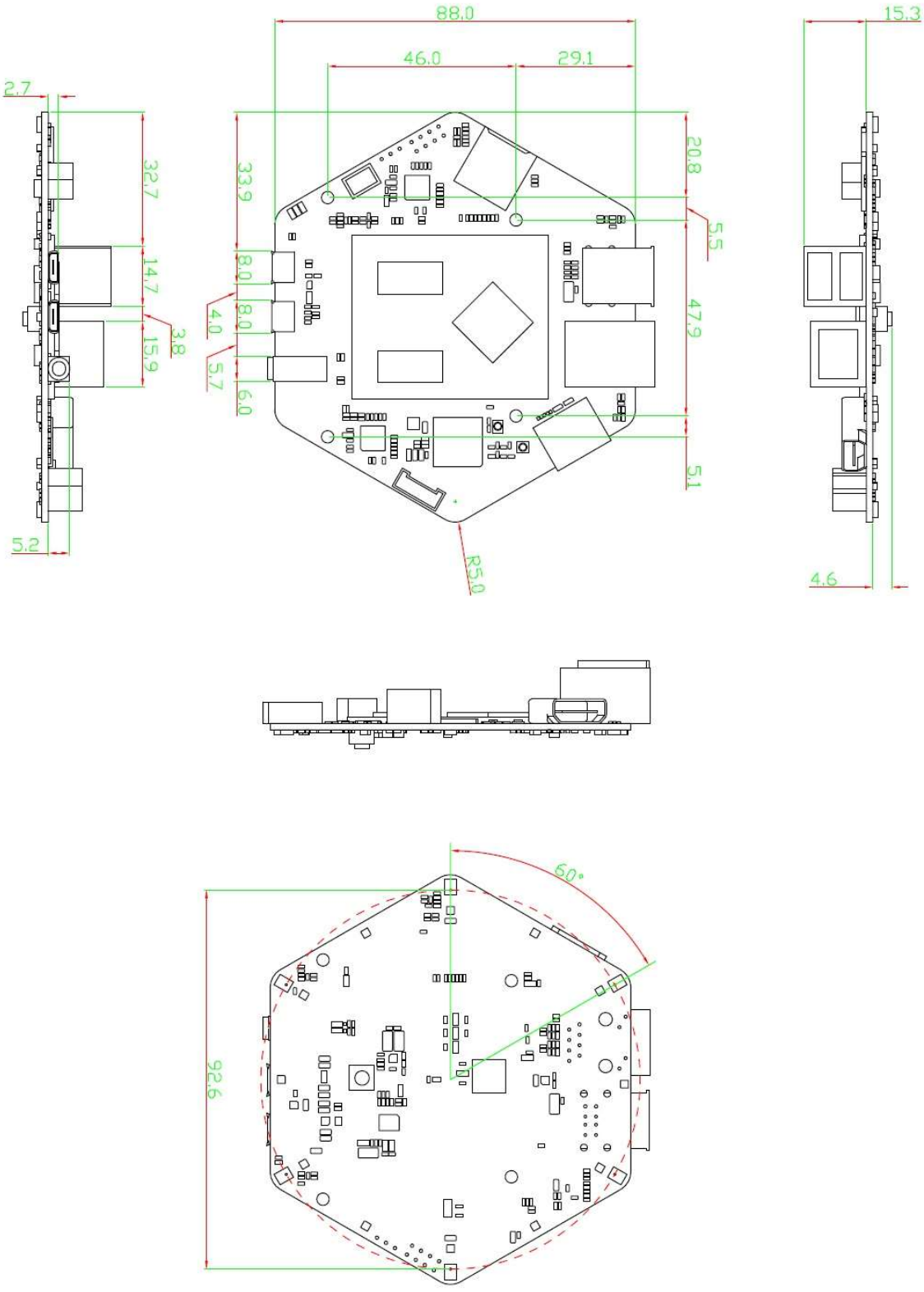
- Debian-based Linux system
- SDK for speech algorithm with Full documents
- C++ SDK and Python wrapper
- Speech algorithms and features
- Keyword wake-up
- BF(Beam-Forming)
- DoA (Direction of arrival)
- NS(Noise suppression)
- AEC (Acoustic echo cancellation) and AGC (Automatic gain control)

Specification

Features		
Soc(Rockchip RK3229)	CPU	Quad-Core Cortex-A7,up to 1.5GHz
	GPU	Mali400MP, Support OpenGL ES1.1/2.0
	Memory	1GB RAM(Core Module includes RAM and PMU)
	System	Operating Voltage:3.6-5V
		80 pins on-module
		PMU on-module
Peripheral	Networks	WiFi b/g/n; BLE 4.0; Ethernet
	USB	2 x USB Host; 1 x USB OTG; 1 x USB power
	Grove	1 x Grove socket (I2C and Digital)
	Vedio	HDMI 2.0 with HDCP 1.4/2.2, up to 4K/60Hz
	Audio	6 Microphone Array; 3.5mm Audio Jack; JST2.0 Audio output connector
	Storage	4GB eMMC on-board; SD slot
	Others	12 x RGB LEDs; 8 GPIO pins
Power Consumption	Standby Mode	360mA

Hardware Overview

Interface and storage



Applications

- Smart speaker
- Intelligent voice assistant systems