FACULTY OF ENGINEERING
AND ARCHITECTURE

Coherence-based methods for clustering ad-hoc distributed microphones

Martijn Meeldijk
Student number: 02111587

Supervisor: Prof. dr. ir. Nilesh Madhu
Counsellors: Stijn Kindt, Alexander Bohlender

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Information Engineering Technology

Academic year 2022-2023

GHENT
UNIVERSITY

Acknowledgements

//TODO

Toelichting in verband met het masterproefwerk

Deze masterproef vormt een onderdeel van een examen. Eventuele opmerkingen die door de beoordelingscommissie tijdens de mondelinge uiteenzetting van de masterproef werden geformuleerd, werden niet verwerkt in deze tekst.

Melding van vertrouwelijkheid (enkel indien van toepassing)

Bekijk hiervoor de informatie op de facultaire website - **Nota in verband met de vorm van de masterproef (alle opleidingen)**

Abstract

Meer informatie op https://masterproef.tiwi.ugent.be/verplichte-taken/ - Korte abstract (Nederlands en/of Engels)

Contents

List of Figures

List of Tables

List of Acronyms

ASN	Acoustic Sensor Network.
FFT	Fast Fourier Transform.
NMF	Nonnegative Matrix Factorization.
WASN	Wireless Acoustic Sensor Network.

List of Code Fragments

List of Algorithms

1

Introduction

1.1 The cocktail party problem

In today's world, we are constantly surrounded by various types of sounds, whether it be in a crowded party or a busy street. However, these sounds can often interfere with one another, creating a challenging problem known as the cocktail party problem. This problem arises when several sound sources are present in an environment, where they are mixed together to create a single audio signal, making it difficult to distinguish the signals coming from individual sound sources.

For instance, imagine you are in a crowded party where several conversations are happening simultaneously, and music is playing in the background. You are trying to have a conversation with someone, but their voice is barely audible amidst the surrounding noise. This scenario clearly illustrates the cocktail party problem, where sound waves from multiple sources are mixed together, creating a complex and challenging signal processing problem.

The same challenge also arises in many other real-world scenarios, such as in hearing aids, speech recognition, and surveillance systems. In these scenarios, the goal is to extract and enhance specific sound sources from a noisy environment. However, this task is not trivial, as the signals are often mixed together in a non-linear and time-varying manner. Therefore, developing efficient and accurate signal processing algorithms to separate sound signals from multiple sources is a critical problem in various fields.

1.2 The cocktail party solution

One solution is to make use of microphone arrays in order to help extract the wanted speech component from the signal. A microphone array consists of multiple microphones placed close to each other, typically inside the same device. From the microphone array's signals, time-frequency masks can be created. These are subsequently used in combination with the short-time Fourier transform of the signal to extract the speech component.

To further improve on this solution, it is possible to make use of (ad hoc) distributed microphone arrays. This way, several microphones or microphone arrays are placed at different locations, opening up new possibilities for signal processing such as utilizing the different amplitudes of the signals received in different locations. Signal capture using ad hoc distributed microphones, or acoustic sensor networks (ASNs), is an active and rapidly expanding field of research. With the inclusion of microphones in an increasing variety of smart devices, distributed audio capture is becoming increasingly available - with potential for application in a wide range of fields such as surveillance for assisted living and healthcare, hearing aids, communications. The challenges, however, are also manifold. Compared to traditional, compact microphone arrays, where multiple microphones are placed close to each other with predefined geometries, the relative locations of sensors are not known a priori, and their placement with respect to audio sources of interest can be arbitrary. The processing power and bandwidth available to each node can also be limited, constraining on-edge processing and data communication with a central hub.

Since nodes in a WASN don't have predefined geometries and are at most weakly synchronized, it is necessary to develop algorithms that can handle these constraints. In many cases, the first step is to perform a clustering procedure to divide the nodes in groups assigned to the different sources. An additional cluster is sometimes added for nodes dominated by reverberation and background noise.

This thesis aims to contribute to this field by performing an in-depth analysis of several existing solutions, as well as a comparison between the performance of several methods. An emphasis is laid on clustering methods using ad-hoc distributed microphone arrays.

1.3 Summary of results

//TODO

2

Background

2.1 Signal Model

The acoustic environment considered in this thesis consists of N acoustic sources and D microphones which are distributed, typically in an unknown arrangement, within the boundaries of the environment. The signal received by a microphone d may be described in continuous time t as:

$$x_d(t) = \sum_{n=1}^{N} \int_0^{\infty} h_{nd}(\tau) s_n(t-\tau) d\tau \tag{2.1}$$

- $s_n(t)$: the n-th source signal
- $h_{nd}(t)$: the impulse response from source n to microphone d
- $x_d(t)$: the resulting microphone signal

This can displayed by making use of the convolution operator like so:

$$x_d(t) = \sum_{n=1}^{N} h_{nd}(t) * s_n(t) \tag{2.2}$$

The microphone signals are sampled and transformed to the short-time discrete Fourier domain.

$$X_d(k,b) = \text{STFT}[x_d(l)] \tag{2.3}$$

- $x_d(l)$: the sampled signal of microphone d
- l: the time sample index
- k: the frequency bin index
- b: the time frame index

As of now, this model doesn't take into consideration possible interference or noise. This will be discussed in the next paragraph.

2.1.1 Interference

//TODO

2.2 Features

In the realm of ad-hoc microphone clustering, the process of feature extraction prior to performing the clustering is a commonly used approach. Such a technique is often required for the employed clustering algorithm, but also brings with it the added advantage of avoiding the need to transfer encoded signals from individual nodes to a central hub. Feature extraction serves to transform the raw data obtained from the individual microphones into a format that is more amenable to clustering. Once the features have been extracted, they can be fed into the clustering algorithm to create clusters based on the inherent similarities between the individual features.

The benefit of this approach is particularly relevant for scenarios where the individual nodes are geographically dispersed and/or the data transfer rates are limited, which are typical scenarios for ad-hoc distributed microphones. In such cases, the extraction of features prior to transmission helps to minimize the amount of data that needs to be transferred, thus reducing the overall transmission time and bandwidth requirements. Additionally, the use of feature extraction methods can enhance the robustness and accuracy of the clustering process, thereby improving the overall quality of the results.

It is worth noting, however, that the choice of feature extraction technique can have a significant impact on the final clustering results. Consequently, careful consideration and selection of the appropriate feature extraction method is crucial to the success of the overall clustering process.

2.2.1 MFCC-based features

One way to extract features from signals is to consider their cepstro-temporal representations, which are called Modulation Mel-Frequency Cepstral Coefficients (Mod-MFCCs) features [2].

2.2.1.1 Mel-frequency cepstrum and the mel scale

Computing the inverse Fourier transform of the logarithm of the spectrum of a signal results in a cepstrum [3]. The mel-Frequency Cepstrum (MFC) is another way to represent the short-term power spectrum of a sound signal. It is essentially a way of representing a sound signal in a manner that approximates the human auditory system's response [4]. The MFC differs from the cepstrum due to the use of the Mel scale [5], which rescales the normal frequency scale so that pitches are perceived (by humans) to be equal in distance from one another. One way to convert f herts into m mels is:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.4}$$

2.2.1.2 Mod-MFCC features

The first step in computing the Mod-MFCC features consists of sampling the audio signal $x(t)$ with sample rate f_s, after which it can be represented in a time-discrete manner by $x(l)$. Consequently, this representation is segmented into B (possibly overlapping) frames with length Y using a window function, where after the discrete Fourier transform (DFT) of the weighted frames is calculated:

$$X(k,b) = \sum_{y=0}^{Y-1} x(bP+y)\mathcal{W}(y)e^{-j2\pi yk/Y} \tag{2.5}$$

- l: The discrete time index
- B: The amount of frames
- $\mathcal{W}(y)$: The window function, with $y = 1, 2, \cdots, Y$
- b: The frame index, with $b = 0, 1, \cdots, B-1$
- k: The frequency bin index, with $k = 0, 1 \cdots, K-1$

The magnitude-squared spectrum $|X(k,b)|^2$ is transformed to the mel scale, outlined in paragraph 2.2.1.1. This transformation is accomplished by using overlapping triangular windows $\mathcal{V}_{k'}(k)$ that function as band-pass filters [6]. This results in a mel-spectrum:

$$X_{\text{mel}}(k',b) = \sum_{k=0}^{Y/2} |X(k,b)|^2 \mathcal{V}_{k'}(k) \tag{2.6}$$

- k': The mel scale frequency bin index, with $k' = 0, 1 \cdots, K'-1$

Calculation of the MFCCs is done by computing the discrete cosine transform (DCT) of the logarithm of the mapped power spectrum.

$$X_{\text{mfcc}}(\eta,b) = \sum_{k'=0}^{K'-1} \log(|X_{\text{mel}}(k',b)|)\ \cos\left(\frac{\pi}{K'}(k'+0.5)\eta\right), \tag{2.7}$$

- η: The cepstral coefficient index

$$\hat{X}_{\text{mfcc}}(\nu,\eta,c) = \sum_{\ell=0}^{L-1} X_{\text{mfcc}}(\eta, cQ + \ell)e^{-j2\pi\,\ell\,\nu/L}, \tag{2.8}$$

$$\tilde{X}_{\text{mfcc}}(\nu,\eta) = \frac{1}{C_T}\sum_{c=0}^{C_T-1} |\hat{X}_{\text{mfcc}}(\nu,\eta,c)| \tag{2.9}$$

$$r_{\nu_1|\nu_2}(\eta) = \frac{\sum_{\nu=\nu_1}^{\nu_2} \tilde{X}_{\text{mfcc}}(\nu, \eta)}{(\nu_2 - \nu_1 + 1)\tilde{X}_{\text{mfcc}}(0, \eta)} \tag{2.10}$$

$$\overline{X}_{\text{mfcc}}(\eta) = \frac{1}{L} \sum_{\nu'=0}^{L-1} \tilde{X}_{\text{mfcc}}(\nu', \eta) \tag{2.11}$$

2.2.2 Features from speaker embeddings

An alternative to mod-MFCC-based features is to use embeddings generated by speaker verification networks. These often lead to more robust features for clustering.

After training a neural network to recognize speakers, it becomes possible to extract low-dimensional speaker embeddings from the bottleneck layer that precedes the network's output layer. These embeddings are then utilized to characterize the speaker in the input recording. To verify whether two recordings are from the same speaker, the embeddings extracted from an enrollment and a test recording are compared, and a hypothesis is formed. This comparison can be carried out using a simple cosine distance measurement [1].

This can be applied to ad-hoc microphone arrays. In speaker verification tasks, the embeddings are used to test if two audio samples are are spoken by the same person, by computing a similarity metric. Creating embeddings for microphones in the ad-hoc scenario allows for the calculation of similarity metrics between microphones, meaning that microphones dominated by the same speaker should yield nearly identical embeddings and thus a high similarity measure [7].

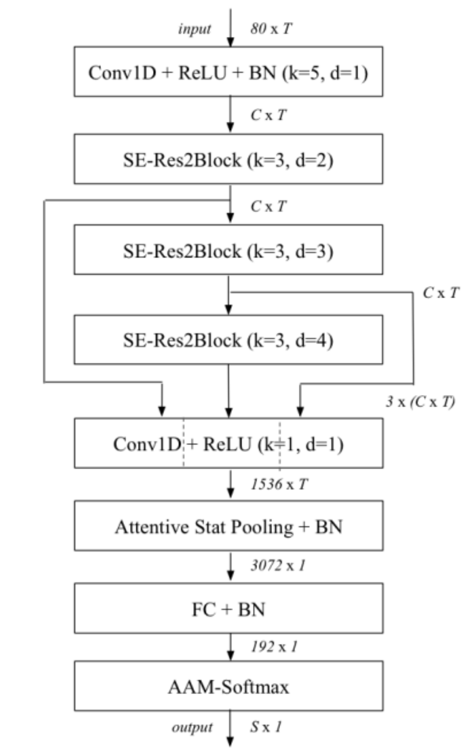
For clustering, the embeddings from each microphone are used as input features for the algorithm. These are generated from the Emphasized Channel Attention, Propagation and Aggregation Time Delay Neural Network (ECAPA-TDNN) [1], which improves upon the x-vector architecture [8] with several enhancements. The network topology as described in [1] is provided in Figure 2.1 for illustrative purposes. In essence, ECAPA-TDNN does three things to make better embeddings:

- It pays more attention to important parts of the voice data, which is achieved with an attentive statistics pooling layer.
- It adds more context to the voice data by looking at other parts of the recording by using a speech adapted version of Squeeze-Excitation [9]. The SE technique works by looking at different parts of the voice data and deciding which parts are more important. The "squeeze" part of SE compresses the data, and the "excitation" part expands it again, with more emphasis on the important parts.
- It combines different layers of information to create a more complete picture of the voice. Multilayer feature aggregation before the pooling layer gives the model the opportunity to incorporate information learned from multiple levels in the network.

This allows for the usage of cosine similarity to compare two speaker embeddings, which can in turn be used as a distance metric for clustering algorithms. This will be elaborated upon in section 2.3.1.

Chapter 4 provides an in-depth comparison of the methods based on mod-MFCC features from [2] and speaker embeddings from [7, 10] against the methods proposed in this thesis.

(a) Network topology of the ECAPA-TDNN. k denotes kernel size and d denotes dilation spacing of the Conv1D layers or SE-Res2Blocks. C and T correspond to the channel and temporal dimension of the intermediate feature-maps respectively. S is the number of training speakers.



(b) The SE-Res2Block of the ECAPA-TDNN architecture. The standard Conv1D layers have a kernel size of 1. The central Res2Net [16] Conv1D with scale dimension $s = 8$ expands the temporal context through kernel size k and dilation spacing d.

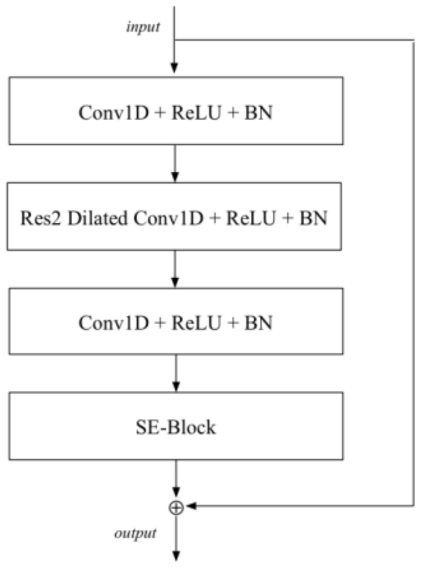


Figure 2.1: The network topology of the ECAPA-TDNN [1]

2.3 Clustering source-dominated microphones

A first key step in separating sources captured by ad-hoc distributed microphones is the assignment of said microphones to the appropriate clusters, albeit source-dominated or background clusters. Usually, $N + 1$ clusters are constructed. One cluster for each of the N sources, and one additional background cluster for microphones dominated by background noise and interference. It is worth noting that not all of the algorithms described below necessitate the use of a background cluster. An example solution to a clustering scenario with 2 sources, 15 microphones and 3 clusters is displayed in figure 2.2 Many methods exist for performing this procedure, some of which shall be discussed later on.

One method suggests estimating clusters with fuzzy c-Means based on a feature set composed of MFCC's and their modulation spectra [11]. A variation on this method applies speaker embedding frameworks instead of hand-engineering embeddings from MFCC's[10], while a somewhat different approach utilizes a coherence based method and non-negative matrix factorization (NMF) for determining clusters [12]. When taking into account the privacy-related challenges imposed by ad-hoc microphone arrays, a more privacy-aware method based on unsupervised federated learning demonstrates promising results[13].

The subsequent sections of this chapter will delve deeper into the technical nuances of several of the aforementioned techniques, expounding on their theoretical underpinnings to provide a more comprehensive understanding. The aim of this analysis is to shed light on the intricate workings of each approach and to provide valuable insights into their practical applications.

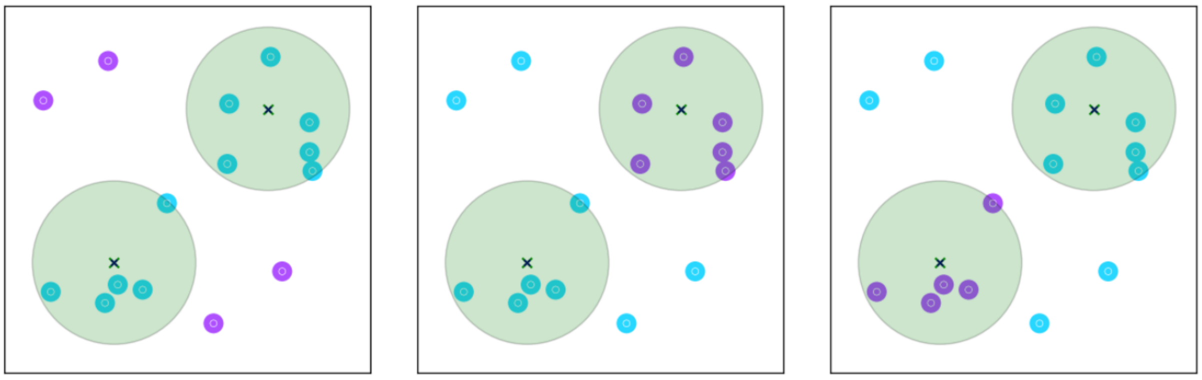


Figure 2.2: Example solution to a clustering scenario, where the microphones are represented by dots. The sources are denoted by a cross, with their critical distances marked in green.

2.3.1 Fuzzy clustering

The following method involves conducting a fuzzy clustering of audio features obtained from microphone signals. The resulting fuzzy membership values are then evaluated to categorize the microphones into either a source-dominated or a background cluster. This procedure enables the implementation of fuzzy-membership value (FMV) aware delay-and-sum beamforming for source separation, which will be eleborated upon in section 2.4.1.

2.3.1.1 Fuzzy c-Means

//TODO more about the algorithm //TODO cossim

The first step in the fuzzy clustering procedure consists of extracting a feature set composed of Mel-frequency cepstral coefficients (MFCCs) and their modulation spectra. This is essentially a way of representing a sound signal in a manner that approximates the human auditory system's response to the signal. [4] This feature set is computed across signal segments of 4s, after which the effects of reverberation are reduced via cepstral mean normalization. For each signal segment, a feature vector is generated for each of the D microphones. These vectors, denoted by $\mathbf{v}_d$, consist of A features. [2].

The next step is to estimate clusters of microphones dominated by one of the sources. Several algorithms exist to estimate an optimal fuzzy partition of the set of observations. A well studied and popular method is the Fuzzy c-Means algorithm (FCM) , which evaluates the least-squared error functional:

$$J_m(\Delta, \mathbf{u}) = \sum_{d=1}^{D}\sum_{n=1}^{N}(\mu_{nd})^{\alpha}||\mathbf{v}_d - \mathbf{u}_n||_{\beta}^{2} \tag{2.12}$$

The value $\mu_{n,d}$ denotes the fuzzy membership value. The matrix ∇ contains all the $\mu_{n,d}$ and is iteratively estimated. An example is shown in table 2.1.

	Cluster n		
Mic. d	1	2	3
1	0.1	0.3	**0.6**
2	**0.5**	0.3	0.2
3	**0.7**	0.2	0.1
4	0.25	0.2	**0.55**
5	0.25	**0.6**	0.15
6	0.15	**0.65**	0.2
7	0.05	0.15	**0.8**

Table 2.1: Example fuzzy membership values for 7 microphones and 3 clusters

2.3.2 Federated Learning

The increasingly declining cost of acoustic sensors and the rapid rise in popularity of wireless networks and mobile devices have aided in providing the technological infrastructure necessary for WASNs. Examples of scenarios where these can be beneficial range from smart-homes and ambient-assisted living to machine diagnosis and surveillance. ASN applications typically have to deal with multiple simultaneously active sound sources, resulting in the fact that an exchange of information about microphone positions or information-rich signal representations is often necessary. Transferring such data over a potentially insecure wireless network carries a considerable amount of risk regarding privacy. Even in the context of a small-scale environment such as a smart-home, the act of eavesdropping by an unauthorized user who gains access to the network can result in significant privacy risks through the interception of sensitive data. Furthermore, as the importance of privacy in our world increases, and privacy regulations such as the European Union General Data Protection Regulation (EU GDPR) arise, the need for a more privacy-aware solution becomes almost undeniable.

Clustered federated learning (CFL) [13] aims to provide such a solution. Instead of using feature representations derived from raw audio data, this methods solely requires ASN nodes to share locally learned neural network parameter updates with a central node. So far, federated learning has only been used in (semi-) supervised learning applications where (weak) classification labels were available, with its intended practical use consisting of massively distributed systems that handle large amounts of data [14]. The task of adapting CFL to an unsupervised scenario and implementing it effectively within the context of ASNs is a complex and challenging endeavor.

2.3.2.1 Federated learning

Federated learning operates by following a three-step iterative procedure over a certain amount of communication rounds τ. The initial stage involves the synchronization of clients with the server, accomplished by downloading the most recent model parameters represented by the column vector θ. Secondly, each client i improves its own model parameters $\boldsymbol{\theta}_i^\tau$ independently with stochastic gradient descent (SGD) on their data $\boldsymbol{D}_i$. Finally, each client uploads their model parameters updates $\boldsymbol{\Delta\theta}_i^\tau$ to the server, where they are aggregated according to

$$\boldsymbol{\theta}^{\tau+1} = \sum_{i=1}^{M} \frac{|\boldsymbol{D}_i|}{|\boldsymbol{D}|} \boldsymbol{\Delta\theta}_i^\tau \tag{2.13}$$

- M: The number of clients
- $|\boldsymbol{D}_i|$ The cardinality of the dataset of the i-th client
- $|\boldsymbol{D}|$: The cardinality of the total dataset

In cases where the clients' data originates from incongruent distributions, it is shown [15, 16] that there exists no single set of parameter updates able to optimally minimize the loss of all clients simultaneously. The suggested approach is to cluster clients following similar distributions, and training separate server models for each cluster. The first step in the procedure involves calculating the cosine similarity $a_{i,j}$ between the nodes' update vectors.

$$a_{i,j} = \frac{\langle \boldsymbol{\Delta\theta}_i, \boldsymbol{\Delta\theta}_j \rangle}{||\boldsymbol{\Delta\theta}_i, \boldsymbol{\Delta\theta}_j||} \tag{2.14}$$

- $\langle \cdot \rangle$ The inner product
- $||\cdot||$ The L_2 norm

Subsequently, these cosine similarities $a_{i,j}$ are collected in the symmetric matrix $\boldsymbol{A}$, on which hierarchical clustering using bi-partitioning can be applied. The two resulting clusters, c_1 and c_2, obtained from each bi-partitioning step are generated in such a way that the maximum cosine similarity between clusters is always less than the minimum cosine similarity within any of the two clusters.

$$\max_{\forall i \in c_1, k \in c_2} (a_{i,k}) < \min(\min_{\forall i,j \in c_1} (a_{i,j}), \min_{\forall k,l \in c_2} (a_{k,l})) \tag{2.15}$$

By recursively repeating this process, new sub-clusters are formed until the data distributions' congruence condition is no longer violated. This can be verified for each cluster c with

$$\Delta\overline{\theta}_c = \left\| \frac{1}{|c|} \sum_{i \in c} \boldsymbol{\Delta\theta}_i \right\| \text{ and } \Delta\hat{\theta}_c = \max_{i \in c}(\|\boldsymbol{\Delta\theta}_i\|) \tag{2.16}$$

which denote the mean and maximum Euclidian norms of the weight update vectors $\boldsymbol{\Delta\theta}_c$. A low value for $\Delta\overline{\theta}_c$ with a higher value for $\Delta\hat{\theta}_c$ are observed whenever the server had reached a stationary solution, but some clients are still converging to a locally stationary point. This indicates incongruent data distributions, which triggers another bi-partitioning step.

Algorithm 1 describes the procedure used in this technique. Before carrying out federated learning, a light-weight autoencoder h is pre-trained, after which all layers apart from the bottleneck layer are frozen. The bottleneck layer is reset with random parameters every time CFL is applied. This reduction is necessary to avoid overfitting, with the added benefit of reducing bandwidth usage and computational cost [17]. In addition to the incongruity verification based on the Federated learning stopping criterion $\epsilon_1 \geq \Delta\overline{\theta}_c$ and the clustering stopping criterion $\epsilon_2 \geq \Delta\hat{\theta}_c$ described in [15], a third verification $\epsilon_3 \leq |\nabla\Delta\overline{\theta}_c|$ is added [13]. This addition is based on the intuition that a slowing increase of $\Delta\overline{\theta}_c$ indicates the system nearing a stationary solution.

Table 2.2: Neural network architecture of autoencoder h

Layer	Input	Operator	Out ch.	Stride	Kernel/Nodes	Activation
1	128 x 128	Conv2d	6	1	5 x 5	ReLu
2	6 x 124 x124	MaxPool	-	2	2 x 2	-
3	6 x 62 x 62	Conv2d	16	1	5 x 5	ReLu
4	16 x 58 x 58	MaxPool	-	2	2 x 2	-
5	16 x 29 x 29	Dense	-	-	29	ReLu
6	16 x 29 x 29	Unpool	-	2	2 x 2	-
7	16 x 58 x 58	ConvTrans2d	6	1	5 x 5	ReLu
8	6 x 62 x 62	Unpool	-	2	2 x 2	-
9	6 x 124 x 124	ConvTrans2d	1	1	5 x 5	Sigmoid

2.3.2.2 Server pre-training

the architecture of autoencoder h is illustrated in Table 2.2 The autoencoder is trained to reconstruct the Log-Mel Band Energy (LMBE) input feature representation $\boldsymbol{Y}$. [18] Here, the mean squared error (MSE) between the input features and the reconstructed features serves as the loss function, which is minimized across the entire set of model parameters $\boldsymbol{\Theta}$.

$$\min_{\boldsymbol{\Theta}} L_{\text{mse}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \min_{\boldsymbol{\Theta}} \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{2.17}$$

2.3.2.3 Membership values

Cluster membership values are computed to assess the contribution of each node to its respective cluster. This happens after each bi-partitioning into clusters c_1 and c_2. For two clusters: Firstly, average intra-cluster similarities are calculated for each client i and arranged in the vector $\boldsymbol{q}$

$$q_i = \frac{1}{|c_x| - 1} \sum_{j \in c_x / i} a_{i,j} \tag{2.18}$$

Subsequently, average cross-cluster similarities are calculated for each client i and arranged in the vector $\boldsymbol{r}$

$$r_i = \frac{1}{|c_y|} \sum_{k \in c_y} a_{i,k} \tag{2.19}$$

The average intra- and cross-cluster are calculated for $\forall i \in c_x$ and $(c_x, c_y) \in \{(c_1, c_2), (c_1, c_1)\}$, where $|\cdot|$ denotes the cardinality.

Algorithm 1 Unsupervised CFL for the estimation of source-dominated microphone clusters in ASNs

```
Input: Pre-trained autoencoder $h$, thresholds $\epsilon_1$, $\epsilon_2$ and $\epsilon_3$,
maximum no. of rounds max$_\tau$
freeze all parameters of $h$ except $\boldsymbol{\theta}$
while audio buffer != empty do
    read audio $\boldsymbol{D}$ of $M$ clients
    initialize cluster list $C = \{\{1, \dots M\}\}$ with a single
    cluster element that contains all $M$ clients
    $C' = \{\}$
    $\boldsymbol{\theta}_c \leftarrow$ random initialization
    for $\tau = 1$ to max$_\tau$ do
        for $c \in C$ do
            for $i \in c$ do
                $\boldsymbol{\Delta\theta}_i^\tau \leftarrow$ SGD$(h_{\boldsymbol{\theta}_c}(\boldsymbol{D}_i)$
            end for
            $\Delta\overline{\theta}_c = \|\frac{1}{|c|}\sum_{i \in c} \boldsymbol{\Delta\theta}_i\|$
            $\Delta\hat{\theta}_c =$ max$_{i \in c}(\|\boldsymbol{\Delta\theta}_i\|)$
            if $\Delta\overline{\theta}_c \leq \epsilon_1$ & $\Delta\hat{\theta}_c \geq \epsilon_2$ & $|\nabla\Delta\overline{\theta}_c| \leq \epsilon_3$ then
                $a_{i,j} = \frac{\langle\boldsymbol{\Delta\theta}_i, \boldsymbol{\Delta\theta}_j\rangle}{||\boldsymbol{\Delta\theta}_i, \boldsymbol{\Delta\theta}_j||}$
                $c_1, c_2 \leftarrow$ bi-partition$(\boldsymbol{A})$
                $\boldsymbol{\theta}_{c_1}^{\tau+1} = \boldsymbol{\theta}_c^\tau + \sum_{i \in c_1} \frac{|\boldsymbol{D}_i|}{|\boldsymbol{D}_{c_1}|}\boldsymbol{\Delta\theta}_i^\tau$
                $\boldsymbol{\theta}_{c_2}^{\tau+1} = \boldsymbol{\theta}_c^\tau + \sum_{j \in c_2} \frac{|\boldsymbol{D}_j|}{|\boldsymbol{D}_{c_2}|}\boldsymbol{\Delta\theta}_j^\tau$
                $C' = C' + \{c_1, c_2\}$
                $\tau =$ max$_\tau + 1$
            else
                $\boldsymbol{\theta}_c^{\tau+1} = \boldsymbol{\theta}_c^\tau + \sum_{i \in c} \frac{|\boldsymbol{D}_i|}{|\boldsymbol{D}_c|}\boldsymbol{\Delta\theta}_i^\tau$
                $C' = C' + \{c\}$
            end if
        end for
        $C = C'$
    end for
end while
```

Next, min-max normalization is applied to obtain vector $\boldsymbol{p}$, containing the aggregated mean cosine similarity measure for each client:

$$p_i = \lambda \frac{q_i - \min(\boldsymbol{q})}{\max(\boldsymbol{q}) - \min(\boldsymbol{q})} + (1-\lambda)\frac{r_i - \min(\boldsymbol{r})}{\max(\boldsymbol{r}) - \min(\boldsymbol{r})} \tag{2.20}$$

It is necessary to perform 2.20, given that nodes close to a source, as well as nodes positioned at extremities would receive small mean intra-cluster cosine similarity values. This requires additional cross-cluster information in order to be able to distinguish them, hence the combination of both intra- and cross-cluster similarities in 2.20. As a result, only nodes close to a cluster source will express small p_i values. Subsequent to the aforementioned developments, the node with the smallest p_i is selected as a reference node. The resulting membership value μ_i of a node is the cosine similarity between this node and the reference node

$$\mu_i = a_{i,\arg\min(p_j)}, \forall i, j \text{ and } c_x \in \{c_1, c_2\} \tag{2.21}$$

These membership values are collected in the vector $\boldsymbol{\mu}$, on which min-max normalization is applied once more. An example scenario of a single simulation is provided in figure 2.3.

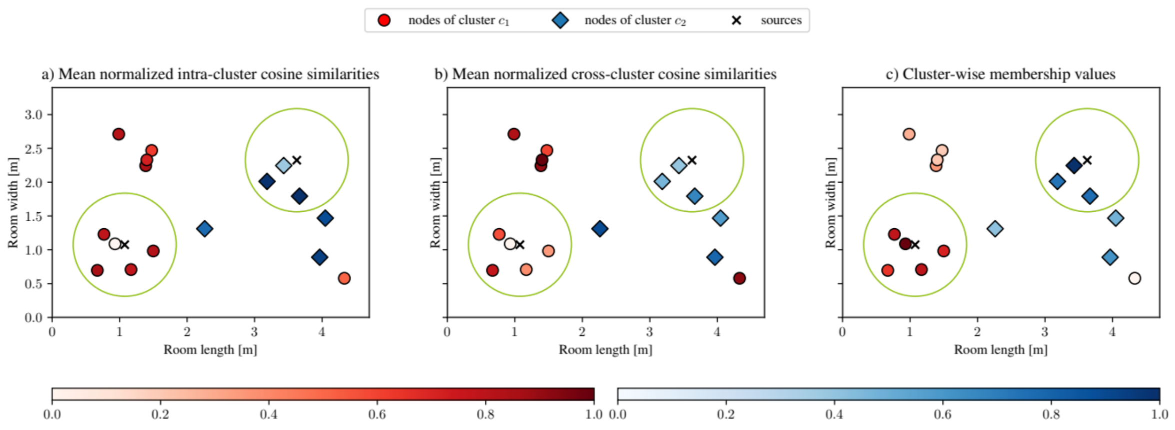


Figure 2.3: Example cluster membership values for unsupervised clustered federated learning.

2.3.3 Coherence-based clustering

A relatively novel and slightly different approach to clustering in ad-hoc microphone arrays proposes a method based on the magnitude-squared-coherence between microphones' observations, which measures their degree of linear dependency by analyzing similar frequency components [12] .

Subsequently, a non-negative matrix (NMF) based approach is utilized, with the goal of obtaining an optimal clustering, whereby nodes are assigned into subnetworks based on their respective microphone observations.

The suggested method offers the capability to dynamically perform clustering while imposing a low computational burden, rendering it highly applicable to various audio signal processing applications. Consequently providing a notable advantage in terms of processing efficiency, making the method an attractive option for real-world scenarios where computational resources may be limited or where rapid processing is required.

2.3.3.1 Signal model

To include interference in the signal model, the observed signal $x_d(t)$ at microphone d can be represented like so:

$$x_d(t) = s_d(t) + v_d(t) \tag{2.22}$$

Where $v_d(t)$ denotes the noise signal plus interference at time instant t. The linear signal model in equation 2.22 can be conveniently restated to denote the collection of a frame of samples into a vector form:

$$\begin{aligned} \mathbf{x}_d(t) &= [x_d(t)x_d(t-1)\cdots x_d(t-T+1)]^T \\ &= \mathbf{s}_d(t) + \mathbf{v}_d(t) \end{aligned} \tag{2.23}$$

- T: frame size
- $\_^T$: matrix transpose
- $\mathbf{x}_d(t)$: observed signal vector
- $\mathbf{s}_d(t)$: clean speech vector
- $\mathbf{v}_d(t)$: noise signal vector

2.3.3.2 Clustering algorithm

Magnitude-squared coherence

By utilizing the magnitude squared coherence, it is possible to conduct an analysis of the linear relationship between two signals $x(t)$ and $y(t)$. First, the Fast Fourier Transform (FFT) of both signals is computed. After which the coherence is measured as a function of the center frequency of the filter. The magnitude-squared coherence can be obtained with the following formula [19]:

$$\Gamma_{xy}(f) = \frac{|S_{xy}(f)|^2}{S_{xx}(f)S_{yy}(f)} \tag{2.24}$$

- f: The center frequency of the filter
- S_{xx}: The auto spectral density of x
- S_{yy} The auto spectral density of y
- S_{xy} The cross-spectral density

The power spectra $S_{xx}(f)$ and $S_{yy}(f)$ describe the distribution of power into frequency components composing the signals $x(t)$ and $y(t)$. [20] To compute the cross-spectral density, the following equation can be used:

$$S_{xy}(f) = \sum_{k=1-T}^{T-1} R_{xy}(k)e^{-i2\pi fk} \tag{2.25}$$

- $R_{xy}(k)$: The cross-correlation between $x(t)$ and $y(t)$
- T: The frame size

For the special case $x(t) = y(t)$, equation 2.25 reduces to $S_{xx}(f)$, $R_{xy}(k)$ can be estimated with:

$$R_{xy}(k) = \begin{cases} \frac{1}{T}\sum_0^{T-1-k} x(t)y(t+k) & k = 0, \dots, T-1 \\ R_{xy}(-k) & k = -(T-1), \dots, -1 \end{cases} \tag{2.26}$$

Now, sufficient information is provided to be able to compute $\Gamma_{xy}(f)$. This calculation yields a value between 0 and 1, with higher values denoting a stronger linear correlation. By calculating

$$C_{xy} = \frac{\sum_{f=0}^{F} \Gamma_{xy}(f)}{F} \in [0,1] \tag{2.27}$$

all frequency bins are assigned the same weight regardless of their power. By arranging all coherence measures C_{xy} between the audio signals, a non-negative coherence matrix **C** can be obtained.

$$\boldsymbol{C} = \begin{bmatrix} 1 & \cdots & \cdots & C_{1M} \\ C_{12} & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ C_{1M} & C_{2M} & \cdots & 1 \end{bmatrix} \in \mathbb{R}_+^{M\times M} \tag{2.28}$$

An example arrangement of coherence measures C_{xy} in matrix $\boldsymbol{C}$ is displayed in figure 2.4. In this case, 3 clusters and 10 microphones are used.

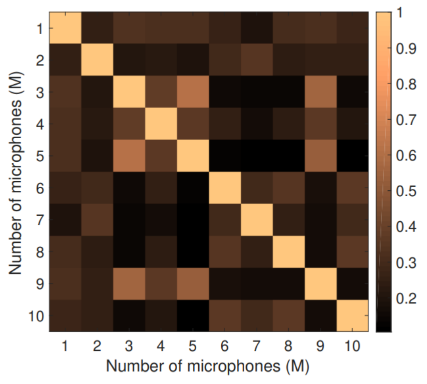


Figure 2.4: An example coherence matrix.

2.3.3.3 Non-negative matrix factorization

The matrix $\boldsymbol{C}$ contains values that represent the degree of correlation between the signals observed by each combination of microphones, meaning that each row (or column) j of $\boldsymbol{C}$ represents the degree of correlation between the j-th microphone signal and all other signals. As a result, groups of microphones close to a specific source will be highly correlated.

The next step consists of exploiting the inherent clustering property of NMF [21]. $\boldsymbol{C}$ can be considered as a linear subspace of dimension M. By downgrading this subspace into a linear subspace with dimension corresponding to the amount of sources K, a clustering can be achieved. The matrix $\boldsymbol{C}$ is non-negative and can be modelled as:

$$\boldsymbol{C} = \boldsymbol{B}\boldsymbol{B}^T \odot (\boldsymbol{1} - \boldsymbol{I}) + \boldsymbol{I} \tag{2.29}$$

- $\boldsymbol{B} \in \mathbb{R}^{M \times K}$: The cluster matrix, where K is the amount of speakers (the amount of clusters)
- $\odot$: Element-wise product
- $\boldsymbol{I}$: The identity matrix
- $\boldsymbol{1}$: The all-ones matrix

The latter two are introduced because the main diagonal of $\boldsymbol{C}$ does not provide any relevant information in the learning process of $\boldsymbol{B}$. Because $\boldsymbol{C}$ is symmetric, we model it as $\boldsymbol{B}\boldsymbol{B}^T$. It is possible to estimate $\boldsymbol{B}$ using iterative multiplicative update rules based on Euclidian divergence [22]:

$$\boldsymbol{B} \leftarrow \boldsymbol{B} \odot \frac{(\boldsymbol{C} \odot (\boldsymbol{1} - \boldsymbol{I}))\boldsymbol{B}}{(\boldsymbol{B}\boldsymbol{B}^{\boldsymbol{T}} \odot (\boldsymbol{1} - \boldsymbol{I}))\boldsymbol{B}} \tag{2.30}$$

Now each column of $\boldsymbol{B}$ contains the contribution of a microphone to each cluster. We can obtain the clustering result with:

$$\gamma_m = \{j \in [1, K] : B_{mj} \geq B_{mk}, \forall k \in [1, K]\} \tag{2.31}$$

The value γ_m denotes the cluster assigned to the m-th microphone. This is simply the largest value of column m. An example cluster matrix for 10 microphones and 3 clusters is illustrated in figure 2.5.

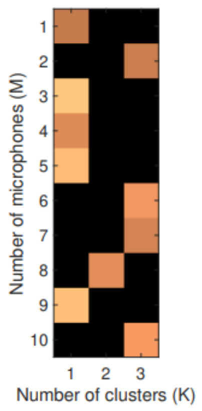


Figure 2.5: An example cluster matrix.

Algorithm 2 displays a suggested clustering method using the aforementioned techniques [12].

Algorithm 2 Clustering using coherence and NMF

```
for $i = 1$ to $M$ do
    for $j = 1$ to $M$ do
        Compute the cross correlation $R_{x_i,x_j}(k)$ using 2.26
        Compute the CSD $S_{x_i,x_j(f)}$ using 2.25
        Obtain the coherence measure $\Gamma_{x_i,x_j}(f)$ using 2.24
        Compute the coherence metric $C_{x_i,x_j}$ using 2.27
    end for
end for
Initialize $\boldsymbol{B}$ with random values.
for iters $= 1$ to maxIter do
    Update $\boldsymbol{B}$ according to 2.30
end for
Obtain the optimal clustering by 2.31
```

2.3.4 Time-domain coherence-based clustering

//TODO Another way to determine microphone clusters could be to calculate coherence in the time-domain representation of the signals. Firstly, two signals x and y are considered, the latter of which is delayed by $\Delta\tau$ samples.

$$\begin{aligned} x(n) &= s(n-\tau) \\ y(n) &= s(n-\tau-\Delta\tau) \end{aligned} \tag{2.32}$$

A filter h_1 is iteratively estimated so that $x * h_1 \approx y$. This is done by evaluating the derivative of the cost function J:

$$J = (y - h * x)^2 \tag{2.33}$$

$$\begin{aligned} \frac{\partial J}{\partial h} &= 2(x - h * y)(-y) \\ &= -2y(x - h * y) \end{aligned} \tag{2.34}$$

This results in the following update rule for h_1:

$$h_1(n+1) = h_1(n) - \alpha \frac{\partial J(h)}{\partial h_1}\Big|_{h=h(n)} \tag{2.35}$$

After which h_1 can be Fourier transformed to result in:

$$h_1 = \frac{R_{xy}}{R_{xx}} \xrightarrow{\mathfrak{F}} \frac{S_{xy}}{S_{xx}} \tag{2.36}$$

Analogous to the previous, a filter h_2 can be estimated so that $y * h_2 \approx x$.

$$h_2 = \frac{R_{yx}}{R_{yy}} \xrightarrow{\mathfrak{F}} \frac{S_{yx}}{S_{yy}} \tag{2.37}$$

The fourier transform of the convolution of h_1 and h_2 results in an estimated coherence measure.

$$h_1 * h_2 \xrightarrow{\mathfrak{F}} \frac{|S_{xy}|^2}{S_{xx}S_{yy}}(f) \tag{2.38}$$

Figures 2.7 and 2.8 show example estimations for h_1, h_2 and their convolutions for microphones with different relationships. The acoustic scenario considered is illustrated in Figure 2.6.

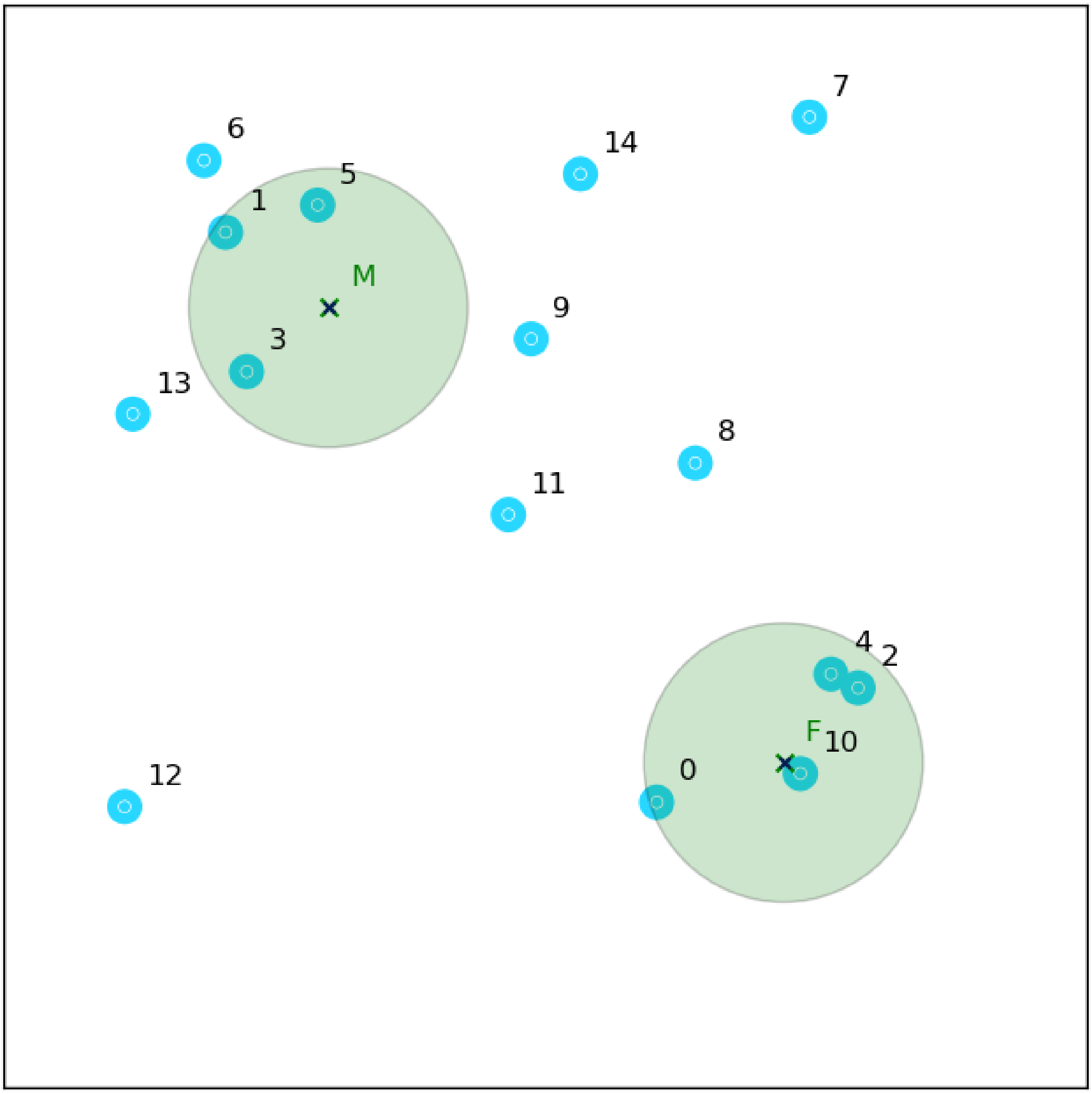
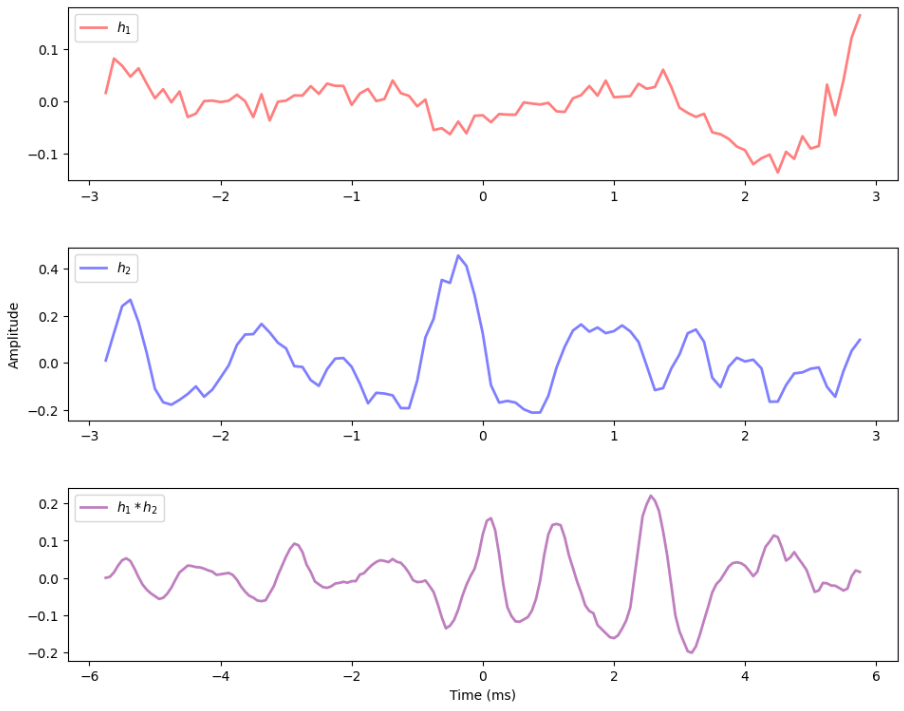


Figure 2.6: Example acoustic scenario.

(a) Microphones 1 and 4, which should definitely end up in different clusters.



(b) Microphones 2 and 10.

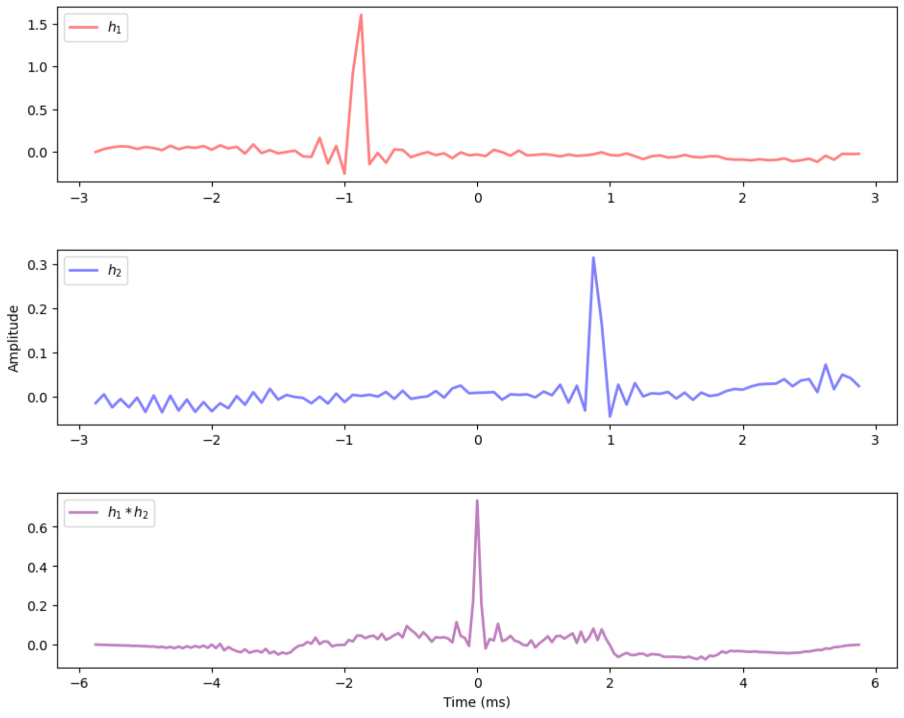
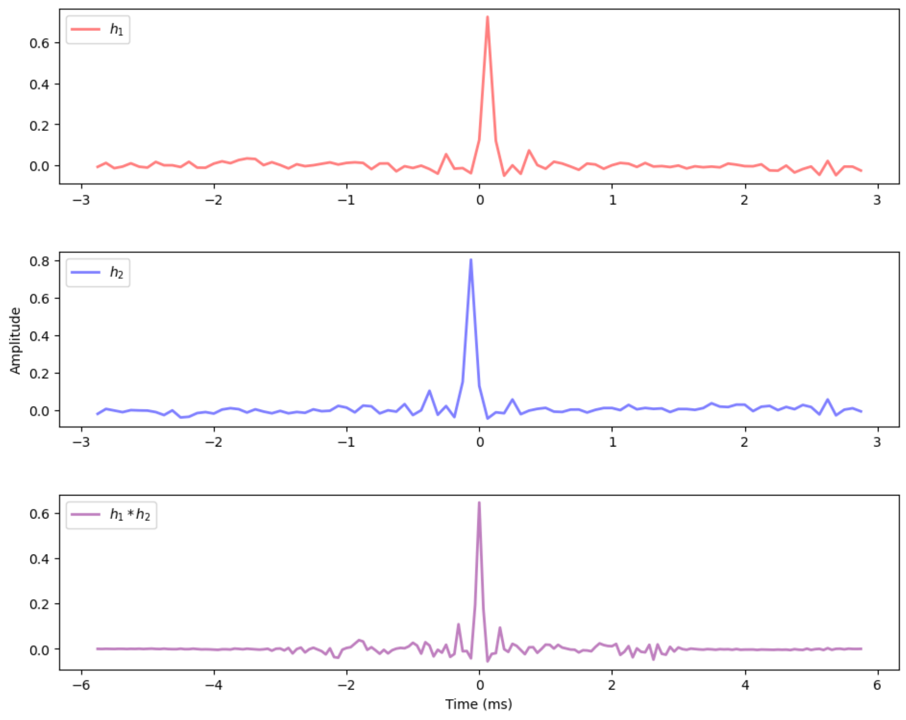


Figure 2.7: Example h_1 and h_2 filters and the result of their convolutions.

(a) Microphones 2 and 4.



(b) Microphone 0, compared to itself.

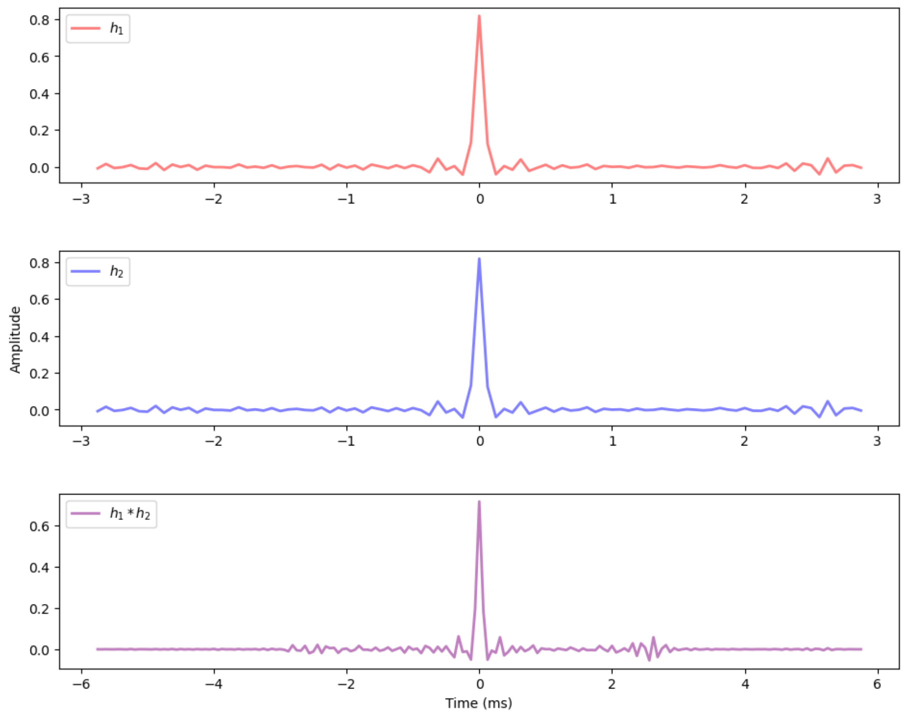


Figure 2.8: Example h_1 and h_2 filters and the result of their convolutions.

2.4 Source separation

Imagine you are attending a crowded party, and there are several conversations happening at once. The room is filled with noise from music, chatter, and laughter. You are standing in a group of people, trying to listen to the person speaking to you, but it's challenging to focus on their voice amidst the surrounding noise. This situation illustrates the cocktail party problem [23]. The sound waves from all the conversations, including the music and the ambient noise, are mixed together and received by your ears as a single audio signal. Your brain must then separate and decode the individual sound sources to understand what is being said to you. This is the same challenge that signal processing algorithms face when trying to separate sound signals from multiple sources in a recording. As a sidenote, the cocktail party effect is even observed in penguin colonies. Chicks are able to extract and perceive the calls of their parents from far away, even in busy feeding areas where the perceived signals are dominated by noise and interferers [24].

Source separation aims to provide a solution to this problem by obtaining a signal containing only the target source, effectively suppressing all other sources except the target. The aforementioned principle is recognized as adaptive 'nulling'.[25] In this part, an emphasis will be put on source separation in ad hoc setups, so any techniques that require positional information on microphone nodes will not be discussed.

Before being able to perform source separation, many techniques require a clustering algorithm to determine clusters of source-dominated microphones. Some of these techniques are thoroughly discussed in section 2.3.

2.4.1 Source separation using fuzzy-membership values

One approach to solving the cocktail party problem is by utilizing fuzzy clustering techniques in combination with delay-and-sum beamforming (DSB). By expanding upon the method described in section 2.3.1, this technique aims to improve the performance of DSB by incorporating fuzzy-membership values. These values are used to construct a weighted combination of microphone signals.

Prior to applying DSB, TDOAs are estimated using cross-correlation analysis. These TDOAs are then used in the beamforming stage, after which a post-filtering mask is estimated and applied, resulting in a final, enhanced estimate of the source signal in each cluster.

The fuzzy clustering method improves upon traditional beamforming techniques by allowing for more accurate estimation of the desired signals in the presence of noise and interference. By incorporating fuzzy-membership values, the technique is able to better distinguish between different audio sources, leading to more effective source separation.

2.4.1.1 Fuzzy-membership value aware signal enhancement

Initial source signal estimation

It is possible to perform beamforming using the microphones of a source cluster if the relative delays between the microphones are known for that source. Since the locations of the microphones are unknown, the following needs to be done for each cluster n. An initial estimate of the source signal ($\hat{s}_{i_n}(l)$) at all microphones $d = i_n$ assigned to that cluster is obtained. Subsequently, a reference microphone is selected for each cluster. By performing correlation analysis of all other microphone signals with respect to the reference microphone, time-differences-of-arrival (TDOAs) can be estimated. These TDOAs can afterwards be used in the beamforming stage.

By presuming that different sources are approximately disjoint in the time-frequency (T-F) plane, only one source may be assumed to be dominant at any one T-F point (k, b). This allows for the estimation of a spectral mask $\mathscr{M}_n(k, b)$ for each cluster. Applying this mask to the microphone signals of that cluster will provide an estimate of the underlying source signal with a reduced amount of interference by other sources. The simplest way to represent such a mask would be like so [26]:

$$\mathscr{M}_n(k,b) = \begin{cases} 1 & \text{if source } n \text{ is dominant at } (k,b) \\ 0 & \text{otherwise} \end{cases} \tag{2.39}$$

To estimate this mask, the microphone $d = R_n$ with the highest FMV is selected as reference microphone. By computing the short-time Fourier transform (STFT) representation $X_{R_n}(k, b)$ from the signal of this microphone, the binary mask for cluster n can be acquired by the following:

$$\mathscr{M}_n(k,b) = \begin{cases} 1 & |X_{R_n}(k-b)| > \frac{1}{B}\sum |X_{R_j}(k,b)|, \\ & \quad j = 1,\dots,N \text{ and } j \neq n \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

The parameter B is used to average the spectral amplitudes across time in order to reduce the effect of jitter induced by the large inter-microphone distances in ad-hoc arrays. The masks $\mathscr{M}_n(k, b)$ are applied to the spectra $X_{i_n}(k, b)$ of all microphones i_n assigned to cluster n.

$$\tilde{X}_{i_n}(k,b) = X_{i_n}(k,b)\mathscr{M}_n(k,b) \tag{2.41}$$

Afterwards, the inverse STFT of $\tilde{X}_{i_n}(k, b)$ is computed in order to reconstruct the time-domain signal $\hat{s}_{i_n}$ representing the initial estimate of the source signal. The estimate of the reference microphone is used for the correlation analysis, which yields TDOAs for all microphones of each cluster with respect to the reference microphone.

Clustering-steered beamforming

A generalized DSB can be formed in the time-domain using the relative TDOAs for a cluster.

$$\hat{s}_{n,\text{W-DSB}}(l) = \sum_{i_n} w_{n,i_n} x_{i_n}(l + D_{i_n}) \tag{2.42}$$

- l: The discrete time index
- D_{i_n}: The relative TDOA's
- w_{n,i_n}: The weights allocated to each microphone i_n of cluster n

In [26], all weights were set uniformly, but in [11] the weights are set proportional to the FMV. The latter approach yields better results, since the weighting makes it so that signals with a higher FMV are considered of higher importance, implying a higher signal-to-noise ratio (SNR) in said signals. By selecting the first I_n microphones with the highest FMV per cluster, the uncertainty introduced by microphones with a low FMV is reduced.

Mask re-estimation

After performing the previous, a post-filtering mask is computed in a similar manner to 2.40.

$$\mathscr{M}_{n,\text{DSB}}(k,b) = \begin{cases} 1 & |\hat{S}_{n,\text{FMVA-DSB}}(k,b)| > \frac{1}{B} \sum |\hat{S}_{j,\text{FMVA-DSB}}(k,b)| \\ & \quad j = 1, \dots, N \text{ and } j \neq n \\ 0 & \text{otherwise} \end{cases} \tag{2.43}$$

This mask is applied to $\hat{S}_{n,\text{FMVA-DSB}}(k,b)$, after which the time-domain signal is reconstructed. This results in a final, enhanced estimate of the source signal in each cluster. Figure 2.9 depicts a high-level view of the algorithm in the case of two sources.

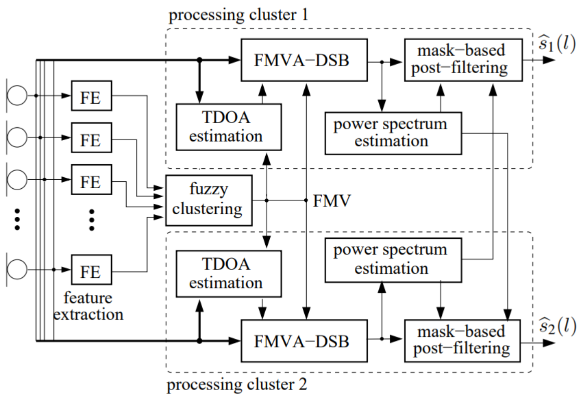


Figure 2.9: Schematic representation of the algorithm for 2 clusters.

3

Methods

3.1 Frameworks

3.1.1 Pyroomacoustics

Pyroomacoustics is an open-source software package designed to streamline the development and evaluation of audio array processing algorithms. It provides a set of powerful tools for simulating and analyzing acoustic environments, including the generation of room impulse responses, the simulation of sound propagation, and the generation of synthetic audio signals. With its user-friendly interface and intuitive programming API, Pyroomacoustics is an accessible and versatile tool for researchers, students, and practitioners working in the field of audio processing. Its modular design and extensive documentation make it easy to extend and adapt to a wide range of research applications, from speech enhancement and source localization to sound event detection and acoustic scene analysis. Overall, Pyroomacoustics represents a valuable resource for anyone interested in exploring the fascinating and rapidly-evolving field of audio processing.

In the context of this thesis, Pyroomacoustics is used for the implementation, evaluation and comparison of several clustering methods.

3.1.2 SINS dataset

SINS is a collection of audio recordings that were captured in a real-life setting of a vacation home, where one individual lived for a duration of over a week. The audio was captured using a network of 13 microphone arrays that were strategically placed across multiple rooms. Each microphone array was composed of four microphones that were arranged linearly. The recordings were labeled according to the different levels of daily activities that were performed in the environment. An illustration of the recording setup is provided in figure 3.1.

The realistic room impulse responses (RIRs) from the SINS database provide means of evaluation more similar to real world scenarios, resulting in more challenging problems to be solved by the evaluated algorithms.

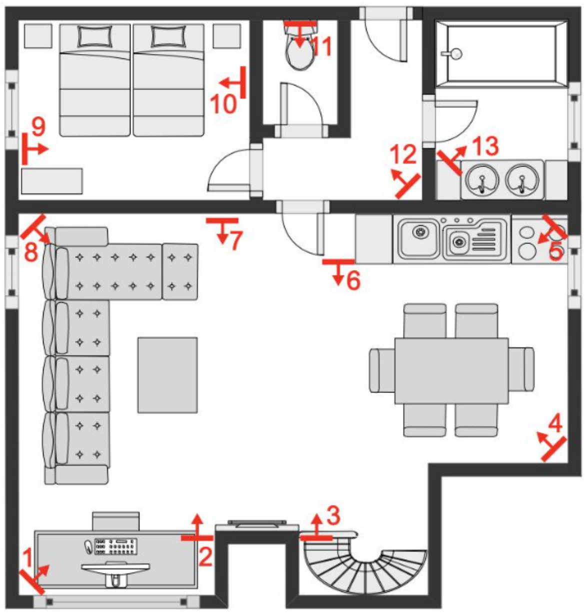


Figure 3.1: Setup of the vacation home.

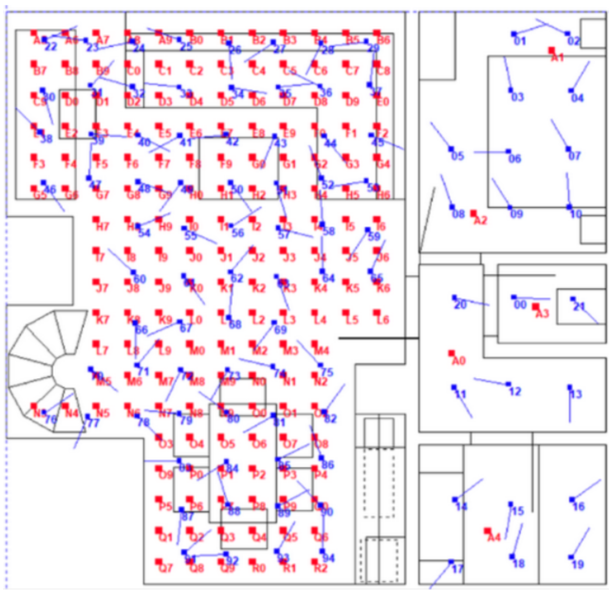


Figure 3.2: Different options of the SINS dataset.

3.2 Implementation of coherence based clustering in python

3.2.1 Coherence matrix calculation

The implementation follows the method discussed in section 2.3.3. Instead of calculating the coherence between two signals manually, which takes a very long time, the *signal.coherence()* function from Scipy is used.

```
    def calculate_coherence_matrix(self):
        """
        Calculates the coherence matrix based on the Fourier representation of
↪   the signals.
        """
        num_channels = self.Signals.shape[0]
        num_bins = self.Signals.shape[1]
        num_frames = self.Signals.shape[2]

        coherence_matrix = np.zeros((num_channels, num_channels))
        self.coherence_matrix = coherence_matrix

        for i in range(num_channels):
            for j in range(num_channels):

                co = signal.coherence(self.signals[i], self.signals[j],
                ↪  fs=self.fs, nfft=self.nfft)
                c = np.sum(co[1])/num_bins
                coherence_matrix[i,j] = c
        return self.coherence_matrix
```

Code Fragment 3.1: Calculation of coherence matrix in python

3.2.2 Non-negative matrix factorization

In the algorithm described in section 2.3.3, the matrix $\boldsymbol{B}$ is initialized with random values, but the performance of NMF can be optimized by generating good initial values. This can be achieved by SVD-based initialization [27]. An implementation of this method is displayed in code fragment 3.4. //TODO more about SVD

```python
def nmf(self, C,k,max_iter,init_mode='random'):

    if init_mode == 'random':
        B ,_ = self.random_initialization(C,k)
    elif init_mode == 'nndsvd':
        B ,_ = self.nndsvd_initialization(C,k)

    I = np.identity(C.shape[0])
    one = np.ones(C.shape[0])
    for n in range(max_iter):
        # Update B
        top = (np.multiply(C, one - I)) @ B
        bottom =  (np.multiply(B @ np.transpose(B), (one - I) ) ) @ B
        B = np.multiply(B, np.divide(top, bottom))

    return B
```

Code Fragment 3.2: Calculation of non-negative matrix factorization in python

```python
def random_initialization(self, A,rank):
    number_of_documents = A.shape[0]
    number_of_terms = A.shape[1]
    W = np.random.uniform(1,2,(number_of_documents,rank))
    H = np.random.uniform(1,2,(rank,number_of_terms))
    return W,H
```

Code Fragment 3.3: Random initialization of matrix B

```python
def nndsvd_initialization(self, A,rank):
    u,s,v=np.linalg.svd(A,full_matrices=False)
    v=v.T
    w=np.zeros((A.shape[0],rank))
    h=np.zeros((rank,A.shape[1]))

    w[:,0]=np.sqrt(s[0])*np.abs(u[:,0])
    h[0,:]=np.sqrt(s[0])*np.abs(v[:,0].T)

    for i in range(1,rank):

        ui=u[:,i]
        vi=v[:,i]
        ui_pos=(ui>=0)*ui
        ui_neg=(ui<0)*-ui
        vi_pos=(vi>=0)*vi
        vi_neg=(vi<0)*-vi

        ui_pos_norm=np.linalg.norm(ui_pos,2)
        ui_neg_norm=np.linalg.norm(ui_neg,2)
        vi_pos_norm=np.linalg.norm(vi_pos,2)
        vi_neg_norm=np.linalg.norm(vi_neg,2)

        norm_pos=ui_pos_norm*vi_pos_norm
        norm_neg=ui_neg_norm*vi_neg_norm

        if norm_pos>=norm_neg:
            w[:,i]=np.sqrt(s[i]*norm_pos)/ui_pos_norm*ui_pos
            h[i,:]=np.sqrt(s[i]*norm_pos)/vi_pos_norm*vi_pos.T
        else:
            w[:,i]=np.sqrt(s[i]*norm_neg)/ui_neg_norm*ui_neg
            h[i,:]=np.sqrt(s[i]*norm_neg)/vi_neg_norm*vi_neg.T

    return w,h
```

Code Fragment 3.4: NNDSVD initialization of matrix B

4 Results

4.1 Evaluation Metrics

Defining a good performance metric for a clustering algorithm can be challenging because of several reasons. First, in many cases, the true labels or classes for the data are not available, making it difficult to measure the algorithm's performance objectively. Second, clustering is an unsupervised learning method, which means that there is no single correct way to group the data, and the clustering result can depend on the algorithm's parameters and assumptions. Third, different performance metrics may capture different aspects of clustering quality, and there can be trade-offs between them, such as favoring compactness and separation versus semantic coherence. These factors make it hard to define a performance metric that can capture all aspects of clustering quality and be applicable to different datasets and algorithms.

As an example, relying on ground truths that are solely based on oracle knowledge of the Room Impulse Responses (RIRs) [12] or distances [28, 29] does not provide a complete representation of the signal mixing in the ad hoc scenario. In order to address this issue effectively, there is a need to explore novel approaches for assessing cluster quality. The forthcoming paragraph will delve deeper into the techniques described in [10].

4.1.1 Cluster Metrics

One way to evaluate cluster performance is by analyzing the distribution of the direct-to-reverberant ratio (DRR) and the distribution of the direct-to-reverberant, interference, and noise ratio (DRINR) [10]. These metrics are computed for source-dominated microphone clusters, and are computed as follows for a microphone signal y_m and source cluster c:

$$\text{DRINR} = \frac{\sum_n (x_{c,m}^{\text{dir}}(n))^2}{\sum_n (y_m(n) - x_{c,m}^{\text{dir}}(n))^2} \tag{4.1}$$

$$\text{DRR} = \frac{\sum_n (x_{c,m}^{\text{dir}}(n))^2}{\sum_n (x_{c,m}^{\text{rev}}(n))^2} \tag{4.2}$$

4.1.2 Source Separation Metrics

4.1.2.1 Source-to-interference ratio

The source-to-interference ratio (SIR) is usually interpreted as the amount of other sources that can be heard in a source estimate, similar to the concept "bleed" or "leakage".

$$\text{SIR} := 10 \log_{10} \frac{||s_{\text{target}}||^2}{||e_{\text{interf}}||^2} \tag{4.3}$$

4.1.2.2 Perceptual evaluation of speech quality

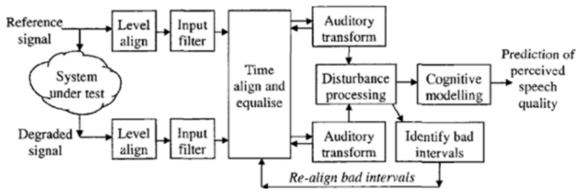


Figure 4.1: Structure of PESQ model.

4.1.2.3 Short-time objective intelligibility

//TODO

Proposed by [30], the short-time objective intelligibility (STOI) measure provides a way to evaluate the quality of noisy speech processed by noise reduction or speech separation algorithms.

$$X_j(m) = \sqrt{\sum_{k=k_1(j)}^{k_2(j)-1} |\hat{x}(k,m)|^2} \tag{4.4}$$

$$SDR_j(n) = 10 \ \log_{10} \left(\frac{X_j(n)^2}{(\alpha Y_j(n) - X_j(n))^2} \right). \tag{4.5}$$

$$Y' = \max(\min(\alpha Y, X + 10^{-\beta/20} X), X - 10^{-\beta/20} X) \tag{4.6}$$

$$d_j(m) = \frac{\sum\limits_{\text{n}} \left(X_j(n) - \frac{1}{N} \sum\limits_l X_j(l) \right) \left(Y_j'(n) - \frac{1}{N} \sum\limits_l Y_j'(l) \right)}{\sqrt{\sum\limits_n \left(X_j(n) - \frac{1}{N} \sum\limits_l X_j(l) \right)^2 \sum\limits_n \left(Y_j'(n) - \frac{1}{N} \sum\limits_l Y_j'(l) \right)^2}} \tag{4.7}$$

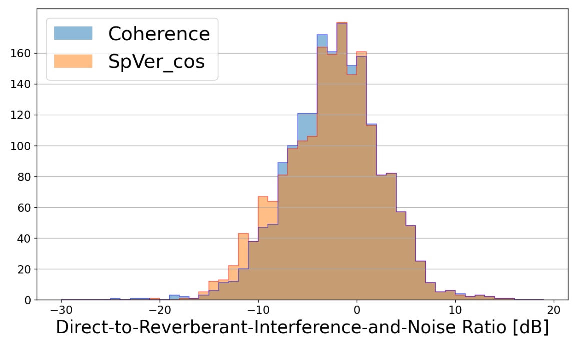
$$d = \frac{1}{JM} \sum_{j,m} d_j(m) \tag{4.8}$$

4.2 Evaluation using pyroomacoustics

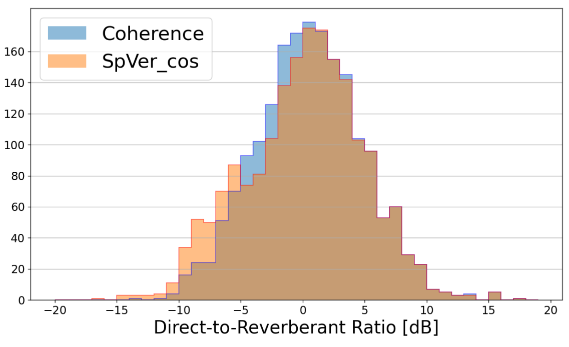
4.2.1 Frequency domain Coherence-based clustering

//TODO

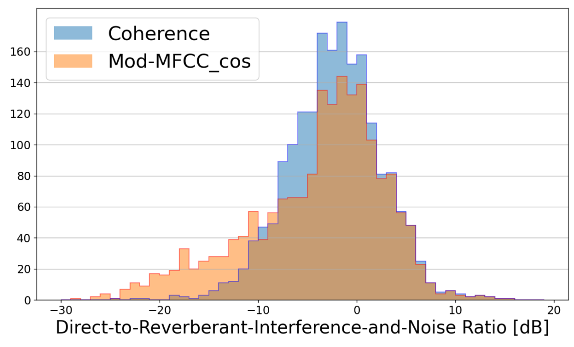
4.2.2 Time domain Coherence-based clustering



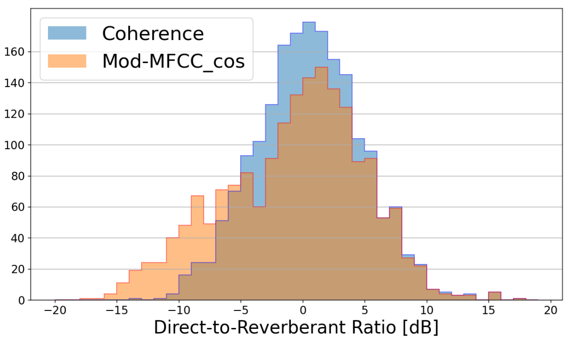
(a)



(b)



(c)



(d)

Figure 4.2: Comparing speaker verification and mod-MFCC with the coherence-based method

4.3 Evaluation using the SINS database

4.3.1 Comparison between MFCC, SpVer and Coherence

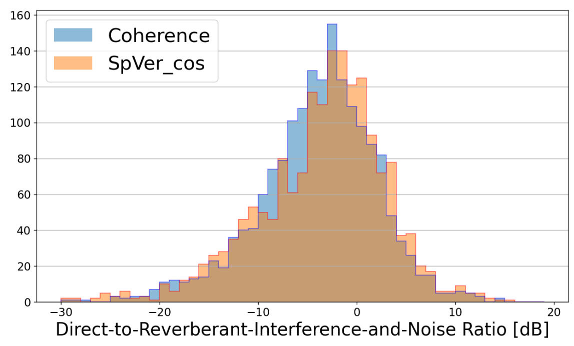
4.3.1.1 Cluster metrics

The metrics from section 4.1.1 are used to compare the performance of the coherence-based clustering algorithm with those of speaker verification and mod-MFCC. For the latter two, cosine similarity is used as a distance metric, as it has been shown to yield the best results [7].

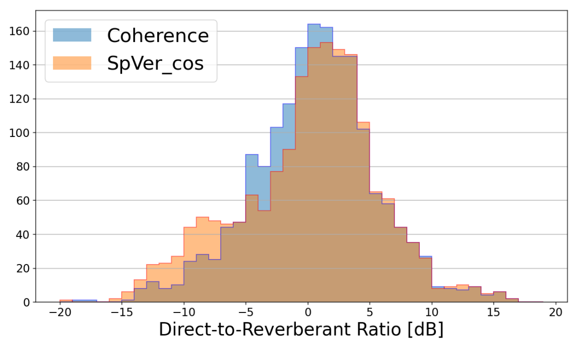
Far distances

At first glace, the DRINR and DRR values indicate good clustering results when using the coherence-based method. In this scenario, coherence-based clustering even outperforms speaker verification.

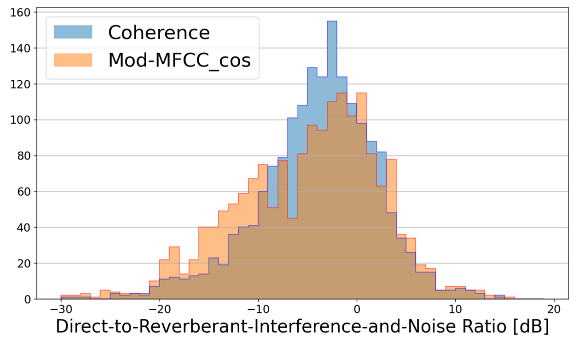
Close distances



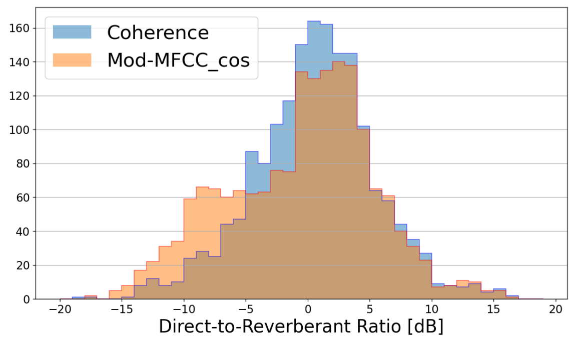
(a)



(b)



(c)



(d)

Figure 4.3: Comparing speaker verification and mod-MFCC with the coherence-based method for close distances

4.3.1.2 Source separation metrics

//TODO

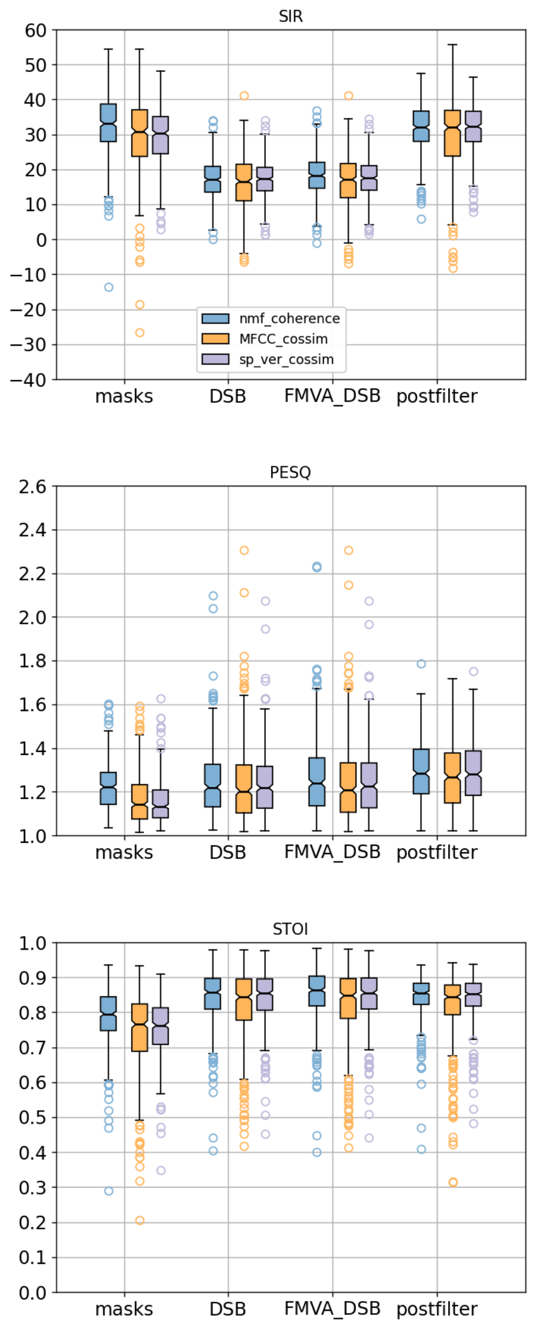


Figure 4.4: SIR, PESQ and STOI metrics.

Conclusion

//TODO

References

[1] Desplanques, Brecht and Thienpondt, Jenthe and Demuynck, Kris, “ECAPA-TDNN : Emphasized Channel Attention, Propagation and Aggregation in TDNN based speaker verification,” in *INTERSPEECH 2020*. International Speech Communication Association (ISCA), 2020, pp. 3830–3834. [Online]. Available: http://dx.doi.org/10.21437/Interspeech.2020-2650

[2] R. M. Sebastian Gergen, Anil Nagathil, “Classification of reverberant audio signals using clustered ad hoc distributed microphones,” pp. 1–12, 2014.

[3] A. Oppenheim and R. Schafer, “From frequency to quefrency: a history of the cepstrum,” *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 95–106, 2004.

[4] F. Zheng, G. Zhang, and Z. Song, “Comparison of different implementations of MFCC,” *Journal of Computer Science and Technology*, vol. 16, no. 6, pp. 582–589, nov 2001. [Online]. Available: https://link.springer.com/article/10.1007/BF02943243

[5] S. S. Stevens, J. Volkmann, and E. B. Newman, “A scale for the measurement of the psychological magnitude pitch,” *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937. [Online]. Available: https://doi.org/10.1121/1.1915893

[6] S. Furui, *Digital speech processing: synthesis, and recognition*. CRC Press, 2018.

[7] L. B. S. Kindt, J. Thienpondt and N. Madhu, “Ad-hoc microphone clustering with speaker embeddings under realistic and challenging scenarios.” *2023 IEEE International Conference*, pp. 1–18, 2023.

[8] D. Snyder, D. Garcia-Romero, G. Sell, D. Povey, and S. Khudanpur, “X-vectors: Robust dnn embeddings for speaker recognition,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 5329–5333.

[9] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[10] J. T. S. Kindt and N. Madhu, “Exploiting speaker embeddings for improved microphone clustering and speech separation in ad-hoc microphone arrays,” *2023 IEEE International Conference*, pp. on Acoustics, Speech and Signal Processing, 2023, (ACCEPTED BUT NOT PUBLISHED).

[11] N. M. Sebastian Gergen, Rainer Martin, “Source separation by fuzzy-membership value aware beamforming and masking in ad hoc arrays,” pp. 1–5, 2018.

[12] M. G. C. Antonio J. Munoz-Montoro, Pedro Vera-Candeas, “A Coherence-based Clustering Method for Multichannel Speech Enhancement in Wireless Acoustic Sensor Networks,” pp. 1–5, 2018.

[13] R. M. Alexandru Nelus, Rene Glitza, “ESTIMATION OF MICROPHONE CLUSTERS IN ACOUSTIC SENSOR NETWORKS USING UNSUPERVISED FEDERATED LEARNING,” pp. 1–5, 2021.

[14] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, “Federated optimization: Distributed machine learning for on-device intelligence,” 2016.

[15] F. Sattler, K.-R. Müller, and W. Samek, “Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3710–3722, 2021.

[16] F. Sattler, K.-R. Müller, T. Wiegand, and W. Samek, “On the byzantine robustness of clustered federated learning,” in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8861–8865.

[17] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, “Robust and communication-efficient federated learning from non-i.i.d. data,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[18] A. Nelus, J. Ebbers, R. Haeb-Umbach, and R. Martin, “Privacy-preserving variational information feature extraction for domestic activity monitoring versus speaker identification,” in *INTERSPEECH 2019, Graz, Austria*, 2019.

[19] W. A. Gardner, “A unifying view of coherence in signal processing,” *Signal Processing*, vol. 29, no. 2, pp. 113–140, 1992. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0165168492900150

[20] P. Stoica and R. Moses, *Spectral analysis of signals*. Prentice Hall, 2004. [Online]. Available: https://user.it.uu.se/~ps/SAS-new.pdf

[21] H. D. S. Chris Ding, Xiaofeng He, “On the Equivalence of Nonnegative Matrix Factorization and Spectral Clustering,” p. 606–610, 2005.

[22] D. Lee and H. S. Seung, “Algorithms for non-negative matrix factorization,” in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., vol. 13. MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper/2000/file/f9d1152547c0bde01830b7e8bd60024c-Paper.pdf

[23] E. C. Cherry, “Some experiments on the recognition of speech, with one and with two ears,” *The Journal of the Acoustical Society of America*, vol. 25, no. 5, pp. 975–979, 1953. [Online]. Available: https://doi.org/10.1121/1.1907229

[24] T. Aubin and P. Jouventin, “Cocktail–party effect in king penguin colonies,” *Proceedings of the Royal Society of London. Series B: Biological Sciences*, vol. 265, no. 1406, pp. 1665–1673, 1998. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspb.1998.0486

[25] N. Madhu, “Acoustic source localization: Algorithms, applications and extensions to source separation,” Ph.D. dissertation, Ruhr-Universität Bochum, 2009.

[26] N. M. Sebastian Gergen, Rainer Martin, “SOURCE SEPARATION BY FEATURE-BASED CLUSTERING OF MICROPHONES IN AD HOC ARRAYS,” pp. 1–5, 2018.

[27] C. Boutsidis and E. Gallopoulos, “Svd based initialization: A head start for nonnegative matrix factorization,” *Pattern Recognition*, vol. 41, no. 4, pp. 1350–1362, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320307004359

[28] I. Himawan, I. McCowan, and S. Sridharan, “Clustered blind beamforming from ad-hoc microphone arrays,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 661–676, 2011.

[29] S. Pasha, Y. Zou, and C. Ritz, “Forming ad-hoc microphone arrays through clustering of acoustic room impulse responses,” 07 2015, pp. 84–88.

[30] C. H. Taal, R. C. Hendriks, R. Heusdens, and J. Jensen, “A short-time objective intelligibility measure for time-frequency weighted noisy speech,” in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010, pp. 4214–4217.

Appendices

Appendix A