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.
MFC	mel-Frequency Cepstrum. , 5
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:

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

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

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

$$x_d(t) = s_d(t) + v_d(t) \tag{2.3}$$

Where $v_d(t)$ denotes the noise signal plus interference at time instant t. The linear signal model in equation 2.3 can be conveniently restated to denote the collection of a frame of samples into a vector form:

$$\begin{aligned} \mathbf{x}_d(t) &= [x_d(t)x_d(t-1)\cdots x_d(t-T+1)]^T \\ &= \mathbf{s}_d(t) + \mathbf{v}_d(t) \end{aligned} \tag{2.4}$$

- 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

2.2 Sampling

Usually, microphone signals are sampled, resulting in a sampled signal $x_d(l)$. Transforming this signal to the short-time discrete Fourier domain results in $X_d(k, b)$

$$X_d(k,b) = \text{STFT}[x_d(l)] \tag{2.5}$$

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

2.3 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.

2.3.1 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]. These coefficients model the temporal and spectral modulations of the considered signal [3]. The resulting features offer a unified and flexible framework for speech/music/noise classification tasks, but can also be applied for clustering ad-hoc microphones.

2.3.1.1 Mel-frequency cepstrum and the mel scale

Computing the inverse Fourier transform of the logarithm of the amplitude spectrum of a signal results in a cepstrum [4]. 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 [5]. The MFC differs from the cepstrum due to the use of the Mel scale [6], 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:

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right) \tag{2.6}$$

2.3.1.2 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:

$$X(k,b) = \sum_{y=0}^{Y-1} x(bP+y)\mathcal{W}(y)e^{-j2\pi yk/Y} \tag{2.7}$$

- 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 2.3.1.1. This transformation is accomplished by using overlapping triangular windows $\mathcal{V}_{k'}(k)$ that function as band-pass filters [7]. This results in a mel-spectrum:

$$X_{\text{mel}}(k',b) = \sum_{k=0}^{Y/2} |X(k,b)|^2 \mathcal{V}_{k'}(k) \tag{2.8}$$

- 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.

$$X_{\text{mfcc}}(\eta,b) = \sum_{k'=0}^{K'-1} \log(|X_{\text{mel}}(k',b)|)\ \cos\left(\frac{\pi}{K'}(k'+0.5)\eta\right), \tag{2.9}$$

- η: The cepstral coefficient index

Exploiting the time evolution of short-time features is shown to improve classification results [3, 8]. To consider this, the short time MFCC modulation spectrum $\hat{X}_{\text{mfcc}}(\nu, \eta, c)$ is calculated using a sliding window DFT.

$$\hat{X}_{\text{mfcc}}(\nu, \eta, c) = \sum_{\ell=0}^{L-1} X_{\text{mfcc}}(\eta, cQ + \ell) e^{-j2\pi \ell \nu / L}, \tag{2.10}$$

- ν: The modulation frequency bin index, with $\nu = 0, 1, \cdots, L/2$
- c: The modulation window index, with $c = 0, 1, \cdots C_T - 1$
- Q: The modulation window shift

When commencing at the sub-frame index $b = cQ$, the sliding window takes into account L consecutive sub-frames [3]. When assuming that the averaged temporal behavior of a signal, in contrast to short-time audio features, is relatively stationary, a time averaging of aforementioned modulation spectra increases the robustness of the feature set against synchronization errors in the WASN. For this reason, the absolute values of the modulation spectra are averaged over all C_T frames and cepstral modulation ratios (CMR) $r_{\nu_1|\nu_2}$ are computed for the purpose of approximating the modulation spectrum.

$$\tilde{X}_{\text{mfcc}}(\nu, \eta) = \frac{1}{C_T} \sum_{c=0}^{C_T-1} |\hat{X}_{\text{mfcc}}(\nu, \eta, c)| \tag{2.11}$$

$$r_{\nu_1|\nu_2}(\eta) = \frac{\sum_{\nu=\nu_1}^{\nu_2} \tilde{X}_{\text{mfcc}}(\nu, \eta)}{(\nu_2 - \nu_1 + 1)\tilde{X}_{\text{mfcc}}(0, \eta)} \tag{2.12}$$

The normalization of the average of multiple modulation frequency bands falling within $\nu_1 \leq \nu \leq \nu_2$ is performed in equation 2.12, on the zeroth frequency band. If $\nu_1 = \nu_2 \neq 0$, this reduces to a single modulation frequency band $\nu_1 \geq 1$ that is normalized by the zeroth band.

Finally, the modulation spectrum is averaged over all modulation frequencies ν for each MFCC bin η.

$$\bar{X}_{\text{mfcc}}(\eta) = \frac{1}{L} \sum_{\nu'=0}^{L-1} \tilde{X}_{\text{mfcc}}(\nu', \eta) \tag{2.13}$$

Two CMRs and $\bar{X}_{\text{mfcc}}$ are stacked into a vector. This results in the final feature vector.

In most practical scenarios, audio signals are affected by room-specific reverberation and additional noise. This reverberation is determined by the path from the sound source to the receiver and can be represented using the room impulse response (RIR). One method to lessen the impact of reverberation in audio signal processing is Cepstral Mean Normalization (CMN). The convolutional distortion occurring in the time domain is equivalent to an additive term in the cepstral domain. By taking an average over a time period where the RIR is assumed to be constant and then subtracting this average from the cepstrum X_{mfcc}, the effect of reverberation can be reduced.[2, 9].

2.3.2 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 [10].

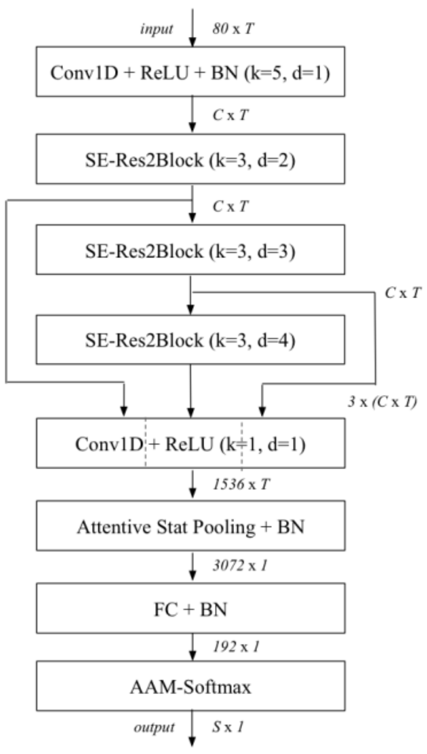
For clustering, the embeddings from each microphone are used as input features for the algorithm. These are generated from the Emphasized Channel Attention, Propagation and Aggregation Time Delay Neural Network (ECAPA-TDNN) [1], which improves upon the x-vector architecture [11] with several enhancements. 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 [12]. 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. Section 2.4.1 will provide further elaboration on this topic.

Chapter 5 provides an in-depth comparison of the methods based on mod-MFCC features from [2] and speaker embeddings from [10, 13] 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.

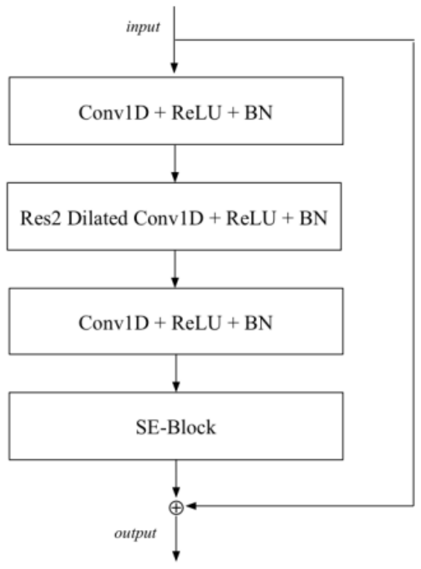


Figure 2.1: The network topology of the ECAPA-TDNN [1]

2.4 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 clusters are constructed, with each cluster corresponding to one specific sound source, but some methods add one additional background cluster for microphones dominated by background noise and interference. It is worth noting that not all of the subsequently described algorithms are restricted to one of the two approaches. 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 [14]. A variation on this method applies speaker embedding frameworks instead of hand-engineering embeddings from MFCC's[13], while a somewhat different approach utilizes a coherence based method and non-negative matrix factorization (NMF) for determining clusters [15]. 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[16].

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.

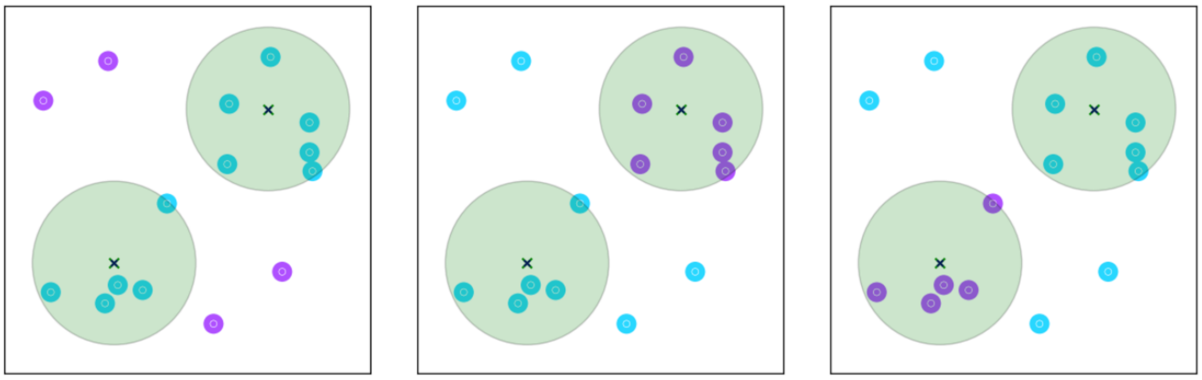


Figure 2.2: Example solution to a clustering scenario, where the microphones are represented by dots. The sources are denoted by a cross, with their critical distances marked in green.

2.4.1 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 2.5.1.

2.4.1.1 Fuzzy c-Means

The first step in the fuzzy clustering procedure consists of extracting a feature set. This can be done in several ways, two of which have been previously described. The first option extracts a feature set composed of Mel-frequency cepstral coefficients (MFCCs) and their modulation spectra, as described in section 2.3.1.2. The second option is to use features from the speaker embeddings outlined in section 2.3.2.

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 minimises the least-squared error functional:

$$J = \sum_{c=0}^{C-1} \sum_{d=0}^{D-1} (\mu_{c,d})^{\alpha} \delta(\boldsymbol{\mathcal{F}}_d, \boldsymbol{\mathcal{C}}_c) \tag{2.14}$$

- C: Amount of clusters, equal to $N+1$.
- $\mu_{c,d}$: FMV of d-th microphone for cluster c.
- $\delta(\boldsymbol{\mathcal{F}}_d, \boldsymbol{\mathcal{C}}_c)$: Distance metric between features of microphone d and cluster center $\boldsymbol{\mathcal{C}}_c$).
- α: Fuzzy weighing exponent.

Choosing a higher α value results in fuzzier clusters, with $\alpha = 1$ corresponding to hard clustering. To minimize the loss function in equation 2.14, the cluster centers $\boldsymbol{\mathcal{C}}_c$ are updated iteratively using the following equations.

$$\boldsymbol{\mathcal{C}}_c = \frac{\sum_{d=0}^{D-1} \mu_{c,d}^{\alpha} \boldsymbol{\mathcal{F}}_d}{\sum_{d=0}^{D-1} \mu_{c,d}^{\alpha}} \tag{2.15}$$

$$\mu_{c,d} = \left(\sum_{\tilde{c}=0}^{C-1} \left(\frac{\delta(\boldsymbol{\mathcal{F}}_d, \boldsymbol{\mathcal{C}}_c)}{\delta(\boldsymbol{\mathcal{F}}_d, \boldsymbol{\mathcal{C}}_{\tilde{c}})} \right)^{2/(\alpha-1} \right)^{-1} \tag{2.16}$$

Several options exist for the distance metric in equation 2.14. The method described in [2] uses the standard Euclidian distance:

$$\delta_{\text{Euclid}}(\boldsymbol{\mathcal{F}}_d, \boldsymbol{\mathcal{C}}_c) = \|\boldsymbol{\mathcal{F}}_d - \boldsymbol{\mathcal{C}}_c\|_2^2 \tag{2.17}$$

With $||\cdot||$ denoting the ℓ_2 norm of a vector. When using the speaker embedding features from section 2.3.2, a great option would be to use the cosine distance. This metric only takes into account the orientation and direction of given vectors to determine their similarity. These two happen to be the most discriminating parts of the speaker embedding features [10].

$$\delta_{\text{Cos}} = 1 - \frac{\boldsymbol{\mathcal{F}}_d^T \boldsymbol{\mathcal{C}}_c}{||\boldsymbol{\mathcal{F}}_d||_2 ||\boldsymbol{\mathcal{C}}_c||_2} \tag{2.18}$$

An example clustering result using FCM 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

2.4.2 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) [16] 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 [17]. The task of adapting CFL to an unsupervised scenario and implementing it effectively within the context of ASNs is a complex and challenging endeavor.

2.4.2.1 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

$$\boldsymbol{\theta}^{\tau+1} = \sum_{i=1}^{M} \frac{|\boldsymbol{D}_i|}{|\boldsymbol{D}|} \boldsymbol{\Delta\theta}_i^{\tau} \tag{2.19}$$

- 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 [18, 19] 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.

$$a_{i,j} = \frac{\langle \boldsymbol{\Delta\theta}_i, \boldsymbol{\Delta\theta}_j \rangle}{||\boldsymbol{\Delta\theta}_i, \boldsymbol{\Delta\theta}_j||} \tag{2.20}$$

- $\langle \cdot \rangle$ 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.

$$\max_{\forall i \in c_1, k \in c_2}(a_{i,k}) < \min(\min_{\forall i,j \in c_1}(a_{i,j}), \min_{\forall k,l \in c_2}(a_{k,l})) \tag{2.21}$$

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

$$\Delta\overline{\theta}_c = \left\| \frac{1}{|c|} \sum_{i \in c} \boldsymbol{\Delta\theta}_i \right\| \text{ and } \Delta\hat{\theta}_c = \max_{i \in c}(\|\boldsymbol{\Delta\theta}_i\|) \tag{2.22}$$

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 [20]. 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 [18], a third verification $\epsilon_3 \leq |\nabla\Delta\overline{\theta}_c|$ is added [16]. 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

2.4.2.2 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 $\boldsymbol{Y}$. [21] 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 $\boldsymbol{\Theta}$.

$$\min_{\boldsymbol{\Theta}} L_{\text{mse}}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \min_{\boldsymbol{\Theta}} \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{2.23}$$

2.4.2.3 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 $\boldsymbol{q}$

$$q_i = \frac{1}{|c_x| - 1} \sum_{j \in c_x / i} a_{i,j} \tag{2.24}$$

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

$$r_i = \frac{1}{|c_y|} \sum_{k \in c_y} a_{i,k} \tag{2.25}$$

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:

$$p_i = \lambda \frac{q_i - \min(\boldsymbol{q})}{\max(\boldsymbol{q}) - \min(\boldsymbol{q})} + (1-\lambda) \frac{r_i - \min(\boldsymbol{r})}{\max(\boldsymbol{r}) - \min(\boldsymbol{r})} \tag{2.26}$$

It is necessary to perform 2.26, 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 2.26. 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

$$\mu_i = a_{i,\arg\min(p_j)}, \forall i, j \text{ and } c_x \in \{c_1, c_2\} \tag{2.27}$$

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.

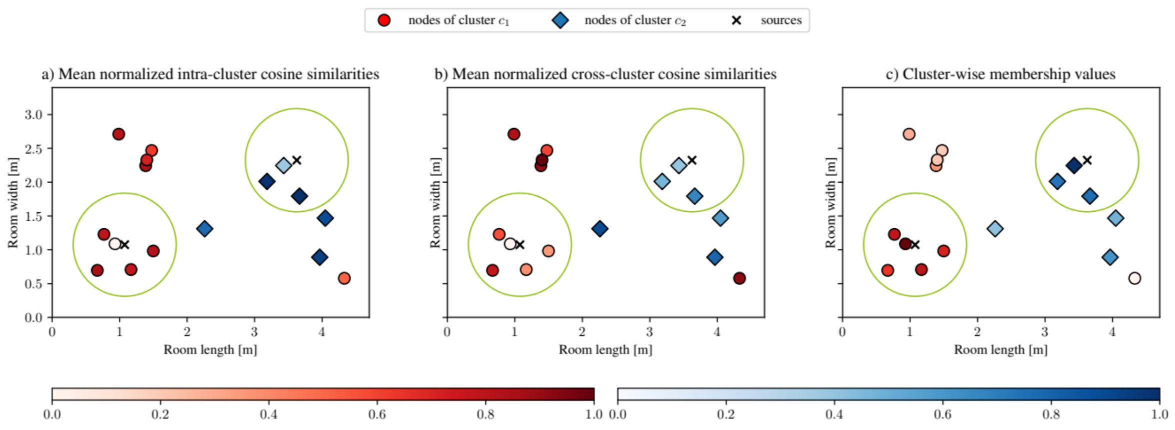


Figure 2.3: Example cluster membership values for unsupervised clustered federated learning.

2.4.3 Coherence-based clustering in the frequency domain

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 [15].

After arranging aforementioned coherence values into a symmetrical coherence matrix, 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. A possible downside to this method is the fact that all audio signals need to be transmitted to a central node in order to be able to calculate all coherence values, thus requiring more bandwidth compared to feature-based clustering methods.

2.4.3.1 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 [22]:

$$\Gamma_{xy}(f) = \frac{|S_{xy}(f)|^2}{S_{xx}(f)S_{yy}(f)} \tag{2.28}$$

- 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 $y(t)$. [23] To compute the cross-spectral density, the following equation can be used:

$$S_{xy}(f) = \sum_{k=1-T}^{T-1} R_{xy}(k)e^{-i2\pi fk} \tag{2.29}$$

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

For the special case $x(t) = y(t)$, equation 2.29 reduces to $S_{xx}(f)$.

$R_{xy}(k)$ can be estimated with:

$$R_{xy}(k) = \begin{cases} \frac{1}{T}\sum_0^{T-1-k} x(t)y(t+k) & k = 0, \dots, T-1 \\ R_{xy}(-k) & k = -(T-1), \dots, -1 \end{cases} \tag{2.30}$$

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

$$C_{xy} = \frac{\sum_{f=0}^{F} \Gamma_{xy}(f)}{F} \in [0, 1] \tag{2.31}$$

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 $\boldsymbol{C}$ can be obtained, where $C_{ii} = 1$.

$$\boldsymbol{C} = \begin{bmatrix} 1 & \cdots & \cdots & C_{1M} \\ C_{12} & 1 & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ C_{1M} & C_{2M} & \cdots & 1 \end{bmatrix} \in \mathbb{R}_{+}^{M \times M} \tag{2.32}$$

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.

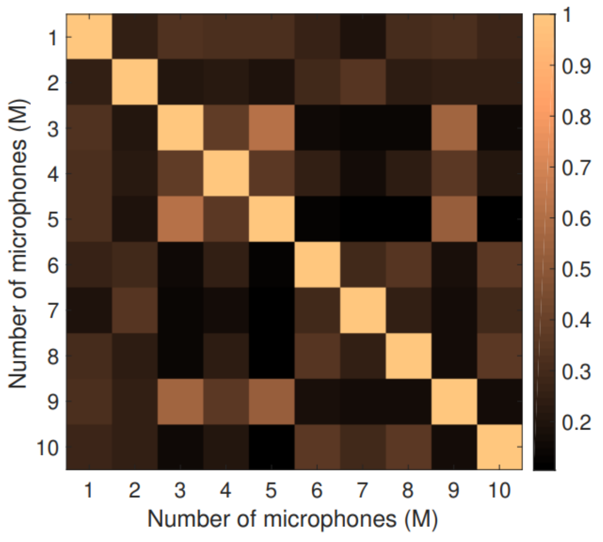


Figure 2.4: An example coherence matrix.

2.4.3.2 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 [24]. $\boldsymbol{C}$ 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:

$$\boldsymbol{C} = \boldsymbol{B}\boldsymbol{B}^T \odot (\mathbf{1} - \boldsymbol{I}) + \boldsymbol{I} \tag{2.33}$$

- $\boldsymbol{B} \in \mathbb{R}^{M \times K}$: The cluster matrix, where K is the amount of speakers (the amount of clusters)
- $\odot$: Element-wise product
- $\boldsymbol{I}$: The identity matrix
- $\mathbf{1}$: 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 [25]:

$$\boldsymbol{B} \leftarrow \boldsymbol{B} \odot \frac{(\boldsymbol{C} \odot (\mathbf{1} - \boldsymbol{I}))\boldsymbol{B}}{(\boldsymbol{B}\boldsymbol{B}^{\boldsymbol{T}} \odot (\mathbf{1} - \boldsymbol{I}))\boldsymbol{B}} \tag{2.34}$$

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

$$\gamma_m = \{j \in [1, K] : B_{mj} \geq B_{mk}, \forall k \in [1, K]\} \tag{2.35}$$

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.

Algorithm 2 displays a suggested clustering method using the aforementioned techniques [15].

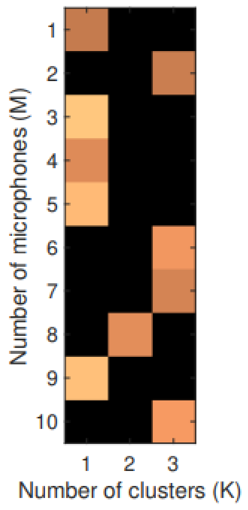


Figure 2.5: An example cluster matrix.

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 2.30
        Compute the CSD $S_{x_i,x_j(f)}$ using 2.29
        Obtain the coherence measure $\Gamma_{x_i,x_j}(f)$ using 2.28
        Compute the coherence metric $C_{x_i,x_j}$ using 2.31
    end for
end for
Initialize $\boldsymbol{B}$ with random values.
for iters $= 1$ to maxIter do
    Update $\boldsymbol{B}$ according to 2.34
end for
Obtain the optimal clustering by 2.35
```

2.5 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 [26]. 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 [27].

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'.[28] 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 2.4.

2.5.1 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 using the results from the method described in section 2.4.1, the following 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.

2.5.1.1 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 [29]:

$$\mathscr{M}_n(k,b) = \begin{cases} 1 & \text{if source } n \text{ is dominant at } (k,b) \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

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:

$$\mathscr{M}_n(k,b) = \begin{cases} 1 & |X_{R_n}(k-b)| > \frac{1}{B}\sum |X_{R_j}(k,b)|, \\ & \quad j = 1, \ldots, N \text{ and } j \neq n \\ 0 & \text{otherwise} \end{cases} \tag{2.37}$$

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.

$$\tilde{X}_{i_n}(k,b) = X_{i_n}(k,b)\mathscr{M}_n(k,b) \tag{2.38}$$

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.

$$\hat{s}_{n,\text{W-DSB}}(l) = \sum_{i_n} w_{n,i_n} x_{i_n}(l + D_{i_n}) \tag{2.39}$$

- 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 [29], all weights were set uniformly, but in [14] 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 2.37.

$$\mathscr{M}_{n,\text{DSB}}(k,b) = \begin{cases} 1 & |\hat{S}_{n,\text{FMVA-DSB}}(k,b)| > \frac{1}{B}\sum |\hat{S}_{j,\text{FMVA-DSB}}(k,b)| \\ & \quad j = 1,\dots,N \text{ and } j \neq n \\ 0 & \text{otherwise} \end{cases} \tag{2.40}$$

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 2.6 depicts a high-level view of the algorithm in the case of two sources.

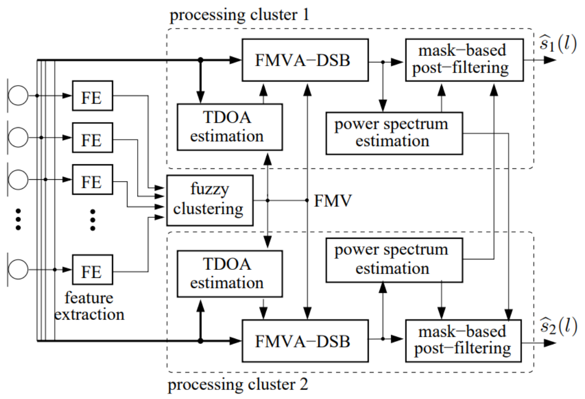


Figure 2.6: Schematic representation of the algorithm for 2 clusters.

3 Proposed Methods

The previous chapter discussed the state-of-the-art. This chapter aims to experiment with new methods and variations on existing methods for clustering.

First of all, some variations and possible improvements to the frequency-domain coherence-based clustering method proposed by [15] will be discussed. The method has already been outlined in section 2.4.3. In the following sections, an overview of alternative methods for calculating the coherence matrix are described, as well as the possibility of adding an additional background cluster. Furthermore, instead of randomizing the initial values of matrix $\boldsymbol{B}$, it could be populated with values generated by an initialization method based on SVD. This could increase the performance of the coherence-based methods.

It is also possible to calculate an estimated coherence metric from the time-domain representation of microphone signals. This method, which from now on will be referred to as time-domain coherence shows promise as an alternative to the frequency-based method. This is mainly due to the fact that the method could be applied to a live setup, where the coherence is continuously updated through time, thus allowing the method to adapt to changes. This could be useful in a real world scenario, where (more often than not) sources tend to not stay in one place.

Afterwards, some other methods that, when used in addition to the previous methods, could improve clustering performance are discussed.

3.1 Frequency domain coherence-based clustering

3.1.1 Coherence matrix calculation

Section 2.4.3 describes a method for calculating a symmetrical coherence matrix. Alternatively, the coherence values can be calculated using Welch's method [30], implemented in the *signal.coherence()* function from Scipy. This implementation calculates a magnitude-squared coherence estimate using a fast Fourier transform.

3.1.1.1 Welch's method

The method proposed by Peter D. Welch computes an estimate of the power spectral density by dividing the data into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms. This power spectral density estimate can in turn be used to calculate an estimated coherence value.

Consider $x(t)$ and $y(t)$, two signals of length N, which are divided into n segments. Welch's method estimates the power spectral density (PSD) of the two signals using the following steps:

Each signal is divided into n overlapping segments of equal length L, so that $N = nL$ and the overlap between adjacent segments is $M = L - K$, where each segment overlaps the previous one with K samples. The windowed segments at index j are denoted as $x_j(t)$ and $y_j(t)$ for the two signals.

$$x_j(t) = x(t + jK) \cdot \mathcal{W}(t) \quad \text{and} \quad y_j(t) = y(t + jK) \cdot \mathcal{W}(t), \tag{3.1}$$

- $\mathcal{W}(t)$: The window function applied to each segment.

The discrete Fourier transform (DFT) of each segment is calculated:

$$X_j(\omega) = \sum_{t=0}^{L-1} x_j(t)e^{-j\omega t} \quad \text{and} \quad Y_j(\omega) = \sum_{t=0}^{L-1} y_j(t)e^{-j\omega t} \tag{3.2}$$

- ω: The angular frequency.

The PSD estimate is calculated for each segment by taking the magnitude squared of the DFT and subsequently averaging over all segments.

$$S_{xx}(f) = \frac{1}{nL}\sum_{j=1}^{n} |X_j(f)|^2 \quad \text{and} \quad S_{yy}(f) = \frac{1}{nL}\sum_{j=1}^{n} |Y_j(f)|^2 \tag{3.3}$$

- f: The frequency in Hz corresponding to ω.

The cross-spectral density (CSD) estimate for the two signals can be obtained by multiplying the DFT of one signal by the complex conjugate of the DFT of the other signal and averaging over all segments:

$$S_{xy}(f) = \frac{1}{nL}\sum_{j=1}^{n} X_j(f)Y_j^*(f) \tag{3.4}$$

- Y_j^*: The complex conjugate of Y_j

Finally, the coherence estimate is calculated for the two signals by dividing the CSD estimate by the product of the two PSD estimates, similar to equation 2.28:

$$\Gamma_{xy}(f) = \frac{|S_{xy}(f)|^2}{S_{xx}(f)S_{yy}(f)}. \tag{3.5}$$

According to Welch himself, this method reduces computational overhead. Furthermore, the method should should be less susceptible to noise, making it an ideal candidate for the clustering of often noisy microphone signals.

3.1.1.2 Implementation

Code fragment 3.1 shows a possible python implementation for the calculation of the coherence matrix. Since coherence is a function of the frequency, it is averaged over all frequency bins, as outlined in equation 2.31.

```
def calculate_coherence_matrix(self):
    num_channels = self.Signals.shape[0]
    num_bins = self.Signals.shape[1]

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

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

Code Fragment 3.1: Calculation of coherence matrix in python

3.1.2 Clustering adaptations

3.1.2.1 Amount of clusters

The frequency-domain coherence-based clustering method from section 2.4.3 results in N clusters, corresponding to the amount of sources. On the other hand, the methods using features froms MFCCs and speaker embeddings in [10, 13, 14, 31] add an additional background cluster for microphones dominated by reverberation and interference.

It would be interesting to investigate the possibility of adding a background cluster to the coherence-based clustering algorithm. If so, it's clustering results could be objectively compared with the other aforementioned methods. How this can be done, is elaborately explained in chapter 4.

3.1.2.2 Non-negative matrix factorization

In the algorithm described in section 2.4.3, 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 [32]. An implementation of this method is displayed in code fragment 3.4. //TODO more about SVD

```
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

```
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 $\boldsymbol{B}$

3.1.2.3 NNDSVD-based initialization

NNDSVD (Non-negative Double Singular Value Decomposition) is an initialization method for non-negative matrix factorization (NMF) that aims to provide a good starting point for the iterative NMF algorithm. It's based on a non-negative approximation of the Singular Value Decomposition (SVD) of the input matrix. the following is a summary of the steps involved in NNDSVD-based matrix initialization:

1. **Compute the SVD of the input matrix A:** Perform Singular Value Decomposition to decompose the matrix A into three matrices U, S, and V, where U and V contain the left and right singular vectors, and S contains the singular values.
2. **Extract the top-k singular triplets:** Choose the top-k singular triplets (singular values and their corresponding singular vectors) from the decomposed matrices.

3. **Initialize the nonnegative matrices W and H:** Iterate through the k singular triplets, and for each triplet, do the following:

   (a) Split the left and right singular vectors into their positive and negative parts.

   (b) Calculate the norms of the positive and negative parts.

   (c) Choose the pair of positive/negative parts that result in the highest product of their norms.

   (d) Normalize the chosen pair of singular vectors and multiply them by the square root of the corresponding singular value scaled by the product of their norms.

   (e) Assign the resulting vectors to the columns of matrix W and rows of matrix H.

NNDSVD-based matrix initialization presents several possible benefits. First of all, NNDSVD initialization often leads to faster convergence in NMF algorithms compared to random initialization, as it starts from a nonnegative low-rank approximation of the input matrix, which is closer to the optimal solution.

The initialization provides a better initial point for the NMF algorithm, leading to more rapid error reduction compared to other initialization techniques like random initialization. Furthermore, it reduces the dependency on random seeds, as it is a deterministic method that relies on the SVD of the input matrix instead of randomly generated values.

An python implementation using this method already exists [33], and is displayed in code fragment 3.4.

```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 $\boldsymbol{B}$

3.2 Time-domain coherence-based clustering

Another way to determine microphone clusters could be to calculate coherence in the time-domain representation of the signals. It demonstrates promising results and presents several advantages, particularly in dynamic environments.

One of the main benefits of time-domain coherence is its applicability in real-time settings. As coherence can be continuously updated over time, this method can adapt to changes in the sources being recorded, in contrast to frequency-based coherence, which is calculated across the entire signal. In real-world scenarios, sources are rarely stationary and tend to move around, making it challenging to maintain stable microphone clusters using traditional frequency-based methods. Time-domain coherence could, therefore, be a useful tool for capturing these dynamic changes and maintaining accurate microphone clusters.

Furthermore, time-domain coherence offers an alternative perspective on the relationship between signals, providing additional information that may not be evident in frequency-domain analysis. By examining the coherence of the time-domain representation of signals, it could be possible to gain insights from the temporal behavior of the signals, which might be helpful for determining microphone clusters.

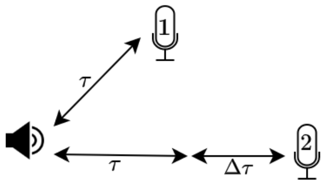


Figure 3.1: Illustration scenario with one source and two microphones.

3.2.1 Adaptive filtering

Firstly, two signals x and y are considered, which correspond to the observations of microphones 1 and 2 depicted in figure 3.1. In this case, with one active source without interference or reflections, the signals correspond to their direct path components. Under these circumstances, y is a delayed version of x with time delay $\Delta\tau$. The theory is that in a real-world scenario with multiple sources, x and y will more closely resemble time shifted versions of each other when they are dominated by the same source.

$$\begin{aligned} x(n) &= s(n-\tau) \\ y(n) &= s(n-\tau-\Delta\tau) \end{aligned} \tag{3.6}$$

The fact that y is a time-delayed version of x implies that a filter h exists so that $x * h \approx y$. An n-tap FIR filter $H(z)$ can be used for this purpose.

$$H(z) = h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-(N-1)} \tag{3.7}$$

So that

$$h(n) * x(n) = \sum_{k=0}^{N-1} h_k x(n-k) = \hat{y}(n) \approx y(n) \tag{3.8}$$

The objective is to minimize the cost function $J(\boldsymbol{h})$, which represents the sum of squared differences between y and $\hat{y}$ over M samples of x and y:

$$J(\boldsymbol{h}) = \sum_m (y(m) - \hat{y}(m))^2 \sum_m e^2 \tag{3.9}$$

With:

- $\boldsymbol{h} = [h_0, h_1, \cdots, h_{N-1}]$
- $\hat{y}(m) = \sum_{k=0}^{N-1} h_k x(m-k)$

The gradient of $J(\boldsymbol{h})$ can be calculated as follows.

$$\frac{\partial J(\boldsymbol{h})}{\partial \boldsymbol{h}} = -2\sum_m = e(m)x(m-k) = -2\sum_m (y(m) - \hat{y}(m))x(m-k) \tag{3.10}$$

By setting the gradient of $J(\boldsymbol{h})$ to zero and substituting $\hat{y}(m) = \sum_\ell h_\ell x(m-\ell)$

$$\sum_\ell \sum_m x(m-\ell)x(m-k) = \sum_m y(m)x(m-k) \tag{3.11}$$

With $\sum_m x(m-\ell)x(m-k)$ being the auto-correlation of the signal.

$$\sum_m x(m-\ell)x(m-k) \approx \sum_m x(m)x(m-|k-\ell|) \triangleq R_{xx}(|k-\ell|) \tag{3.12}$$

$$\sum_m y(m)x(m-k) \triangleq R_{xy}(k) \tag{3.13}$$

This results in the following matrix equation.

$$\begin{bmatrix} R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & \cdots & R_{xx}(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & \cdots & R_{xx}(0) \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \end{bmatrix} = \begin{bmatrix} R_{xy}(0) \\ R_{xy}(1) \\ \vdots \\ R_{xy}(N-1) \end{bmatrix} \tag{3.14}$$

Solving equation 3.14 requires the calculation of a matrix inverse, which induces a great amount of computational overhead when N is large. This renders the method unusable for the real-time situations. The solution is to adapt $\boldsymbol{h}$ sample-by-sample with stochastic gradient descent.

3.2.2 Stochastic gradient descent

The filter $\boldsymbol{h}$ can be iteratively estimated so that $\hat{y}(n) = \sum_{k=0}^{N-1} h_k(n)x(n-k)$, where $\boldsymbol{h}(n)$ is the estimate of $\boldsymbol{h}$ at time n. This is done by evaluating the instantaneous gradient of the cost function $J(\boldsymbol{h}(n))$:

$$J(\boldsymbol{h}(n)) = (y(n) - \hat{y}(n))^2 \tag{3.15}$$

$$\frac{\partial J(\boldsymbol{h})}{\partial h_k}\Big|_{\boldsymbol{h}=\boldsymbol{h}(n)} = -2x(n-k)(y(n) - \hat{y}(n))\Big|_{\boldsymbol{h}=\boldsymbol{h}(n)} \tag{3.16}$$

$$\tag{3.17}$$

This allows for an iterative estimation of $\boldsymbol{h}(n)$, where it is updated by following the direction of the negative gradient with learning rate α.

$$h_k(n+1) = h_k(n) - \alpha\frac{\partial J(h)}{\partial h_k}\Big|_{\boldsymbol{h}=\boldsymbol{h}(n)} \tag{3.18}$$

$$= h_k + 2\alpha(y(n) - \hat{y}(n))x(n-k) \tag{3.19}$$

3.2.3 Coherence estimation

The aformentioned developments allow for the calculation of an estimated coherence measure, which is calculated as follows.

Resulting from 3.14:

$$h_0 = \frac{R_{xy}(0)}{R_{xx}(0)} \tag{3.20}$$

The cross spectral density is equal to the Fourier transform of the cross-correlation:

$$R_{xy}(t) = \sum_f S_{xy}(f)e^{-j2\pi ft} \tag{3.21}$$

When considering $t = 0$, this results in:

$$R_{xy}(0) = \sum_f S_{xy}(f) \tag{3.22}$$

So

$$h_0 = \sum_f \frac{S_{xy}(f)}{S_{xx}(f)} \tag{3.23}$$

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

The fourier transform of the convolution of h and h' results in an estimated coherence measure.

$$h * h' \xrightarrow{\mathfrak{F}} \frac{|S_{xy}|^2}{S_{xx}S_{yy}}(f) \tag{3.24}$$

Combining this with 3.23 results in:

$$(h * h')_0 = \sum_f \frac{|S_{xy}|^2}{S_{xx}S_{yy}}(f) \tag{3.25}$$

Important to note is that h can only be a causal filter. For this reason, both x and y are delayed in time in the actual implementation of this method.

Figures 3.3 and 3.4 show example estimations for h and h' (denoted h_1 and h_2, respectively) and their convolutions for microphones with different relationships. The acoustic scenario considered is illustrated in Figure 3.2.

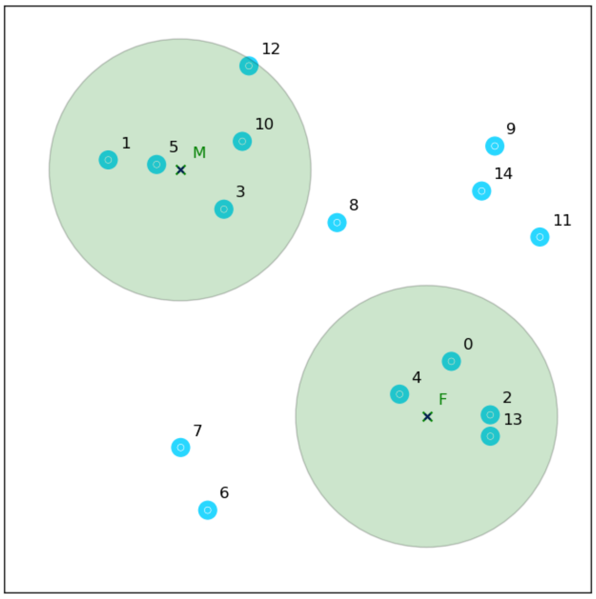
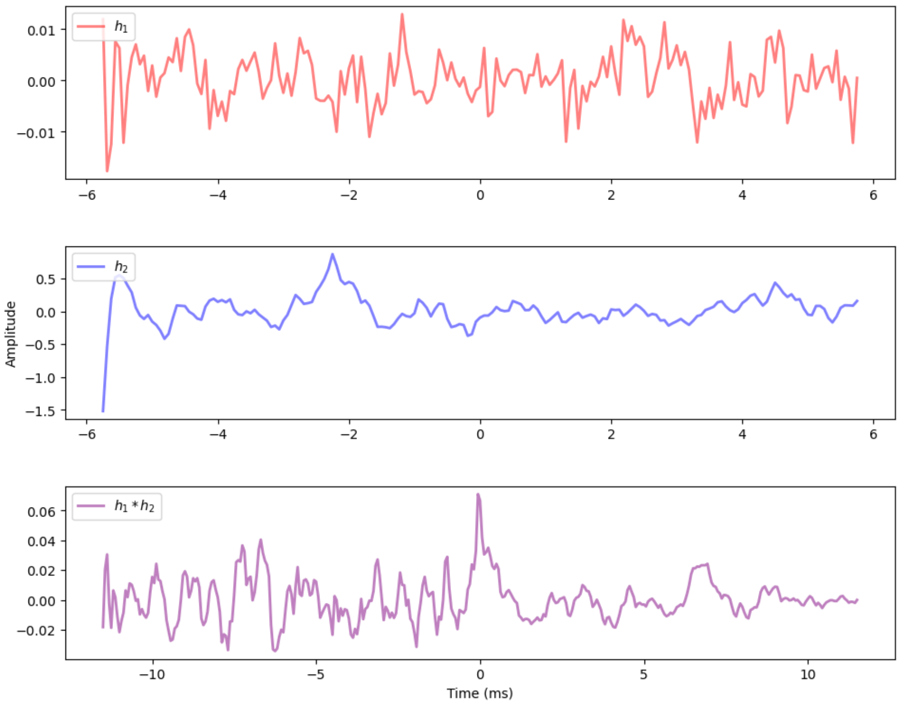


Figure 3.2: Example acoustic scenario.

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



(b) Microphones 3 and 1.

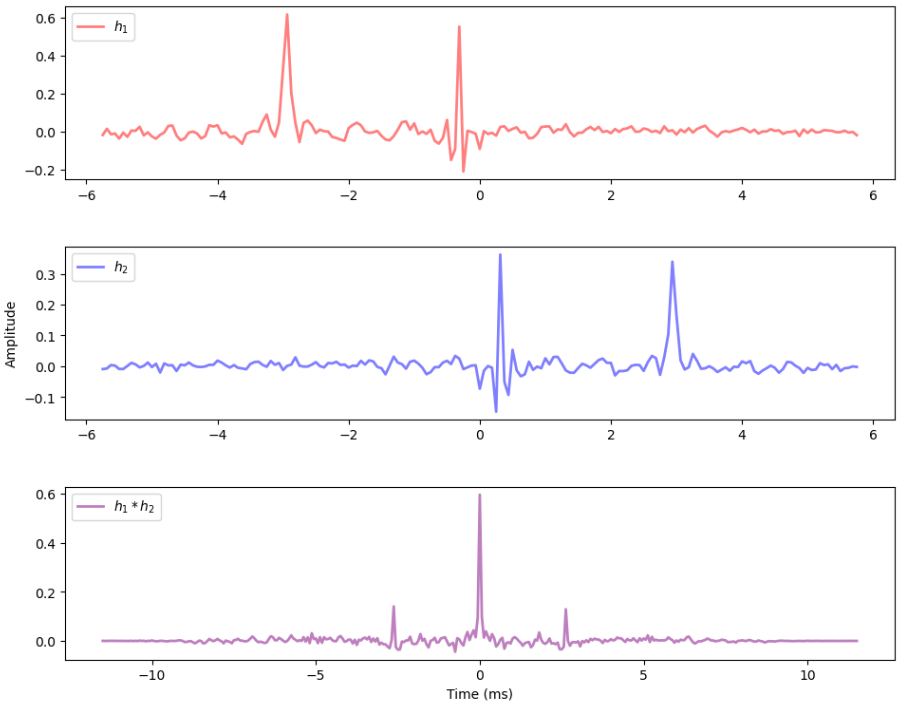
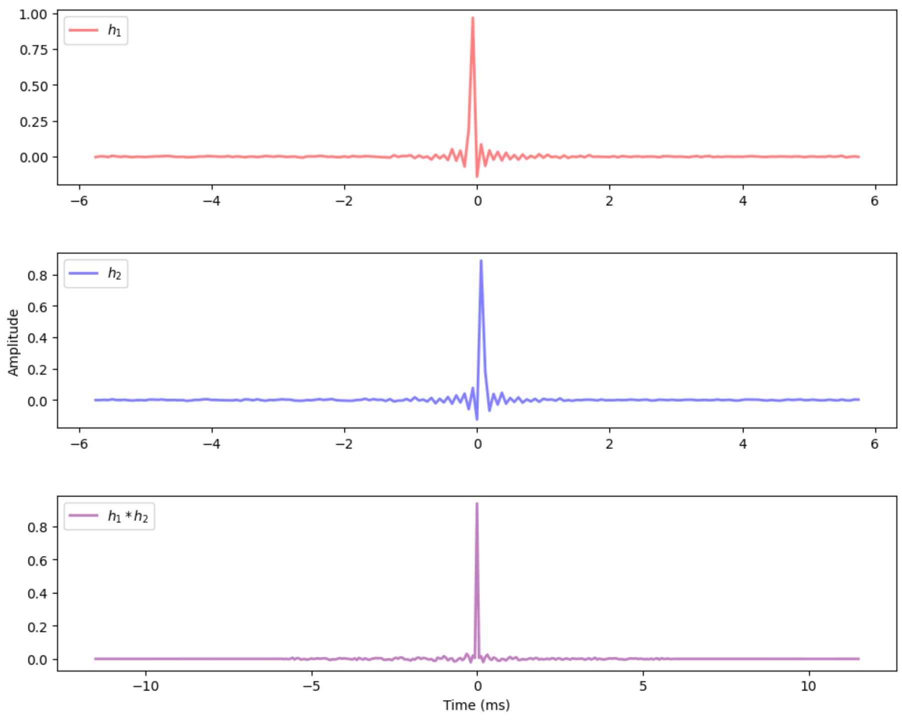


Figure 3.3: Example h_1 and h_2 filters and the result of their convolutions.

(a) Microphones 2 and 13.



(b) Microphone 0, compared to itself.

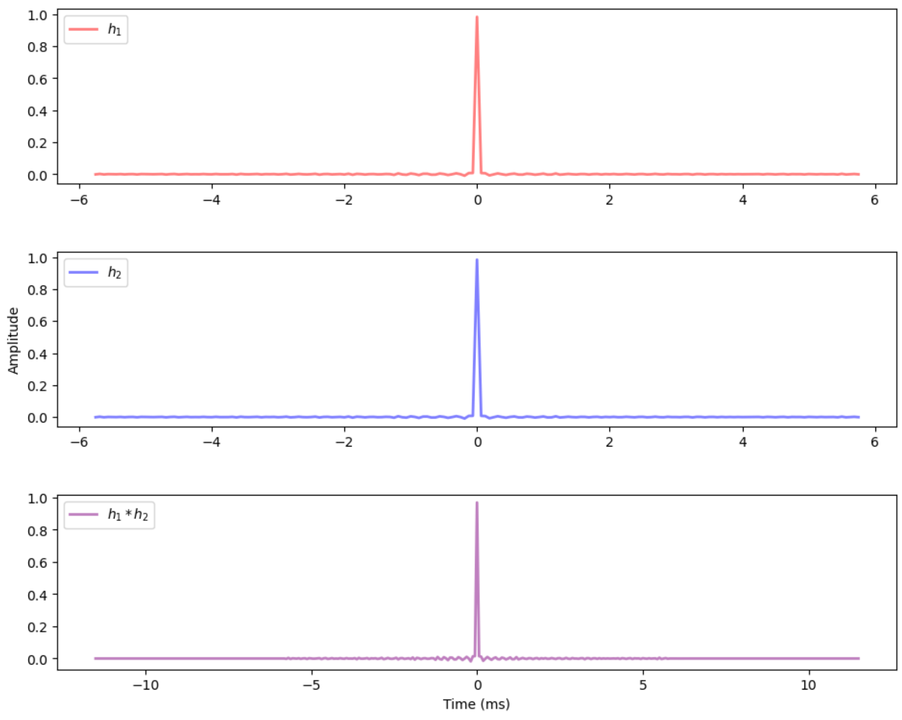


Figure 3.4: Example h_1 and h_2 filters and the result of their convolutions.

3.2.4 Filter size

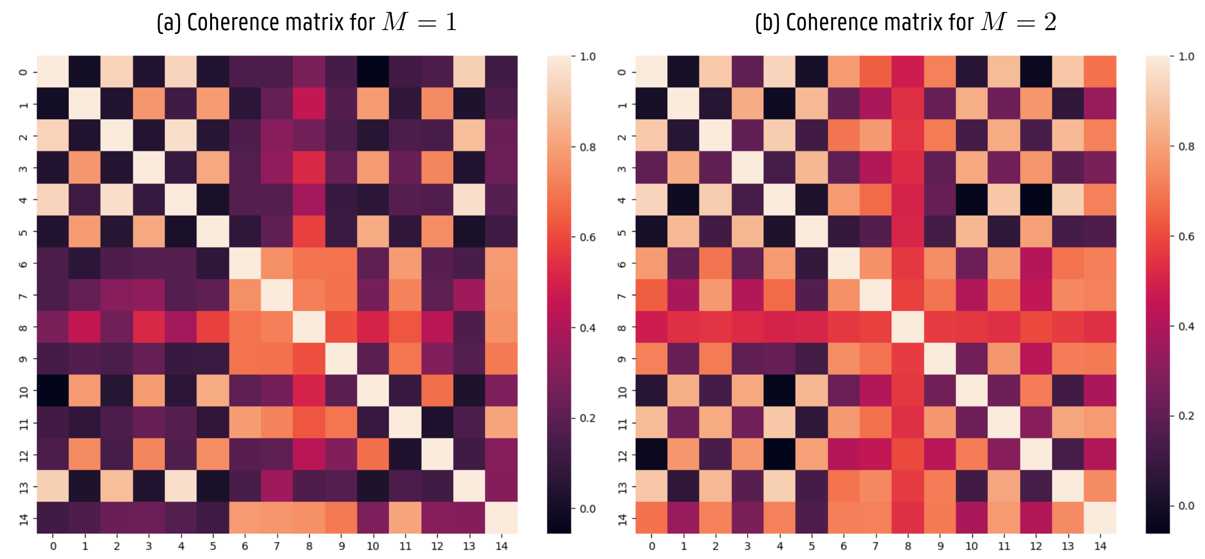


Figure 3.5: Coherence matrices for different filter sizes

//TODO time delay explanation

//TODO different values of M, what does M mean

//TODO two different methods for coherence measure (will I compare them?)

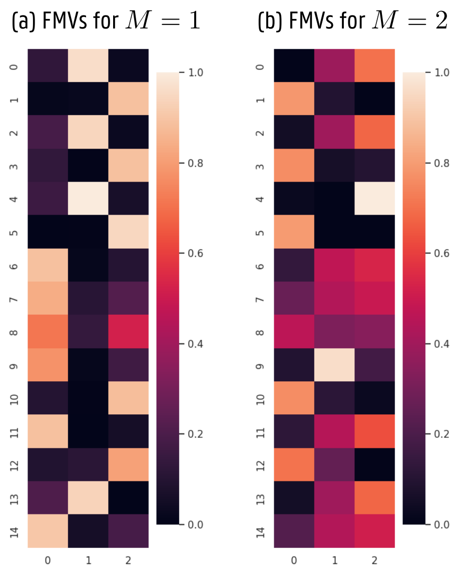
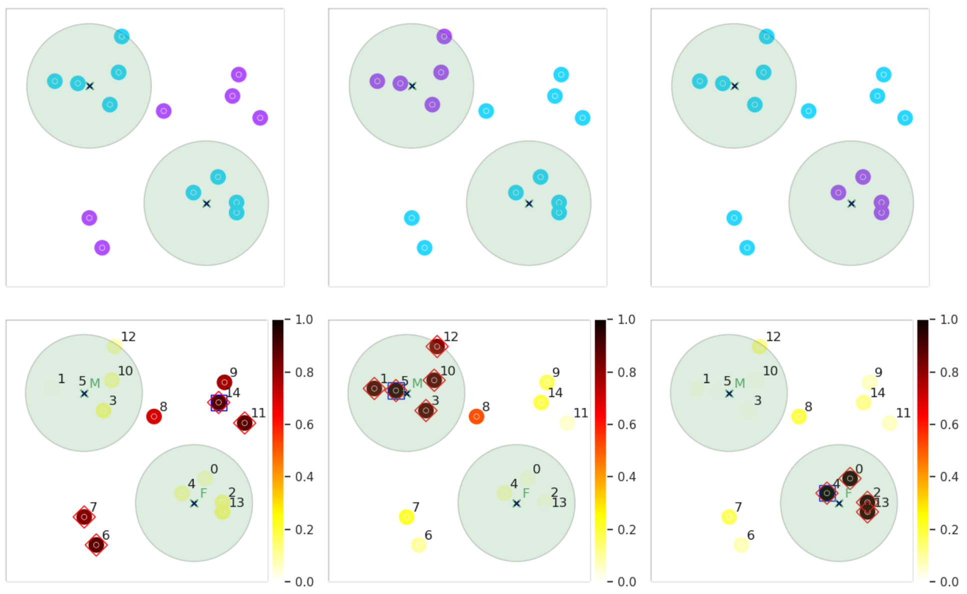


Figure 3.6: Fuzzy membership values for different filter sizes

(a) Clustering result for $M = 1$



(b) Clustering result for $M = 2$

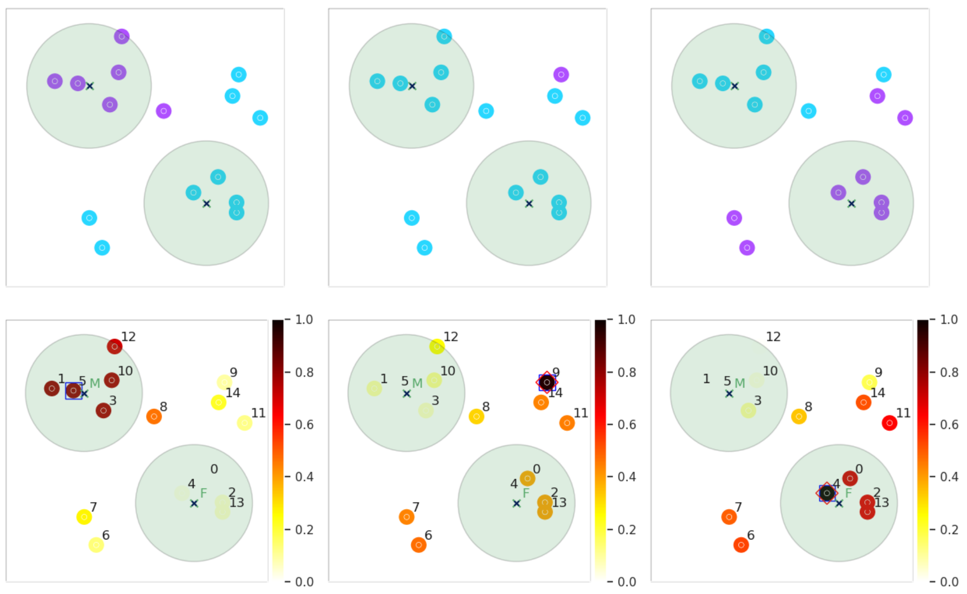


Figure 3.7: Clustering results for different filter sizes

```
def calculate_coherence_matrix(self, mode="frequency", M = 1, time = 1,
↪  maxIter = 1, alpha = 0.05):

    num_channels = self.Signals.shape[0]
    coherence_matrix = np.zeros((num_channels, num_channels))

    sxx = []

    for i in range(num_channels):
        h = self.NLMSCanceller(i, i, M=M, time=time, alpha=alpha,
        ↪  maxIter=maxIter)
        Fi = scipy.signal.fftconvolve(h, h, mode="valid")
        sxx.append(Fi[0])

    for i in range(num_channels):
        coherence_matrix[i,i] = 1
        for j in range(i+1, num_channels):
            h1 = self.NLMSCanceller(i, j, M=M, time=time,
            ↪  maxIter=maxIter,alpha=alpha)
            h2 = self.NLMSCanceller(j, i, M=M, time=time, maxIter=maxIter,
            ↪  alpha=alpha)
            F = scipy.signal.fftconvolve(h1, h2, mode='valid')
            som = 2*F[0]/(sxx[i]+sxx[j])
            coherence_matrix[i, j] = som
            coherence_matrix[j, i] = som

    self.coherence_matrix = coherence_matrix
    return self.coherence_matrix
```

Code Fragment 3.5: Time-domain coherence in python

3.3 Additional enhancements

3.3.1 AR filtering

An autoregressive (AR) signal model is a way to describe and predict time series data based on its own past values. The key idea behind AR models is that the current value of a signal (or time series) can be expressed as a linear combination of its previous values, along with some random noise. This means that an AR model attempts to predict the value at the current time step based on the values at previous time steps.

The AR model generates coefficients a_k that can be used to predict the signal based on the previous samples:

$$x_p(n) = \sum_{k=1}^{K} a_k \, x(n-k) \tag{3.26}$$

where the coefficents a_k are a solution of the form $\mathbf{Ga} = \mathbf{B}$:

$$\begin{bmatrix} R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(K-1) \\ R_{xx}(1) & R_{xx}(0) & \cdots & R_{xx}(K-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(-K+1) & R_{xx}(-K+2) & \cdots & R_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_K \end{bmatrix} = \begin{bmatrix} R_{xx}(\mathbf{1}) \\ R_{xx}(2) \\ \vdots \\ R_{xx}(K) \end{bmatrix} \tag{3.27}$$

with $R_{xx}(\tau)$ the auto-correlation of at time lag τ. Since the signal is real, this is equal to:

$$R(\tau) = \sum_{n} x(n)\, x(n+\tau) \tag{3.28}$$

3.3.2 Butterworth filter

3.3.3 Frame skipping

//TODO

4

Evaluation

4.1 Evaluation setups

4.1.1 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 mostly used for implementation and experimentation. It provides an easy way to test algorithms during development.

4.1.2 SINS dataset

SINS (Sound INterfacing through the Swarm) 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 [34].

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. The apartment consists of a large living area, a bedroom, a bathroom, a toilet, and a hall, with a total floor area of 50m^2 An illustration of the apartment is provided in figure 4.1. The different possible options for the locations of room impulse responses are depicted in figure 4.2.

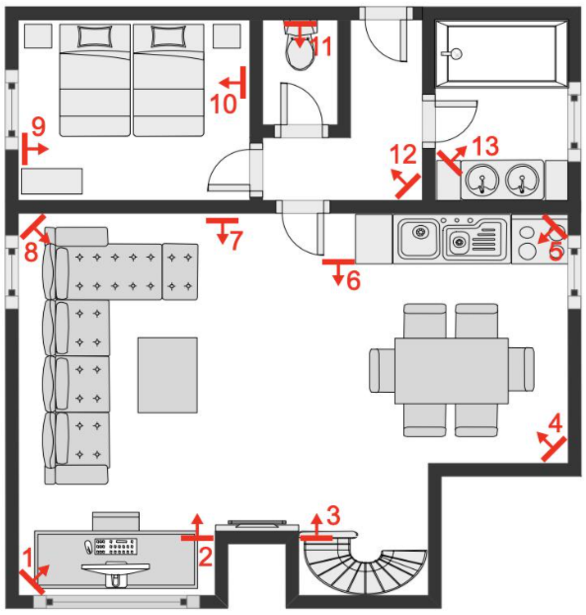


Figure 4.1: Setup of the vacation home.

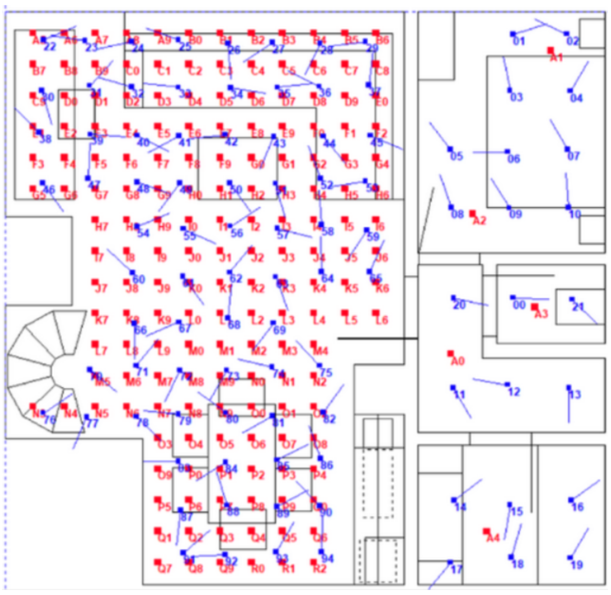


Figure 4.2: Different options of the SINS dataset.

4.2 Evaluation Metrics

4.2.1 Overview

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) [15] or distances [35, 36] 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. One way to do so would be to evaluate the distributions of the direct-to-reverberant ratio (DRR) and the direct-to-reverberant, interference, and noise ratio (DRINR) for each microphone allocated to a speech-source cluster [13].

However good the clustering quality, it is still important to evaluate the quality of of the subsequent source separation, since this is the main purpose of clustering. These metrics indicate the quality if the source separation, but can also give an indication of clustering quality, since good clusters will lead to better source separation. In this regard, three metrics will be utilized, which are identical to those used in [10]. These include the source-to-interference ratio (SIR), the perceptual evaluation of speech quality (PESQ), and the short-time objective intelligibility (STOI).

Using the aforementioned metrics would provide a fair framework for comparing the coherence-based clustering methods with the mod-MFCC and speaker embeddings-based clustering methods, since the authors of [10] used these metrics as well, allowing for a direct performance comparison.

4.2.2 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) [13]. These metrics are computed for source-dominated microphone clusters, and are computed as follows for a microphone signal y_m and source cluster c:

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

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

//TODO notatie fixen

If the evaluated clustering method is good at selecting only the microphones with relevant information for the speaker of that cluster, the resulting DRR and DRINR distributions should be centered around higher values. This makes it possible to compare the clustering quality of different algorithms by plotting and overlaying the distribution of any of the two metrics from multiple scenarios.

Additionally, the amount of spacial diversity available for subsequent tasks like separation can be measured by keeping track of the average number of microphones allocated to a source cluster [10].

4.2.3 Source Separation Metrics

4.2.3.1 Source-to-interference ratio

The SIR metric (source-to-interference ratio) is a performance measure used to evaluate audio source separation algorithms. It measures the quality of the separated audio signals by comparing the energy of the desired source signal to the energy of interference from other sources. The 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" [37] and is expressed in decibels (dB).

To summarize in brief, the SIR is calculated based on the orthogonal projection decomposition of the estimated source signal, resulting in the following components: a version of the true source signal modified by an allowed distortion, interferences from other sources, noise from sensors, and artifacts. The SIR metric is defined as the ratio of the energy of the true source signal to the energy of the interference from other sources [38].

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

- s_{target}: The source signal, modified by an allowed distortion
- s_{interf}: The interference error term

High SIR values indicate that the separation algorithm is successful in minimizing the interference from other sources, while low SIR values suggest that the algorithm is less effective in separating the desired source signal from the mixed audio.

4.2.3.2 Perceptual evaluation of speech quality

Although SIR is a commonly used performance metric, its reliability is limited since it can yield a decent score by completely suppressing the interfering source, even if only a small portion of the target speech remains. However, this can result in poor-quality speech that lacks intelligibility. Therefore, there is a need for a more objective performance metric that can effectively assess the quality of speech in a signal [10].

The Perceptual Evaluation of Speech Quality (PESQ) metric is a method for objectively assessing the quality of speech in various network conditions. It is a combination of the Perceptual Analysis Measurement System (PAMS) and PSQM99, an enhanced version of the Perceptual Speech Quality Measure (PSQM). PESQ can assess speech quality across a wide range of network conditions, including analog connections, codecs, packet loss, and variable delay [39].

The PESQ model consists of several key processes, including:

- Level aligning and filtering the input signals to a standard listening level.
- Time aligning the signals, assuming that the delay of the system is piecewise constant.
- Auditory transformation, which maps the signals into a representation of perceived loudness in time and frequency using a psychoacoustic model.
- Processing the disturbance (the difference between the transformed signals) to extract two distortion parameters.
- Aggregating the disturbance in frequency and time using non-linear averaging (L_p norms) to obtain symmetric and asymmetric disturbance values.
- In some cases, realigning "bad intervals" where time alignment may have failed, and recalculating the disturbance.
- Predicting the Mean Opinion Score (MOS) using a linear combination of the symmetric and asymmetric disturbance parameters.

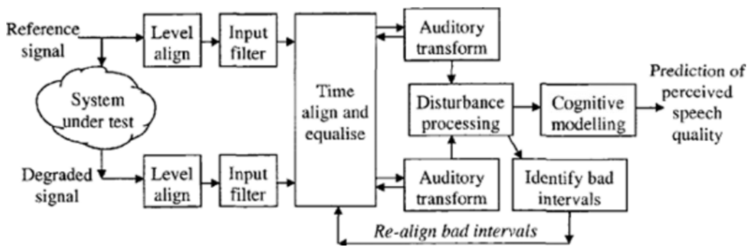


Figure 4.3: Structure of PESQ model.

The structure of the PESQ model is illustrated in figure 4.3. In summary, PESQ provides a more accurate and comprehensive assessment of speech quality across various network conditions [39].

4.2.3.3 Short-time objective intelligibility

Proposed by [40], 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.

Briefly outlined, the STOI metric is calculated according to the following steps:

- Preparation: Take the clean speech signal (original) and the processed speech signal (after applying noise reduction or speech separation techniques). Ensure they are time-aligned and have the same sample rate.
- Time-Frequency (TF) Representation: Convert both signals into their respective time-frequency representations. This step involves dividing the signals into overlapping frames, applying a window function, and then performing a Fourier transform to obtain the frequency components.
- One-Third Octave Band Analysis: Group the frequency components from the previous step into one-third octave bands. This step simplifies the frequency representation, making it more manageable for analysis.
- Intermediate Intelligibility Measure: For each TF unit (a specific time and frequency region), calculate the intermediate intelligibility measure, which estimates the linear correlation between the clean and processed speech signals. This measure gives an indication of how similar the clean and processed speech signals are in that specific TF unit.
- Normalization and Clipping: Normalize the processed speech signal's energy to match that of the clean speech signal within the specific TF region. Additionally, clip the processed signal to set a lower bound for the signal-to-distortion ratio (SDR), ensuring the processed signal is not overly distorted.
- Average Intelligibility: Finally, compute the average of the intermediate intelligibility measures across all time-frequency units and one-third octave bands. This average represents the overall STOI metric, which is a single value indicating the intelligibility of the processed speech signal compared to the clean speech signal.

4.3 Evaluation on encoded signals

In most real-world scenarios, audio signals captured by ad-hoc microphones need to be transmitted wirelessly or through other channels. To transmit these signals efficiently, they are often compressed and encoded into a different format before transmission.

However, the encoding process can introduce various types of distortions and artifacts to the audio signals. These distortions and artifacts can include quantization noise, loss of high-frequency components, and compression artifacts, among others. As a result, these distortions could hinder the clustering process.

Therefore, when designing clustering algorithms for real-world applications, it is essential to ensure that they are robust against the distortions and artifacts introduced by the encoding process. The algorithms must be able to achieve a good clustering of the signals, despite the distortions and artifacts.

Overall, the practical use of clustering algorithms for audio signals captured by ad-hoc microphones heavily depends on their ability to handle the challenges posed by encoding, and it is important to carefully evaluate the robustness of these algorithms under various encoding conditions before deploying them in real-world settings.

4.3.1 LC3plus

The LC3 and LC3plus codecs were designed for various wireless communication applications, with LC3plus being an extended version of LC3 with some additional features. One of LC3's key features is the ability to provide high-quality audio while reducing the required bit rate by approximately 50% compared to legacy codecs. This feature enables low-energy services that can help extend battery life or reduce product sizes, which makes sense, since the original LC3 codec was specified by the Bluetooth Special Interest Group.

LC3 and LC3plus also have several robustness features, such as high-performance packet loss concealment algorithms and forward error correction schemes, making them suitable for use in congested or heavily distorted channels. Additionally, LC3plus includes dedicated high-resolution audio modes that provide high-quality audio transmission, improving measurable audio quality and providing perceptual transparency.

LC3plus is also designed to support super wideband speech/audio quality for VoIP and DECT use cases, while its redundancy frame modes ensure stable phone calls even in congested VoIP channels. The codec can operate at several low-delay modes, down to 5ms at a 2.5ms packet size, improving transmission robustness and reducing delay in use cases such as gaming or video conferencing.

Overall, the LC3 and LC3plus codecs have a flexible design that allows them to be used in a wide range of applications beyond voice services, including high-quality music streaming. These features, combined with their robustness and low-energy requirements, make them well-suited for use in wireless communication platforms such as Bluetooth, DECT, and VoIP terminal equipment.

5 Results

In this chapter, a thorough evaluation of both frequency-domain and time-domain coherence-based clustering methods is conducted. To put these results in perspective, they are compared with the methods using feature extraction based on mod-MFCCs [2, 14, 29] and speaker embeddings [10, 13].

To conduct a comprehensive evaluation of the performance of the methods discussed in real-world scenarios, simulations were carried out using different microphone and source positions in realistic room acoustics. The assessment begins by considering simpler scenarios, in which the sound sources were positioned at adequate distances from each other, thereby preventing any overlap of their critical distances. Consequently, more difficult scenarios are considered, where sources are placed closer together, possibly causing their critical distances to overlap. This should provide more insight in the limitations of the proposed clustering methods.

Up until now, all evaluation was done directly on the signals generated from the simulated microphones. However, in real-world scenarios, signals are typically encoded and decoded using lossy audio codecs, which can introduce unwanted artifacts, noise, and distortion in the decoded signals. In order to determine the robustness of the frequency and time-domain coherence-based clustering methods, the microphone signals are first encoded and decoded using the LC3plus codec, with varying bitrates. The resulting decoded signals are then used to perform clustering.

5.1 Experimental setup

As explained in section 4.1.2, the room impulse responses from the SINS dataset are used to for the simulation setup. The sampling rate for all signals is 16kHz. To generate the simulated microphone signals, the dry sources available in the LibriSpeech clean speech database [41] are utilized. The approach used in this thesis is identical to the one employed in [10] and [42], whereby 10-second signal segments are selected from the train-clean 100 subset of the LibriSpeech database. By employing this method for the evaluation of the proposed techniques in this thesis, a fair comparison with the mod-MFCC-based and speaker embeddings-based clustering methods can be achieved.

5.1.1 Easy scenarios

The first set of scenarios aims to provide a a clear and comprehensible overview of the evaluated methods. Only microphones and sources within the living and kitchen area are selected, with one source positioned in the left half of the room, and the other in the right half. Scenarios where the critical distances of sources could potentially overlap are excluded. Subsequently, 16 simulated microphones are placed into the room, with each source having at least 3 microphones within its critical distance. The remaining microphones are positioned randomly throughout the room.

To maintain consistency with the ad-hoc scenario adopted throughout this thesis, only one microphone is chosen from each of the microphone array nodes, each of which comprises four microphones. To enhance scenario variety, a random microphone from the aforementioned arrays is selected.

5.1.2 Difficult scenarios

To increase the difficulty of microphone clustering, this set of scenarios places sound sources in closer proximity to one another. The distance between the sources is limited to no more than three times the room's critical distance. The lower bound of the inter-source distance is defined by the dataset, which sets the minimum distance between sources at 0.4m.

Given that the critical distance of the room is 0.68m, it's possible for sources to be situated within each other's critical distances. This creates a valuable chance to assess how the evaluated methods perform under extreme conditions. Figure 5.1 depicts an example of each of the two types of scenarios.

(a) Example of an easy scenario.

(b) Example of a difficult scenario.

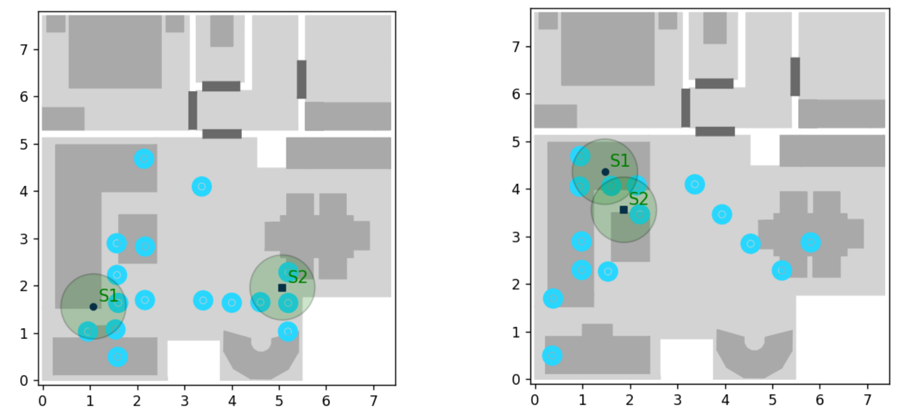


Figure 5.1: Examples of easy and difficult source positions in the SINS apartment.

5.2 Evaluation of frequency-domain coherence

5.2.1 Comparison between MFCC, SpVer and frequency-domain coherence

The metrics from section 4.2.2 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 [10].

5.2.1.1 Easy scenarios

At first glance, the DRINR and DRR values indicate good clustering results when using the coherence-based method. In this scenario, coherence-based clustering even slightly outperforms speaker verification. When compared to the MFCC-based technique, frequency-domain coherence seems to be quite a lot better in picking up source-dominated microphones, as can be seen in figure 5.2 (c, d).

As for the comparison with speaker embeddings, the difference is less noticeable. A slightly higher main lobe in 5.3 (a,b) suggests that the coherence-based method was able to find slightly more useful microphones. In this case, when considering easy scenarios with sufficient space between sources, the frequency-domain coherence seems to perform slightly better in terms of clustering quality according to the DRINR and DRR. The intuition that coherence between microphones that are closer together would be higher than that of more distantly spaced microphones would imply that the coherence-based method works better in scenarios where sources are sufficiently spaced apart.

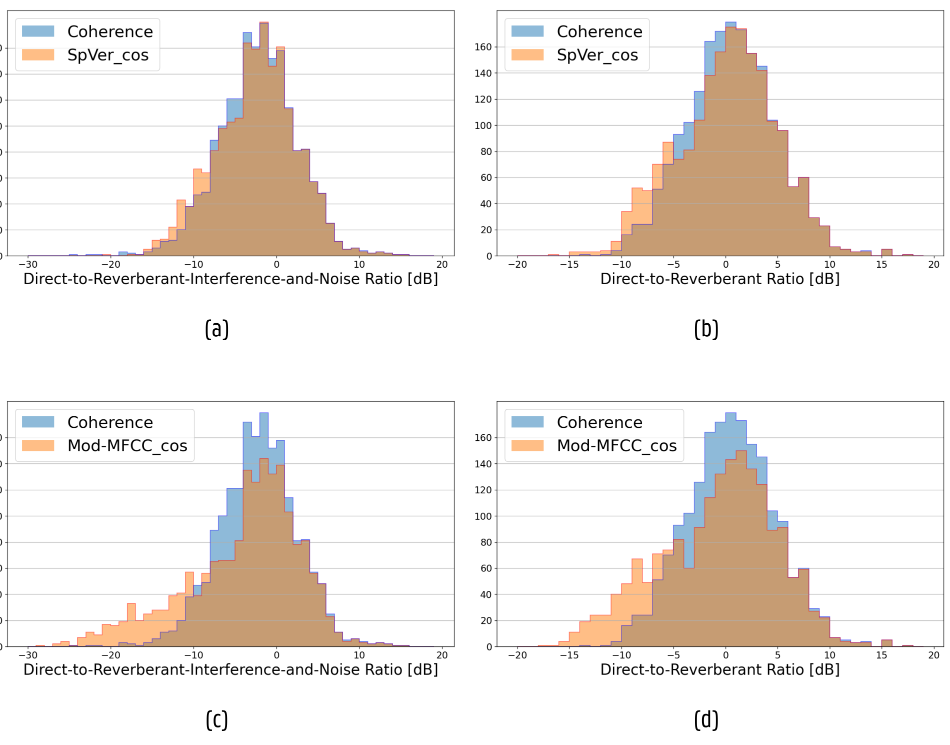


Figure 5.2: Comparing speaker verification and mod-MFCC with the coherence-based method for easy scenarios

Similar conclusions can be made from the SIR, PESQ, and STOI metrics displayed in figure 5.3. Here, frequency-domain coherence seems to slightly outperform the other methods as well, on all three source separation metrics. This is quite an unexpected outcome, given the simplicity of the coherence-based method.

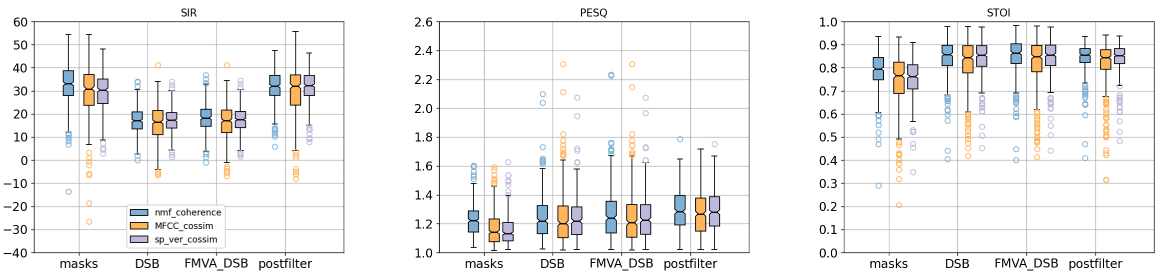


Figure 5.3: SIR, PESQ and STOI metrics for easy scenarios.

5.2.1.2 Difficult scenarios

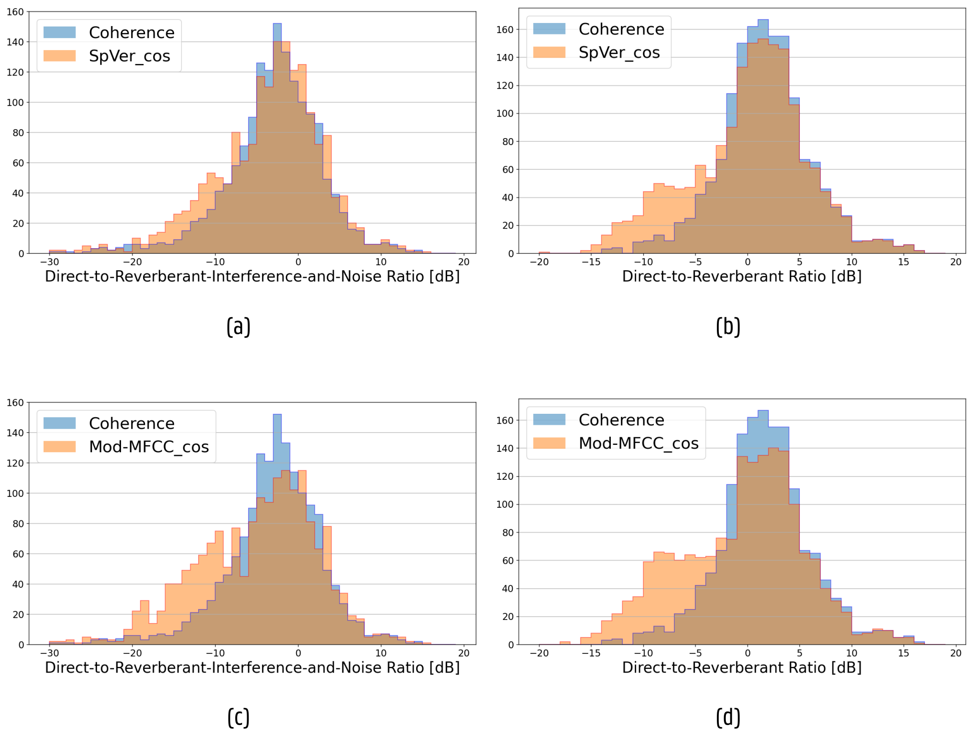


Figure 5.4: Comparing speaker verification and mod-MFCC with the coherence-based method with difficult scenarios

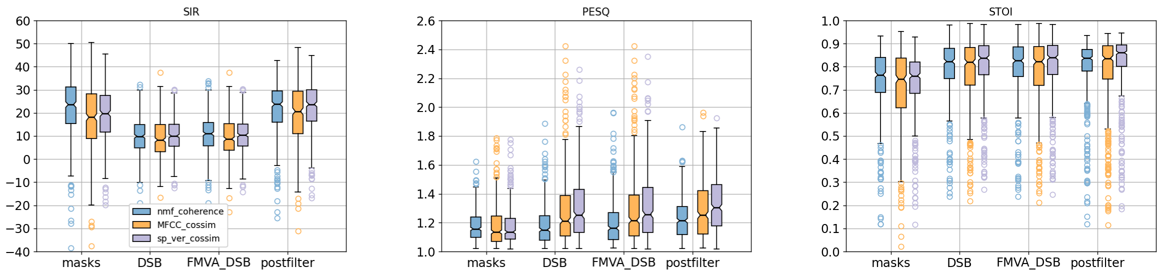


Figure 5.5: SIR, PESQ and STOI metrics for difficult scenarios.

5.2.2 SVD-based initialization

Promising as the method might have seemed, the NNDSVD-based matrix initialization shows no improvement in clustering performance, as can be seen in 5.7. The distribution of the SVD-based method is lightly shifted to the left, which indicates that the method tends to select less useful microphones. Here, the method is compared to the performance of random initialization. Furthermore, it seems to perform slightly worse on all three separation metrics, shown in figure 5.10.

Diving deeper into specific scenarios reveals that the method sometimes results in only two clusters when tested on the more difficult scenarios from the SINS database. This might indicate that this approach for the initialization prior to NMF is not suited for clustering ad-hoc microphones.

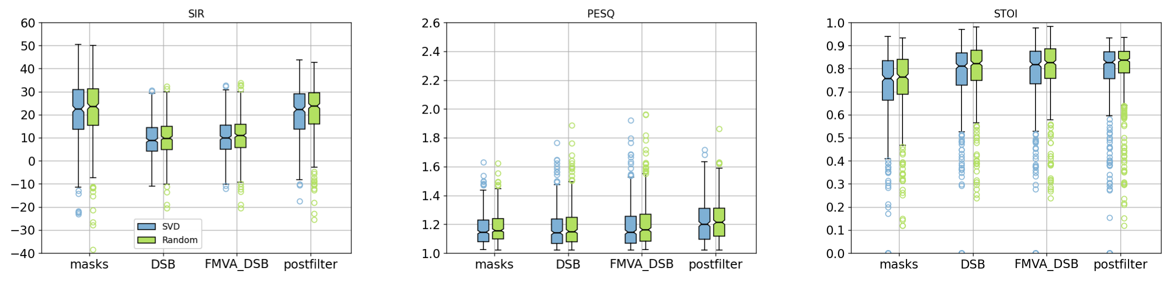
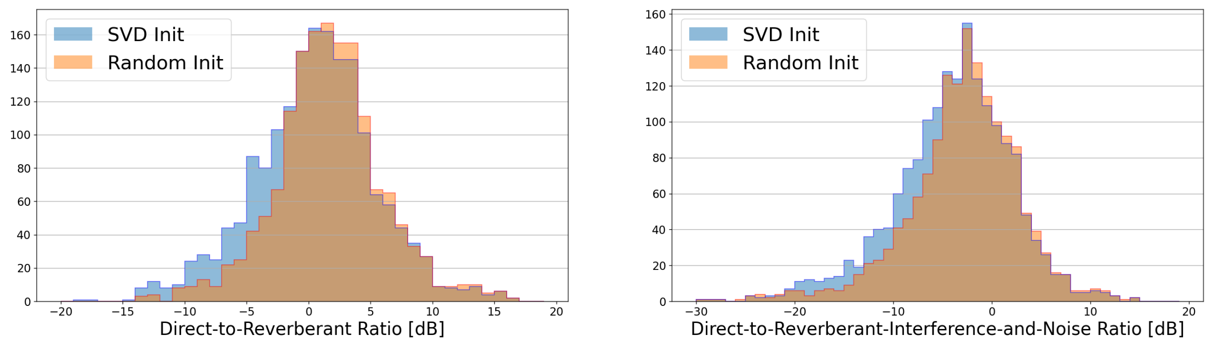


Figure 5.6: SIR, PESQ and STOI metrics for SVD and random initialization.



(a)

(b)

Figure 5.7: Comparing SVD and random initialization

5.3 Evaluation of time-domain coherence

Before comparing the time-domain coherence based clustering method to the other methods, it is important to determine which filter size performs the best.

5.3.1 Filter sizes

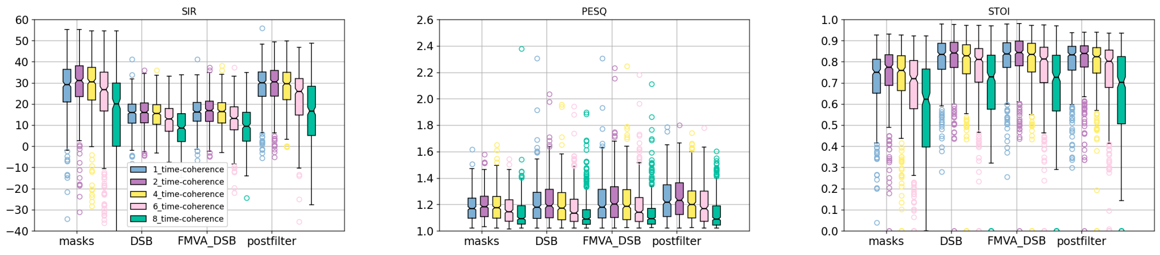


Figure 5.8: SIR, PESQ and STOI metrics for different filter sizes.

5.3.2 Comparison between MFCC, SpVer and time-domain coherence

The

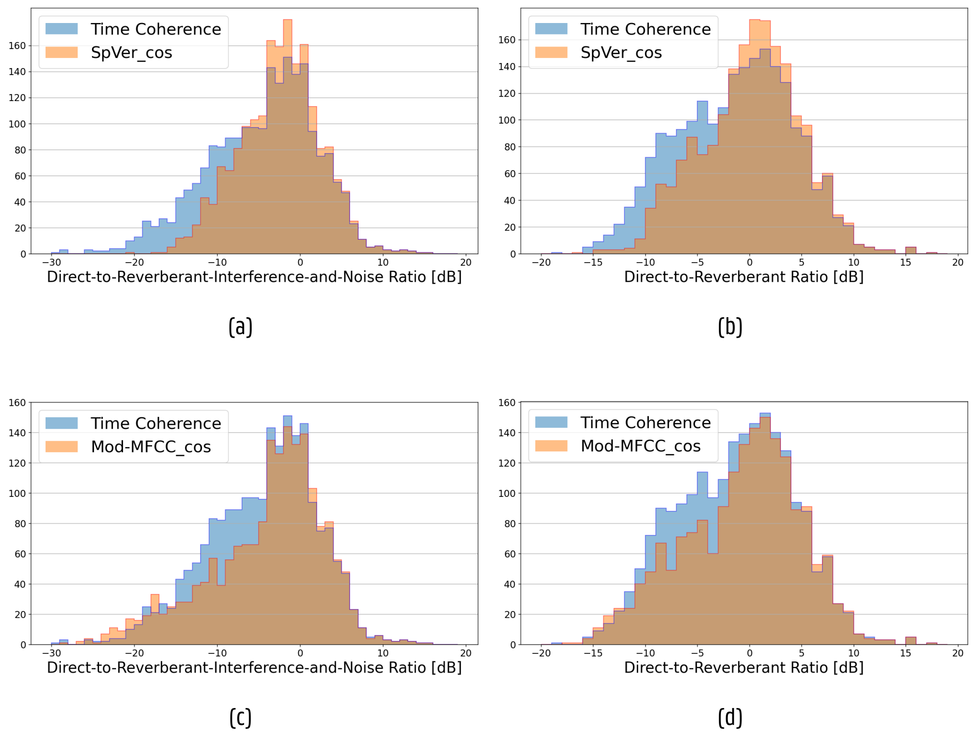


Figure 5.9: Comparing speaker verification and mod-MFCC with the time-domain coherence-based method

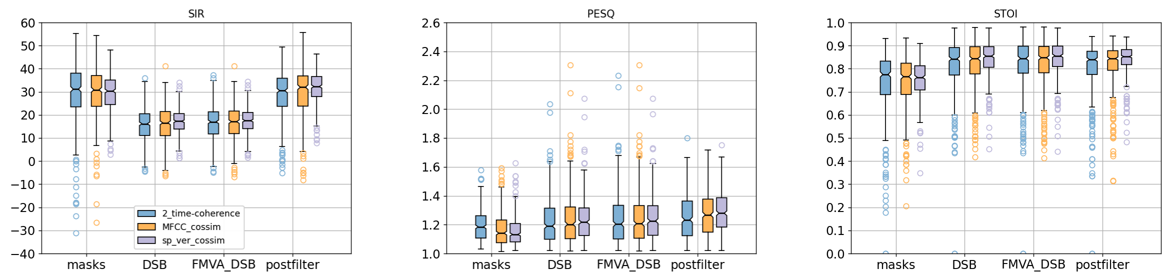
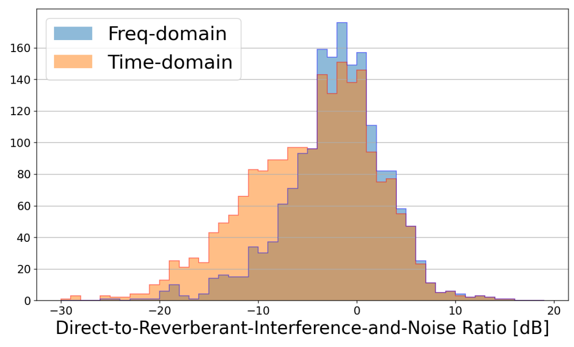


Figure 5.10: SIR, PESQ and STOI metrics for time-domain coherence.

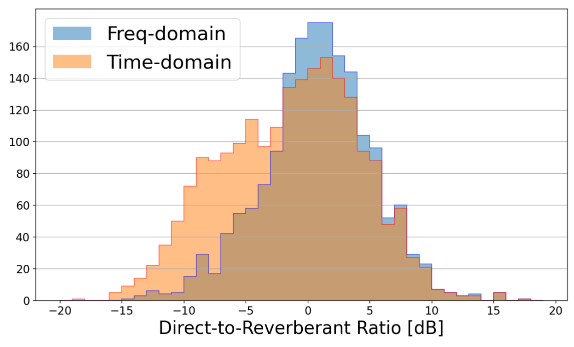
5.3.3 Comparison between time-domain and frequency-domain coherence

Time-domain coherence seems to perform worse than frequency-domain coherence on all metrics. This could be do to the fact that the current implementation uses stochastic gradient descent to estimate the filters h and h', which are used to estimate the coherence. As outlined in section 3.2.2, the method updates the filter on a sample-by-sample basis.

Consider the example of a signal containing two sources, but where one source becomes inactive in the middle of the signal. This would result in a bad clustering, since the coherence is calculated after iterating over all samples of the signal (or signal segment). Since the database used for evaluation contains such signals, it could be less suitable for the evaluation of this method. This could be a possible explanation for the lower performance of the time-domain coherence-based clustering method.



(a)



(b)

Figure 5.11: Comparing time and frequency-domain coherence

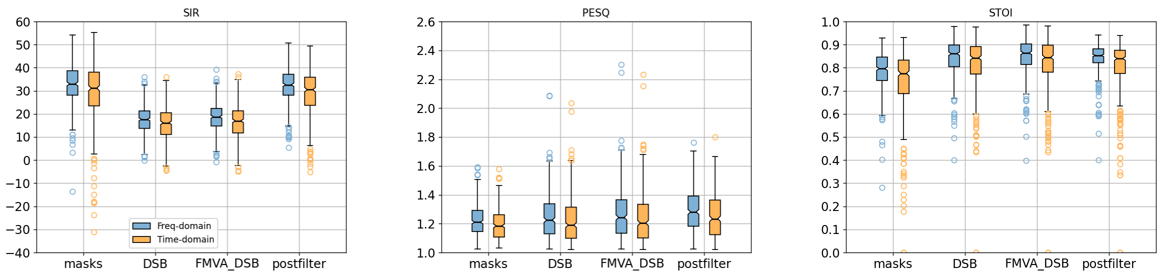


Figure 5.12: SIR, PESQ and STOI metrics for time-domain coherence against frequency-domain coherence.

5.4 LC3plus encoding evaluation

With the purpose of seeing how the clustering performance of the coherence-based methods degrade when performed on signals that have undergone LC3plus encoding, several tests were conducted. The clustering procedure was done on signals encoded with several different bitrates, as to see how the cluster quality would degrade when submitted to increasingly difficult circumstances.

(they are still running)

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] R. Martin and A. Nagathil, "Cepstral modulation ratio regression (cmrare) parameters for audio signal analysis and classification," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 321–324.

[4] 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.

[5] 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

[6] 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

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

[8] A. Holzapfel and Y. Stylianou, "Musical genre classification using nonnegative matrix factorization-based features," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 2, pp. 424–434, 2008.

[9] P. N. Garner, "Cepstral normalisation and the signal to noise ratio spectrum in automatic speech recognition," *Speech Communication*, vol. 53, no. 8, pp. 991–1001, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167639311000720

[10] 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.

[11] 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.

[12] 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.

[13] 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).

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

[15] 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.

[16] A. Nelus, R. Glitza, and R. Martin, “Estimation of microphone clusters in acoustic sensor networks using unsupervised federated learning,” in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 761–765.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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

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

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

[25] 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

[26] 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

[27] 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

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

[29] S. Gergen, R. Martin, and N. Madhu, “Source separation by feature-based clustering of microphones in ad hoc arrays,” in *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*, 2018, pp. 530–534.

[30] P. Welch, “The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms,” *IEEE Transactions on Audio and Electroacoustics*, vol. 15, no. 2, pp. 70–73, 1967.

[31] R. M. Sebastian Gergen, “Estimating Source Dominated Microphone Clusters in Ad-Hoc Microphone Arrays by Fuzzy Clustering in the Feature Space,” pp. 1–5, 2016.

[32] 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

[33] A. Varasteh, “Non negative matrix factorization implemented-in-python,” https://github.com/ahmadvh/Non-Negative-Matrix-factorization---Implemented-in-python, 2020.

[34] G. Dekkers, S. Lauwereins, B. Thoen, M. W. Adhana, H. Brouckxon, T. van Waterschoot, B. Vanrumste, M. Verhelst, and P. a. Karsmakers, “The SINS database for detection of daily activities in a home environment using an acoustic sensor network,” in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2017 Workshop (DCASE2017)*, November 2017, pp. 32–36.

[35] 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.

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

[37] E. Manilow, P. Seetharman, and J. Salamon, *Open Source Tools & Data for Music Source Separation*. https://source-separation.github.io/tutorial, 2020. [Online]. Available: https://source-separation.github.io/tutorial

[38] E. Vincent, R. Gribonval, and C. Fevotte, “Performance measurement in blind audio source separation,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 4, pp. 1462–1469, 2006.

[39] A. Rix, J. Beerends, M. Hollier, and A. Hekstra, “Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs,” in *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, vol. 2, 2001, pp. 749–752 vol.2.

[40] 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.

[41] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, “Librispeech: An asr corpus based on public domain audio books,” in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5206–5210.

[42] R. G. Alexandru Nelus and R. Martin, “Unsupervised Clustered Federated Learning in Complex Multi-source Acoustic Environments,” pp. 1–5, 2021.

Appendices

Appendix A