Two Sensor Array Beamforming Algorithm

for Android Smartphones

Authors	*Student numbers*
Mark Aarts	4015177
Hendrik Pries	4018036
Arjan Doff	4043359

TU Delft Delft University of Technology

July 4, 2012

Preface

This thesis is the result of a quarter of a year of research regarding multiple beamforming algorithms. Also, it is the conclusion to a three year bachelor course of Electrical Engineering at the Delft University of Technology.

First of all we would like to thank our supervisors Richard Hendriks and Richard Heusdens for their help with our project. Without them we would not have made such quick progress with testing and implementing the algorithms. Their recommendation to not start out with something too difficult, but work our way up from something easy made it a lot easier to grasp the concepts of beamforming.

Furthermore we would like to thank Ioan Lager for his efforts in regulating all the different bachelor projects. He made it possible for us all to learn and understand how it is to work in a team intensively.

This project was a great learning experience for us all, and will surely help in the coming years with our master courses.

Mark Aarts, Hendrik Pries and Arjan Doff July 4, 2012

Abstract

Nowadays, noise reduction in mobile communication is an increasingly important topic. This thesis discusses the possibilities of using beamforming with a two microphone array for smartphone applications. The goal of the project is to realise a beamforming algorithm for smartphones to improve on current Signal to Noise ratios attained by standard noise reduction methods.

By means of a literature study, three beamformers were investigated and presented: Delay and Sum (DAS), Minimum Variance Distortionless Response (MVDR) and the Generalized Sidelobe Canceller (GSC). Due to time constraints the DAS and MVDR were chosen for further simulations and testing, based on the terms of reference. Both beamformers were extensively simulated and tested in Matlab. The results were used to compare the two beamformers based on their white noise suppression, punctuated noise source suppression, frequency range and computational complexity.

Results showed that the white noise suppression for both beamformers is the same. For the suppression of punctuated noise sources however, the MVDR beamformer is more reliable, because it adapts its response to the main noise source. Furthermore, the MVDR beamformer performs better on the frequency spectrum of interest. The only advantage of the DAS beamformer over MVDR is a lower computational complexity.

It is concluded that beamforming with a two microphone array is possible. Considering the terms of reference, the MVDR beamformer is the better option of the simulated two for implementation on smartphones.

Contents

List of Figures

Nomenclature

$\hat{\mathbf{v}}$	The noise signal in the frequency domain
$\hat{\mathbf{x}}$	The received signal in the frequency domain
$\mathbf{B}$	The blocking matrix used for the GSC beamformer
$\mathbf{c}$	The constraint vector or look direction
$\mathbf{I}$	The identity matrix
$\mathbf{R}$	The autocorrelation matrix of the noise between the microphones
$\mathbf{w}$	The vector of weights, with which a signal is multiplied
$\mathbf{w}_c$	The constraint weights vector for the GSC beamformer
$\mathbf{w}_u$	The unconstrained weights vector for the GSC beamformer
$\cdot^H$	Hermites transpose, or complex conjugate transpose
$\cdot^T$	Transpose
Δu	The additional distance a signal need to travel, due to an angle between the microphones and the signal of the source, in [m]
$\hat{r}$	The signal of the source in the frequency domain
$\hat{s}$	The signal after filtering in the frequency domain
τ	The delay between the signals in [s]
θ	The angle in which the signals enter the microphones in [rad]
ξ	The phase shift of the second microphone dependent on F_t in [rad]
ζ	The phase shift of the second microphone dependent on F_c in [rad]
c	The speed of sound in [m/s]
d	The distance between the microphones in [m]
F_c	The center frequency of a frequency bin in [Hz]
F_s	The sampling frequency in [Hz]
F_t	The frequency of the original signal in [Hz]
h_k	The weights used for the Hann window
i	The number of the current time frame

k	The number of the current frequency bin
L	The amount of samples per time frame after zero padding
M	The amount of samples per time frame
N	The amount of microphones
n	The number of the current microphone
P	The output power of the beamformer in [W]
P_n	The noise power in [W]
P_s	The signal power in [W]
r	The signal of the source in the time domain
s	The signal after filtering in the time domain
v	The noise signal in the time domain
x	The received signal in the time domain

Chapter 1

Introduction

With the rising of smartphones, telecommunication becomes increasingly important in our day-to-day life. With this thesis we will try to improve on the clarity of this communication. Using the multiple microphones today's phones are equipped with, the amount of noise that is sent when calling with a smartphone can be reduced.

There are several ways in which noise can be introduced to the signal. Sources of noise are signals other than the signal of interest, e.g. background noise or echoes. When using a telephone, the signal of interest usually comes from only one direction. The microphones, however, cannot recognize this and are therefore susceptible to noise from all directions. Even though there are some techniques available [3], this noise cannot easily be removed from the desired source signal.

This thesis proposes to use an existing technique called beamforming to create a new way to reduce noise transmitted by smartphones. Studying the feasibility of beamforming with a two microphone array, the aim is to implement a noise reduction algorithm in an application. By using multiple microphones it is possible to achieve spatial selectivity, which is the possibility to select certain signals based on the angle of incidence. Beamforming has been used for a long time and has therefore been extensively researched [4]. Because of this there are a lot of beamforming techniques available [5].

Three beamforming techniques will be discussed and analysed and two beamformers have been simulated and tested. Considering design constraints like computational complexity, functionality and restrictions imposed by the smartphone, the goal of the thesis is to choose one of the beamformers and implement it in an application for the Android operating system.

This thesis is structured as follows. First the principle of beamforming will be explained in Chapter 2. After this, Chapter 3 will set the goals and restrictions imposed on the algorithm for smarphones which will then be used for the terms of reference. Chapter 4 will elaborate on and compare the possible solutions. Simulations and testing are covered in Chapter 5. Finally, in Chapter 6 our conclusions and results will be outlined. Additional information, such as the terms of reference, a schematic overview of the design process and the Matlab and Java codes can be found in the appendices at the back of the thesis.

Chapter 2

The Principle of Beamforming

This chapter explains the basics of beamforming and how it can be used to reduce noise or interfering sources coming from directions other then the direction of interest. After outlining the principle that makes beamforming work in this chapter, Chapter 4 will continue by introducing and explaining some existing algorithms.

When looking at the response of a normal microphone it can be seen that it is omnidirectional, meaning that the gain for signals from all directions is the same. The signal received by the microphone contains a combination of noise, interfering sources and the target source and often these three components come from different directions. Obviously the Signal-to-Noise Ratio (SNR) can be improved by focusing the response of the microphone in the direction of the target source (Figure 2.1). This focusing of the response to directions of interest is the essence of acoustical beamforming.

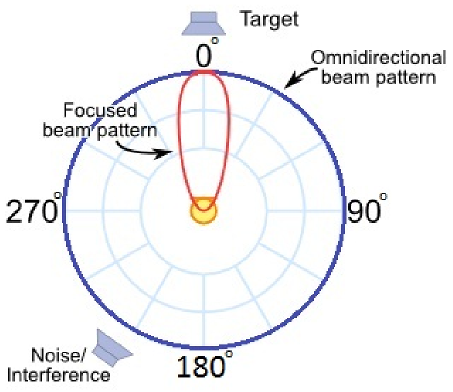


Figure 2.1: Response of omnidirectional and focused beam pattern [1]

To achieve spatial selectivity the beamformer needs to distinguish between the components from different directions of an incoming signal, in order to suppress the parts of the signal that are not coming from the target source(s). To analyse the incoming signal and differentiate between different angles of incidence, beamformers use an array of microphones.

To elaborate, let us consider a linear array of N microphones. When the microphone array picks up a signal coming from an angle other than $0°$ or $180°$, every consecutive microphone will experience an increased delay. This is because the signal entering from an angle needs to travel an additional distance, Δu, to the next microphone in the array. Δu is proportional to the distance between the microphones and the angle of incidence, see Figure 2.2.

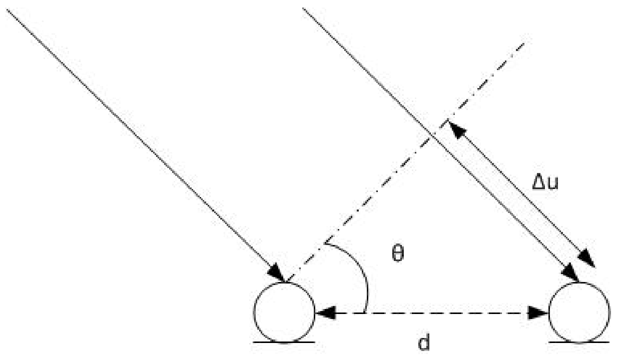


Figure 2.2: The signal travels an additional distance Δu to the next microphone

Since the microphones are fixed, the distance between them is fixed as well and we can relate the additional distance to the angle of incidence of the signal. Finally, because the speed of sound is also fixed, the time delay can be deduced from the direction of the signal by (2.1).

$$\tau = \frac{d}{c} sin(\theta). \tag{2.1}$$

Where τ is the time delay in [s] and d, c and θ are the distance between the microphones in [m], the speed of sound in [m/s] and the angle between the microphones in [rad], respectively.

For simplicity, we consider in this thesis the situation where the target source is positioned in the far- and free-field. In the far-field case the incoming wave front can be assumed to be planar, such that the estimation of Δu as shown in Figure 2.2 holds. In addition, due to the far-field assumption, a difference in damping between the different microphones can be neglected. In the situation that the sources are present in the near-field, special measures have to be taken to estimate the additional distance Δu and the difference in damping between the microphones. From here on we assume the far- and free-field case in order to demonstrate the principles of beamforming and show the potential.

Once the angle of the different components in the signal is known, the next task of the beamformer is to process the data in a way that the remaining signal consists only of the component from the selected direction. If this is done correctly, other noise sources and echoes will be filtered. To do this there are multiple methods known, three of which will be discussed in Chapter 4.

Chapter 3

Goals and Restrictions on the Beamforming Algorithm

In the previous chapter the principle of beamforming was explained. This chapter will discuss the restrictions imposed on- and the goals for the beamforming algorithm and its implementation. These will be used for the terms of reference, which can be found in Appendix A. In the following Chapters, 4 and 5, we will determine in what way the different types of beamforming are able to satisfy the goals and restrictions and the terms of reference.

Section 3.1 will cover the restrictions on the algorithm. After this, Section 3.2 will set the goals for the algorithm.

3.1 Restrictions of the algorithm

In this section we mention the main restrictions for the algorithm. These are obviously influenced by the smartphone that is used. For this thesis we used the Samsung Galaxy S2, which should allow beamforming due to its two microphones.

3.1.1 Restriction due to the environment

An important factor of the smartphone is the placement of the microphones. Firstly, beamforming is susceptible to spatial aliasing. This occurs when the distance between the microphones is larger than half the wavelength of the incoming wave [6], which leads to the following equation [7].

$$d < \frac{\lambda}{2} = \frac{c}{2F_t} \tag{3.1}$$

In this equation d is the distance between the two microphones, λ is the wavelength of the incoming signal, c is the speed of sound (343 m/s in air) and F_t is the frequency of the signal. From measuring we learned that the distance between the two microphones is $d = 0.12$ m, which sets the upper frequency boundary without spatial aliasing around 1400 Hz.

Another aspect influenced by the distance between the microphones is the perceived curve of the waveform. The sound waves from noise and target sources propagate spherically and as explained in Chapter 2 the computations will need planar waves for (2.1) to hold. Because of this, all waves will be assumed to be planar and the performance of the algorithm is likely to deteriorate as the far-field assumption becomes less plausible.

Furthermore the damping of the signal is influenced by the different paths to the microphones. As explained in Chapter 2 all computations assume a free-field for simplicity and performance of the algorithm will deteriorate if the microphones experience significant differences in damping.

Lastly, the target source is restricted to being stationary. The implemented algorithms will be static, which means that they are not automatically tuned to changing angles. If the direction of the target changes the algorithm will have to be reset.

3.1.2 Restrictions due to the smartphone

Apart from the microphone distances there are some other aspects of the smartphone that need to be considered. First of all, Android libraries offer three different sampling rates: 8 kHz, 16 kHz and 44.1 kHz. From these 8 kHz was chosen to mitigate computational complexity while preventing frequency aliasing. Frequency aliasing occurs when the frequency is larger than or equal to half the sample frequency [8], in this case, when the frequency is higher than 4 kHz. Because the voice frequency band is approximately up to 3.4 kHz [9], aliasing is avoided.

The second restriction of the smartphone is the limited power consumption. The battery of the Samsung Galaxy S2 has a capacity of 1650 mAh and works on a voltage level of 3.7 V [10]. This gives the phone a total energy of 6.11 Wh. To be used effectively the algorithm should last for at least three hours. This means the maximum power consumption can be 6.11 / 3 = 2.04 W. To determine what amount of power can be used by the algorithm, we need to know how much energy the phone uses in normal operation. Figure 3.1 shows the distribution of the power consumption when calling with a mobile phone. The figure shows that the total consumption in normal operation is approximately 1050 mW, meaning that our application should not consume more than 2.04 - 1.05 = 0.99 W. From [2] we know that the combination of the CPU and RAM will never draw such power, so a restriction on the power consumption will not be included in the terms of reference.

The third restriction is on the use of time-frames of 20 ms. This means we calculate the fast Fourier transform (FFT) of each time-frame after 20 ms. We wanted to choose reasonably high time-frame duration, such that we would have enough data to make accurate calculations. We did not want to have time-frames larger than 20 ms, because in that case the delay of the received signal would become recognizable for our brain and that would be annoying.

3.2 Goals of the algorithm

The goals set out in this section will also appear in the terms of reference in Appendix A.

The main goal is to create a beamforming algorithm which spatially filters noise significantly and can be used in a smartphone application. By means of simulations and tests we can then give a verdict on the feasibility of beamforming with 2 microphones, and check the validity of the assumptions and choices from the previous section.

One of the goals set for the filtering is to decrease white noise power by 3 dB, equivalent to reducing white noise by a factor two. This should be possible because two microphones are

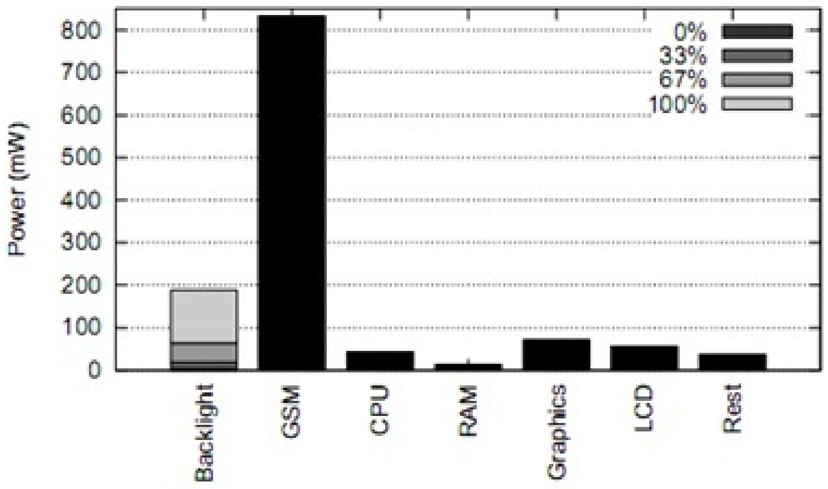


Figure 3.1: GSM phone call average power. Excluding backlight, the aggregate power is 1054.3 mW [2, p. 8].

used. The accumulated normalised signal power is:

$$P_{s_1+s_2} = P_{2s_1} = 4P_{s_1} \qquad \text{for } s_1 = s_2. \tag{3.2}$$

Here s_1 is the signal on microphone 1 after filtering, and s_2 is the signal on microphone 2 after filtering. Following from the far- and free-field assumption the signals are 100% correlated. Thus the signals will add, doubling the amplitude. The power of the summed signal will therefore be four times the original signal power. Taking the same steps for noise, the accumulated normalised noise power (for uncorrelated noise) is:

$$P_{n_1+n_2} = P_{n_1} + P_{n_2} = 2P_{n_1} \qquad \text{for } P_{n_1} = P_{n_2}. \tag{3.3}$$

In this formula n_1 is the noise on microphone 1 after filtering, and n_2 is the noise on microphone 2 after filtering. Since we are considering white noise, the power of both noise signals will be equal. Because the two signals are uncorrelated it follows that the power of the combined noise signal will be two times the noise power on microphone one. This results in a signal gain of 4, but a noise gain of only 2. So, in theory the goal of at least 3 dB noise reduction is fulfilled for the 2 microphone beamformer considering white noise.

As for other, punctuated, noise sources two microphones should be able to filter at least 1 stationary source located at a different location than the target source. The filter should be able to realise at least 10 dB, or a factor 10, reduction in noise power.

Finally the filter should be able to apply noise reduction on the range of human speech for the application to be used in speech applications on smartphone. The voice frequency band ranges from approximately 300 - 3400 Hz [9], so filtering has to be possible within this range.

Chapter 4

Three Possible Solutions for the Beamforming Algorithm

Having looked at the goals and restrictions of the algorithm, this chapter will propose three possible solutions. Firstly, Section 4.1 will look at the simplest possible solution, a Delay-and-Sum (DAS) beamformer. Secondly, Section 4.2 will investigate the possibilities of a Minimum Variance Distortionless Response (MVDR) beamformer. Thirdly, Section 4.3 will explore the Generalized Sidelobe Canceller (GSC) beamformer. And finally, in Section 4.4 a window function will be chosen by means of its spectral features. Chapter 5 will further discuss the advantages and disadvantages of the given solutions.

4.1 Solution 1: Delay-and-Sum beamforming

Given that the direction of the target source is specified, the task of forming the beam and filtering the incoming signal remains. One of the simplest and oldest ways to do this is with the Delay and Sum (DAS) beamformer.

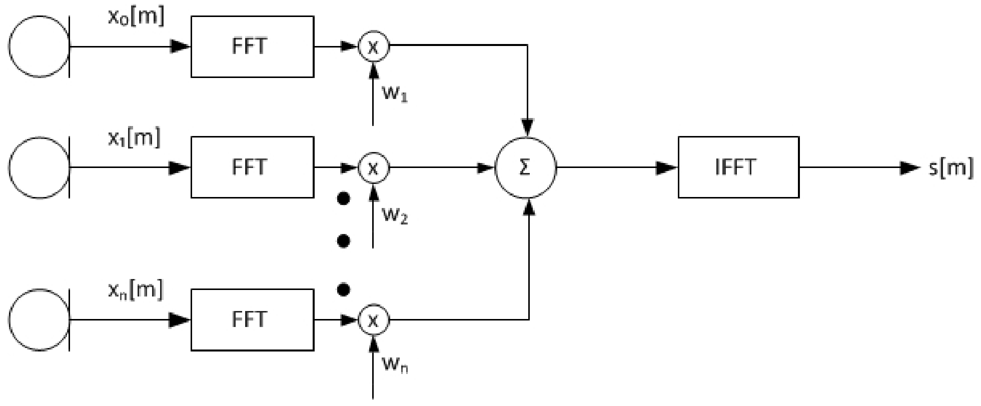


Figure 4.1: Conventional DAS beamformer

4.1.1 DAS beamformer in general

As the name states, this beamformer works by delaying the signals from certain microphones and summing them afterwards. In the previous section it was explained that the signal has a different and increasing delay for every microphone in the array. DAS basically reverses this process by delaying the outputs of all microphones except the last one, which is the reference microphone. By delaying the signals from each microphone appropriately those signals will add constructively if the time delay between them belongs to the selected angle, and will attenuate each other to a certain extend when they do not. Since the delay between the microphones usually consists of a non-integer number of time-samples, the delay is done by a phase change in the Fourier domain rather than in the time domain.

Figure 4.1 shows a conventional DAS beamformer for an N-microphone array. The figure shows that the array produces a signal for each different microphone, x_n. After transforming these signals to the frequency domain the signals are delayed and their amplitude is weighted by multiplying with the appropriate weighting factor $w_1 = w_2 = w_n = \frac{1}{N}$, where N is the number of microphones. The weighted signals are then summed and transformed back to the time domain to create the output signal.

4.1.2 Two sensor DAS beamforming algorithm

The algorithm to be used will only have $N = 2$ microphones available to create a beam, where the 2 microphones are separated by a distance d, see Figure 4.2.

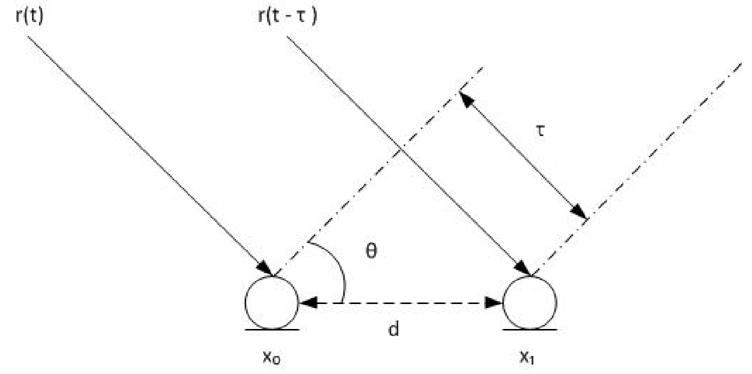


Figure 4.2: The construction of the microphones

Assume the incoming signal at microphone n is represented by:

$$x_n(t) = r(t - n\tau) + v_n(t). \tag{4.1}$$

Here $r(t - n\tau)$ represents the original signal delayed with $n\tau$ and $v_n(t)$ represents the noise at microphone n. From (2.1) it can be seen τ is defined as:

$$\tau = \frac{d}{c} sin(\theta). \tag{4.2}$$

With d the distance between the two microphones in [m], c the speed of sound in [m/s] and θ the angle of the incoming signal in [rad]. As set out in the terms of reference (Appendix A), sampling is divided in i time-frames with a duration of 0.02 seconds, or 20 milliseconds. Since our sampling frequency is restricted to F_s of 8 kHz, sampling of x_n results in $M = 0.02 \cdot 8000 = 160$ samples in one time frame.

For the discrete-to-discrete Fourier transform (DDFT) of a signal it is preferred if the length of the signal is an integer power of 2, resulting in a fast Fourier transform (FFT) [11]. Therefore, the samples per time frame are padded with zeros to $L = 256$ samples, before performing the FFT. When a signal is transformed to the frequency domain it consists of L frequency bins. These bins each cover $\frac{F_s}{L}$ Hz. The following equations show how the DAS beamformer reconstructs the signal. Note that variables in the frequency domain are denoted with a $\hat{\cdot}$.

The center frequency of frequency-bin k is defined as:

$$F_c[k] = k\frac{F_s}{L}. \tag{4.3}$$

The estimated signal for frequency-bin k for one time frame is defined as:

$$\hat{s}[k] = \mathbf{w}_k^H \hat{\mathbf{x}}[k]. \tag{4.4}$$

Where $\mathbf{w}$ is defined as the weights for both microphones calculated by:

$$\mathbf{w}_k = \frac{1}{2}[1, e^{-j\zeta[k]}]^T. \tag{4.5}$$

The symbol $\zeta[k]$ denotes the phase shift of the second microphone at frequency-bin k calculated as follows:

$$\zeta[k] = \frac{2\pi F_c[k] d}{c} sin(\theta). \tag{4.6}$$

When looking at (4.4), it is clear $\hat{\mathbf{x}}[k]$ does not need to be calculated since it is received by the 2 microphones. Though a FFT still has to be performed on the incoming signals.

It can be simulated by:

$$\hat{\mathbf{x}}[k] = [1, e^{-j\xi}]^T \hat{r}[k] + \hat{\mathbf{v}}[k]. \tag{4.7}$$

Where the $\cdot^T$ denotes the transpose and ξ is the phase shift of the second microphone, see (4.8). $\hat{r}[k]$ and $\hat{\mathbf{v}}[k]$ denote the original signal and the noise signal respectively. Both signals are in the frequency domain.

$$\xi = \frac{2\pi F_t d}{c} sin(\theta). \tag{4.8}$$

Here F_t is defined as the frequency of the signal. Note that these calculations are for just one frame i of 20 ms. E.g., to reconstruct a signal of one second this procedure has to be repeated fifty times.

4.2 Solution 2: Minimum Variance Distortionless Response

The relative simplicity of the DAS beamformer provides both advantages with respect to the computational complexity of the algorithm and disadvantages in the form of functionality. In this section another beamformer, the Minimum Variance Distortionless Response (MVDR) beamformer, also known as a superdirective beamformer, will be introduced.

Considering again the goal of the beamformer, focusing the response of the array to a certain direction, the MVDR beamformer takes a different approach than the DAS. Where the latter steers to a certain direction by enhancing the desired signal, MVDR firstly tries to minimize the signals from interferers and noise. MVDR primarily aims to minimize the total output power. Since we would like to keep a clear overview of all calculations in this section, the frame index *i* and frequency-bin *k* have been removed from the equations. Though it is important to know that these calculations have to be done for every time frame *i* and frequency-bin *k*.

The minimum of the average output power of the beamformer per time frame *i* and per frequency-bin *k* is given by [12]:

$$min\{P\} = min\{\mathbf{w}^H\mathbf{R}\mathbf{w}\}, \tag{4.9}$$

where $\cdot^H$ denotes the Hermitian transpose, $\mathbf{w}$ is the weight vector and $\mathbf{R}$ is the autocorrelation matrix of the noise. Thus, it is required to make an estimate of the noise signal before performing the calculations. One way to obtain this signal is to use the fact that the first few time frames are likely to consist only of noise. In a speech application for example, absence of the speech signal during a few start-up time frames enables the algorithm to get an estimate of the noise signal. With this estimation the autocorrelation matrix can be calculated with:

$$\mathbf{R} = \begin{bmatrix} |\hat{\mathbf{v}}_1|^2 & \hat{\mathbf{v}}_1\hat{\mathbf{v}}_2^H \\ \hat{\mathbf{v}}_1^H\hat{\mathbf{v}}_2 & |\hat{\mathbf{v}}_2|^2 \end{bmatrix}, \tag{4.10}$$

where $\hat{\mathbf{v}}_n$ denotes the noise received in the frequency domain at the *n*th microphone.

From (4.9) it can easily be seen that P is minimized if $\mathbf{w}$ is the zero vector $\mathbf{0}$. Despite satisfying the primary aim, this trivial solution will not steer the response in the wanted direction. The MVDR beamformer obviously requires a constraint: Unity gain should be maintained in the target direction. Since the delay in the signal from the different sensors can be written as $\mathbf{c} = [1, e^{-j\zeta}]^T$ this constraint can be defined by (4.11).

$$\mathbf{c}^H\mathbf{w} = 1. \tag{4.11}$$

When (4.9) is combined with (4.11) we end up with the final solution for $\mathbf{w}$ [12–15]:

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{c}(\mathbf{c}^H\mathbf{R}^{-1}\mathbf{c})^{-1}. \tag{4.12}$$

It can be noted that if $\mathbf{R}$ is the identity matrix $\mathbf{I}$, the weights $\mathbf{w}$ would simplify to the same weights as with a DAS beamformer, see (4.13). $\mathbf{R}$ simplified to $\mathbf{I}$ would correspond to the correlation between the 2 microphones being zero, though this does not imply that the 2 microphones are independent of each other [16].

$$\mathbf{w} = \mathbf{I}^{-1}\mathbf{c}(\mathbf{c}^H\mathbf{I}^{-1}\mathbf{c})^{-1} = \mathbf{c}(\mathbf{c}^H\mathbf{c})^{-1} = \mathbf{c}(||\mathbf{c}||^2)^{-1} = \frac{\mathbf{c}}{2} \tag{4.13}$$

As can be seen from the equations, in order to calculate the weights, the inverse of the autocorrelation needs to be calculated for every time frame and every frequency bin. Apart from being computationally intensive, this may not always be possible, since we are not dealing with a deterministic signal [12]. To elaborate, when we are dealing with a non-deterministic signal (a random signal), the autocorrelation matrix *R* could be singular. Which, when calculating the inverse of a singular matrix, would result in dividing by zero.

In Chapter 5 we will further discuss the possibilities of a MVDR beamformer.

4.3 Solution 3: Generalized Sidelobe Canceller

Even though a MVDR beamformer has a better performance than a DAS beamformer, the intensive computation of the weight factors is a disadvantage. The Generalized Sidelobe Canceller (GSC) tries to simplify these computations to calculate the weight factors more easily.

Computational complexity in MVDR is due to the fact that the constrained minimization problem from (4.9) has to be solved for every computation of the weight factors. To simplify this calculation, the GSC beamformer works with a constrained and an unconstrained part.

When a signal **x** enters the GSC beamformer, it follows two different paths, see Figure 4.3. The top path includes the constrained part of the beamformer, where the signal is filtered with the weight vector $\mathbf{w}_c$ using the DAS algorithm. The bottom path consists of a blocking matrix **B** and an unconstrained weight vector $\mathbf{w_u}$.

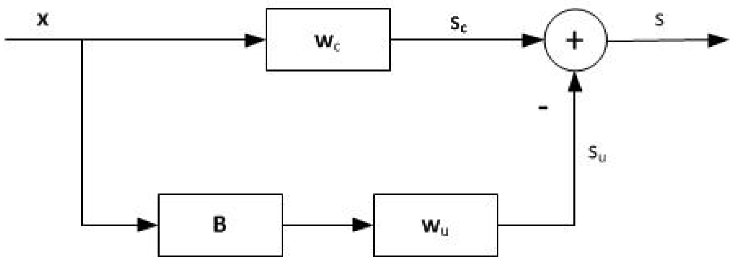


Figure 4.3: Generalized Sidelobe Canceller: a schematic overview

The GSC aims to use the blocking matrix to block the desired part from the signal **x**, so that the remaining signal in the bottom path $\mathbf{s_u}$ consists of only the noise and interference. If the approximation of the noise signal is done correctly it can be removed from the top path's signal by subtraction. The resulting signal should form a beam in the wanted direction due to the unconstrained part, and free from noise and interference due to the constrained part.

Once again the following calculations are done for every time frame *i* and every frequency bin *k* [12, 17].

$$\mathbf{w} = \mathbf{w}_c - \mathbf{B}\mathbf{w}_u. \tag{4.14}$$

Note that $\mathbf{w}_u$ is independent of the frame index. $\mathbf{w}_c$ is easily calculated, because it is the same as for the DAS weights, see Section 4.1. We use the same constraint vector as we did in Section 4.2 to define the blocking matrix [18]:

$$\mathbf{c}^H\mathbf{B} = \mathbf{0}. \tag{4.15}$$

Again a minimization problem needs to be solved:

$$min\{P\} = min\{(\mathbf{w}_c - \mathbf{B}\mathbf{w}_u)^H\mathbf{R}(\mathbf{w}_c - \mathbf{B}\mathbf{w}_u)\}. \tag{4.16}$$

When setting the derivative equal to zero we end up with the final solution for $\mathbf{w}_c$ [12, 19]:

$$\mathbf{w}_u = (\mathbf{B}^H\mathbf{R}\mathbf{B})^{-1}\mathbf{B}^H\mathbf{R}\mathbf{w}_c. \tag{4.17}$$

When we want to construct the blocking matrix **B** we can use the matrix proposed by Griffith and Jim [20], which after it is phase-aligned looks like this [12, 21, 22]:

$$\mathbf{B} = \begin{bmatrix} -1 \\ e^{-j\zeta} \end{bmatrix}. \tag{4.18}$$

A disadvantage of the GSC occurs when the blocking matrix **B** is not chosen adequately. This can result in an effect called signal leakage [21,23,24] and occurs when **B** does not completely block the speech signal. Parts of the desired signal remain in the bottom path and are then subtracted from the original signal.

4.4 Window functions

An overlap of the time frames is often desired to increase the time resolution of a signal. Also, because the edges of the time frame are sharp it is desirable to smooth the transitions between frames. For these reasons a window is used before performing a FFT on a signal. There are multiple windows available with the most common being the rectangular window. In this section other windows such as the Hann, Blackman and Kaiser windows are explored because they possess different, usually better spectral features [25,26]. Figure 4.4 shows a time domain plot for all four different windows.

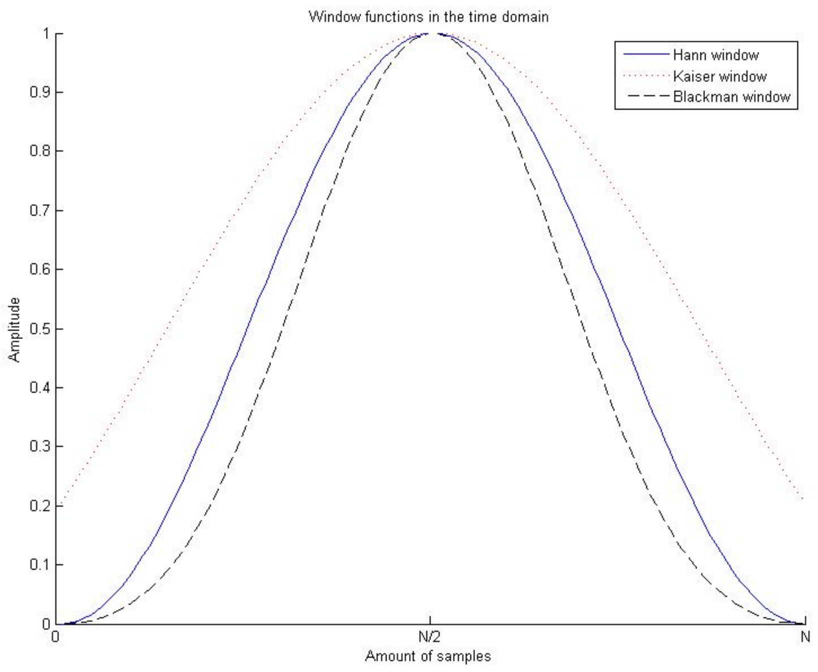


Figure 4.4: The Hann, Blackman and Kaiser windows in the time domain

When transformed to the frequency domain, these windows have different spectral features, see Figure 4.5.

All of these windows could be a good choice for our algorithm. Though we do not need a small main lobe, nor very low sidelobes. Also, we do not want to increase the complexity of our algorithm. Thus we chose the Hann window, whose weights are defined as [27]:

$$h[k] = \frac{1 + cos\left(\frac{2\pi k}{M}\right)}{2}, 0 \leq k \leq M - 1. \tag{4.19}$$

Where k is the current sample, running from 1 to the length of a time frame, that is M.

A useful characteristic of a Hann window is that when you add either the left or the right

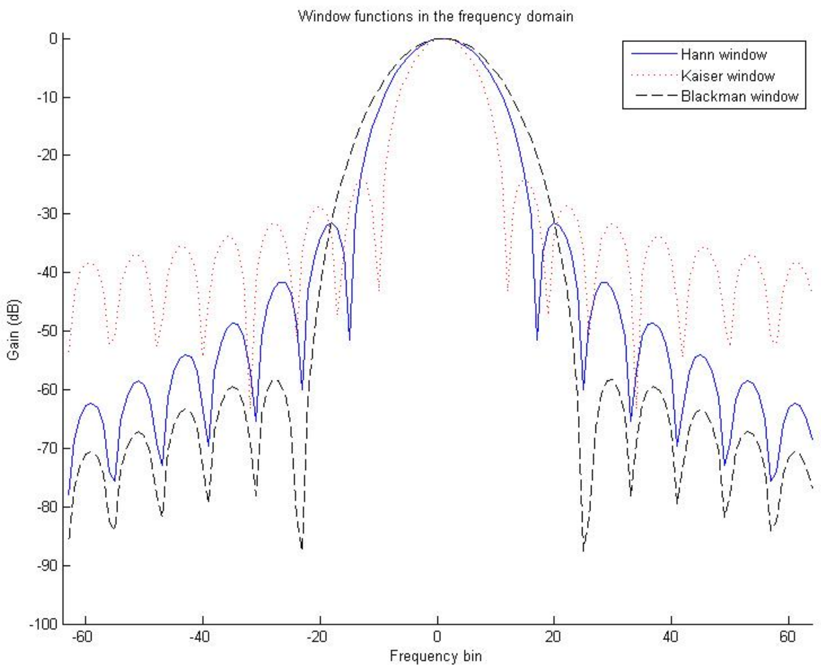


Figure 4.5: The Hann, Blackman and Kaiser windows in the frequency domain

side with another Hann window, the result will be a constant 1. Furthermore Hann windows are often used, as they have they have a good frequency resolution, a low spectral leakage and a sufficient amplitude accuracy [25]. Spectral leakage occurs when a time-variant operation is done on the received signal [28]. The operation in this case is multiplying by the Hann window. For a Hann window the disadvantages of spectral leakage do not outweigh the advantages of the increase in time resolution.

Chapter 5

Results: DAS and MVDR Simulations

To find the best algorithm for implementation on a smartphone, we simulated two of the techniques mentioned in Chapter 4 in Matlab. Time constraints forced us to abandon the GSC beamformer. MVDR was chosen over GSC because they are expected to have similar results, but simulations of the MVDR are easier to realise. Therefore the considerations of the proposed solutions in Section 5.3 we will only include the results of the Delay-And-Sum and Minimum Variance Distortionless Response beamformers.

In Section 5.1 the tests and results from Matlab for Delay and Sum beamforming will be covered. In Section 5.2 we will do the same for Minimum Variance Distortionless Response beamforming. Finally, in Section 5.3 the simulation results of both beamformers are compared by means of the criteria from Chapter 3.

5.1 Delay-And-Sum beamforming: Tests and results

This section covers the results of the Matlab simulations of the Delay-And-Sum beamformer. The beamformers were tested for a single noise and target source from different directions with white Gaussian noise present.

Because of the restrictions of the telephone, we used a sampling frequency of 8 kHz, a distance between the two microphones of 0.12 m and a total sample time of 1 second with time-frames of 20 ms. To clarify the results of the simulations in the time-domain, the sample rate was set to 80 kHz to increase the number of data points in the reconstructed signal. For the first test we used a 1500 Hz sinusoid wave coming from the desired direction, 180°, and a 2500 Hz sinusoid noise wave with a four times bigger amplitude coming from another direction at an angle 40°. We also added some white Gaussian noise to the microphones. In the simulation these angles were used to impose a different time-delay on both signals.

In Figure 5.1 the Delay-And-Sum filter is displayed in a polar plot. The beamformer used the desired signal to construct the filter, but it did not use the properties of the undesired signal. The figure shows that for an angle of 180° and a frequency of 1500 Hz the DAS beamformer has a response of '1'. The undesired wave with frequency 2500 Hz will be almost fully suppressed. It should be noted that because the DAS beamformer is tuned to the desired signal, for a frequency of 2500 Hz an angle of 40° is the optimum case. If the same 2500 Hz sine came from a different direction, for example 90°, the noise power would have lost less than 3 dB. Figure 5.1

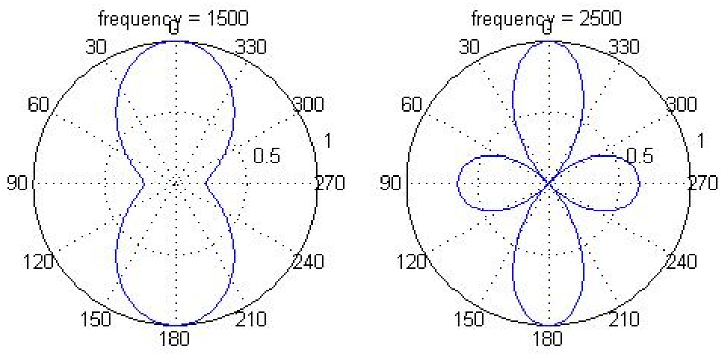


Figure 5.1: DAS: Polar plot of the response for a sine signal with frequency of 1500 Hz tuned to the direction 180° and the response for 2500 Hz with the noise signal coming from direction 40°.

shows that the angle of incidence of the noise is very important for the possibility of significant suppression.

The result for the frequency spectrum is shown in the top plot of Figure 5.2. To smooth and clarify the results, we took the mean of all time-frames in this figure. This has no influence on the frequency peaks, because they represent pure sine waves, which are present during the entire simulation. But for the Gaussian noise this means the spectrum will be flatter and so you can see the power reduction of the noise more clear. The blue line shows the spectrum of both incoming sine signals on the reference microphone, as well as some white Gaussian noise. As expected the undesired sine has an amplitude four times larger than the amplitude of the desired sine.

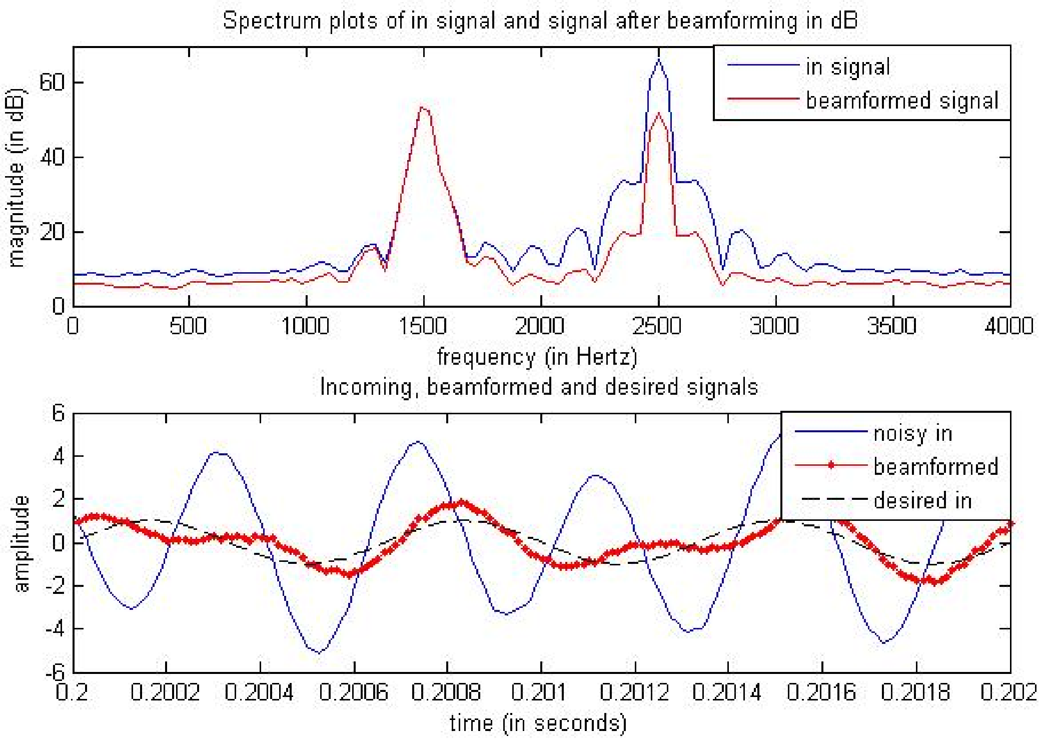


Figure 5.2: DAS: power spectrum plot and signal reconstruction of a sine signal with a frequency of 1500 Hz and a sine noise signal with a frequency of 2500 Hz.

The red line shows the frequency spectrum after beamforming. Figure 5.2 shows a reduction of the unwanted sine at 2500 Hz of almost 20 dB and a reduction of the white Gaussian noise of approximately 3 dB as expected (see Chapter 3). We also have to realise that this is the optimal case because of the angle 40°. Since the beamformer was steered at the direction of the 1500 Hz signal in this simulation, the frequency spectrum shows that the Delay-And-Sum beamformer behaved as expected and filtered the signal from the unwanted direction.

The reconstruction of the same signals in the time-domain is shown in bottom plot of Figure 5.2. As expected it shows that the reconstructed signal (red line) moved from the original signal (blue line) towards the signal from the desired direction (green line). The figure shows that this is indeed the case and the Delay-And-Sum beamformer made a signal similar to the desired sine.

If we choose frequencies around 1400 Hz and angles for the noise which follow from the polar plot, similar results are produced and the reconstructed signal seems to be almost what we would have in an ideal case. For very low or high frequencies however, or for angles that are not filtered much, the results are not always satisfying.

To show what happens if we choose incoming signals of those frequencies, we have simulated an incoming sinusoid signal of 300 Hz from the wanted direction and an unwanted sinusoid signal of 600 Hz.

In Figure 5.3 we see the polar plot for this case. Both angles are chosen the same as in the first test. In the polar plot we see there is almost no suppression at all.

In the top plot of Figure 5.4 we see the frequency spectrum of the two test signals at 300 Hz (wanted signal) and 600 Hz (unwanted signal). The peak at 600 Hz, which should be suppressed significantly, is almost the same as the original in the reconstructed spectrum. This result is conform with what we expected from the polar plots.

In the bottom plot of Figure 5.4 the time-domain signals and the reconstructed signal are shown. As expected, the filtered signal is almost the same as the total incoming signal at the reference microphone. The unwanted sine is hardly suppressed and the beamformer did not reach the goals set out in 3.

We can conclude that the DAS beamformer works well for some frequencies and only for certain angles, but otherwise it has a poor performance. We will discuss this further in Chapter 6.

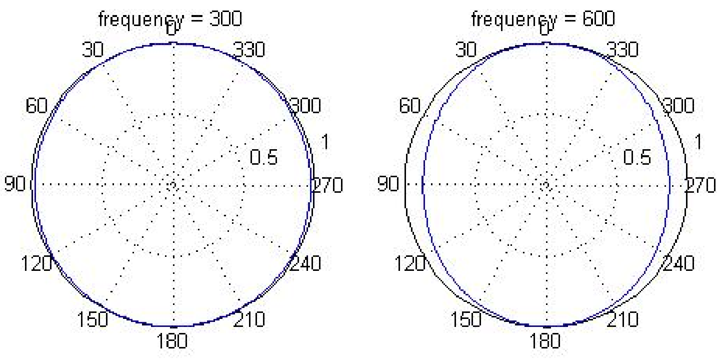


Figure 5.3: DAS: Polar plot of the response for a sine signal with frequency of 300 Hz tuned to the direction 180° and the response for 600 Hz with the noise signal coming from direction 40°.

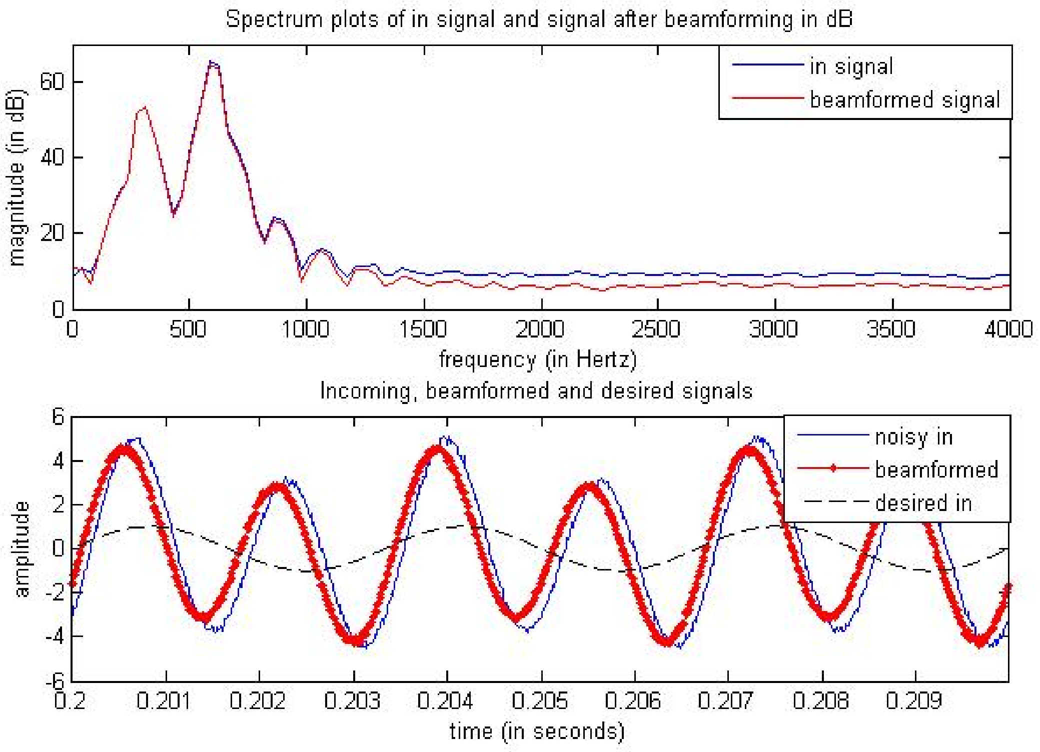


Figure 5.4: DAS: power spectrum plot and signal reconstruction of a sine signal with a frequency of 300 Hz and a sine noise signal with a frequency of 600 Hz.

5.2 Minimum Variance Distortionless Response beamforming: Tests and results

In this section the results of the Matlab simulations for the MVDR beamformer are treated. The MVDR beamformer was tested for the same frequencies and angles as the DAS, as to allow them to be compared.

When simulating the MVDR beamformer, the algorithm needs to make an estimation of the autocorrelation matrix of the noise (see Chapter 4). To do this, we let the algorithm initialize for 0.2 seconds, or 10 time-frames. For each of these ten frames the autocorrelation matrix is calculated after which the mean of these matrices is used as an estimation for the autocorrelation matrix. After initialization the signal source is added, which can be compared to the user suddenly starting to speak. For these tests we used the same signals as for the DAS simulations, sinusoid wave of 1500 Hz coming from the wanted direction with an angle of 180°; and for the noise source a sinusoid wave with frequency 2500 Hz and four times the amplitude of the wanted signal from an angle of 40°.

The polar plot for this simulation is shown in Figure 5.5. It shows that the maximal response for signals with frequency 1500 Hz is larger than ’1’, namely 1.0254. At an angle of 180° we see the response is ’1’, as expected from the constraint set out in Chapter 4. For frequencies of 2500 Hz, Figure 5.5 shows that signals coming from an angle of 40° are greatly suppressed. This can also be explained using Chapter 4. The MVDR beamformer tries to suppress as much of the signal as possible while considering its constraint, so for the loud sine noise source it will try to lay a ’0’-response in that direction, which is shown in the polar plot.

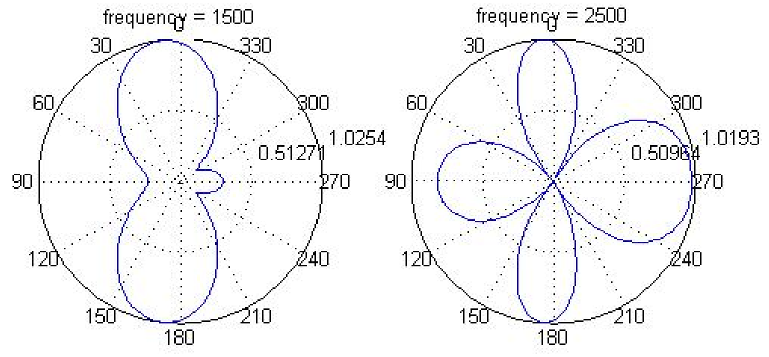


Figure 5.5: MVDR: Polar plot of the response for a sine signal with frequency of 1500 Hz tuned to the direction 180° and the response for 2500 Hz with the noise signal coming from direction 40°.

The result for the frequency spectrum is shown in the top plot of Figure 5.6. For this figure the mean of all time-frames during the simulation was also taken, as to create a flat frequency spectrum for the white noise. The blue line shows the spectrum of both incoming signals on the reference microphone. As expected the undesired sine has an amplitude four times larger than the amplitude of the desired sine, and the frequencies of the signals are 1500 and 2500 Hz respectively. The reconstructed spectrum plot after beamforming is shown with the red line. In this plot amplitude of the white Gaussian noise is shown to be reduced with approximately 3

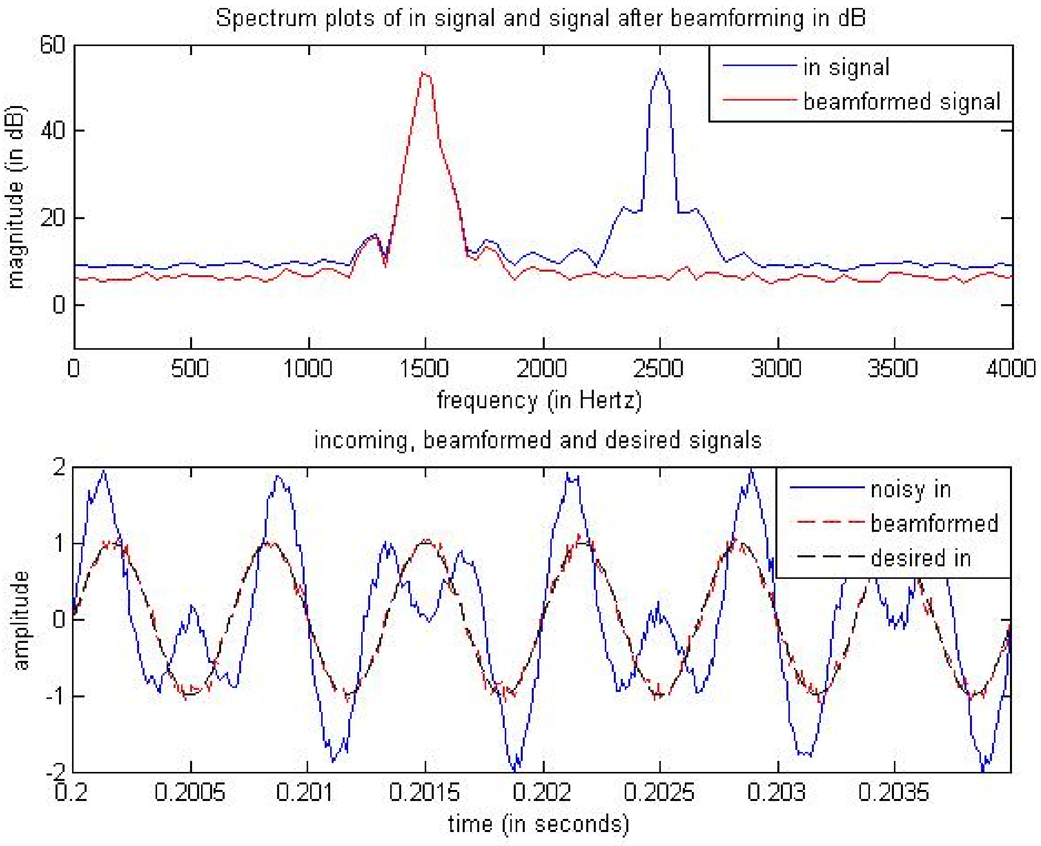


Figure 5.6: MVDR: power spectrum plot and signal reconstruction of a sine signal with a frequency of 1500 Hz and a sine noise signal with a frequency of 2500 Hz.

dB, as expected from Chapter 3 and as was also shown for the DAS beamformer. This figure also shows clearly that the signal at 1500 Hz is not suppressed, while the power of the 2500 Hz signal is reduced by over 30 dB and can hardly be distinguished from the white noise. Because the beamformer was steered at the direction of the 1500 Hz signal in this simulation this frequency spectrum shows that the MVDR beamformer behaved as expected, filtering the white noise by 3 dB and the punctuated noise source almost perfectly. It should be noted that the MVDR can only place one '0'-response with 2 microphones, so the algorithm can not filter two noise sources operating at the same frequency.

The reconstruction of the signals in the time-domain is shown in the bottom plot of Figure 5.6. As expected it can be seen that the reconstructed signal (red line) follows the desired signal is filtered almost completely.

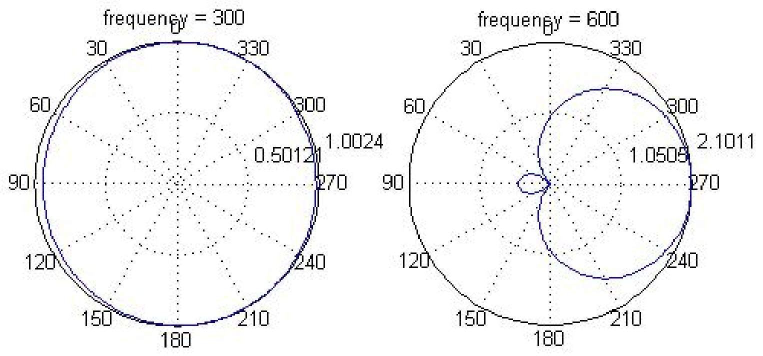


Figure 5.7: MVDR: Polar plot of the response for a sine signal with frequency of 300 Hz tuned to the direction 180° and the response for 600 Hz with the noise signal coming from direction 40°.

To compare the MVDR with DAS at lower frequencies, Figure 5.7 shows the polar plot for a desired 300 Hz sine and an undesired 600 Hz sine. All angles of incidence are kept the same. The figure again shows that the beam at the desired frequency is formed to place a '1'-response at 180° and a '0'-response at 40° for the frequency of the noise. Even though the response for those directions is correct it should be noted that the total output noise power of the beams (dependent of the area of the beam) is larger than at the higher frequencies.

In the top plot of Figure 5.8 the spectrum plot of the low frequency test is showed. In the reconstruction of the spectrum plot (the red line) a white noise suppression of 3 dB is apparent again. It can also be seen that the sine at f = 600 Hz is suppressed with almost 30 dB. In the time-domain reconstruction we see the beamformed signal follows the desired signal very precisely again. Comparing Figure 5.8 with Figure 5.4 it can be seen that the result at 300 Hz improves a lot on the simulations with the DAS beamformer.

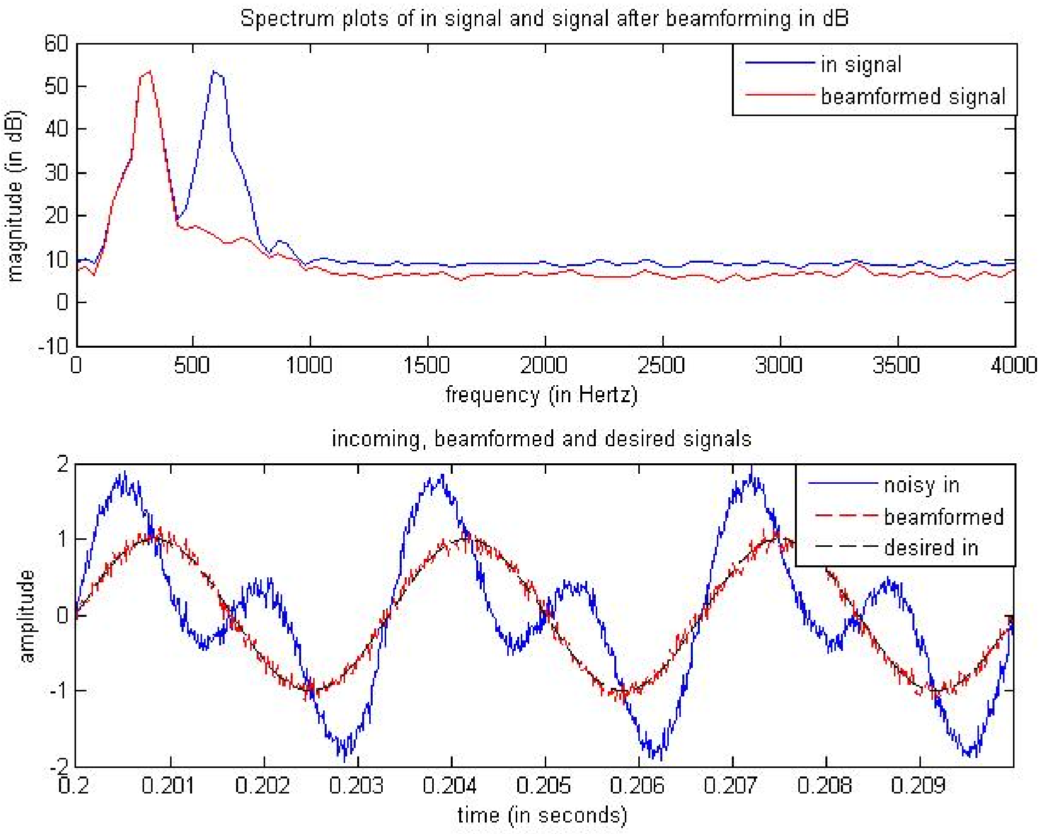


Figure 5.8: MVDR: power spectrum plot and signal reconstruction of a sine signal with a frequency of 300 Hz and a sine noise signal with a frequency of 600 Hz.

5.3 Considerations of the Proposed Solutions

To compare the tested algorithms, the goals for the algorithm set in Chapter 3 have to be considered. This section will briefly list them again after which the algorithms will be compared using the results of the tests from Section 5.1 and 5.2.

The goals for the algorithm were:

- Reduce white background noise with at least 3 dB.
- Ability to reduce a single, stationary and punctuated noise source with at least 10 dB.
- An adequate response on the whole spectrum of the desired frequency band, which is between 300 and 3400 Hz.

Comparing both algorithms for above goals yields:

As explained in the previous section both the DAS and MVDR beamformer reduced white background noise with 3 dB and achieved the set goal (see spectrum plots in previous sections).

In the simulations both beamformers showed at least 10 dB noise reduction for the punctuated noise source at high frequencies. It was noted however, that due to the fact that the DAS beamformer does not adjust its response to the noise source the suppression will be less for different angles (Figure 5.1).

Apart from this, it was seen that the DAS fails to suppress the noise source at lower frequencies (Figure 5.4), while the MVDR algorithm is still able to reduce the noise source by more than 10 dB (Figure 5.8).

Testing showed that both algorithms have varying performance for different frequencies. To test the goal set for the frequency range Figure 5.9 and 5.10 display polar plots for both beamformers on the frequency band from 300 to 3100 Hz in steps of 400 Hz. For every frequency, Gaussian noise and target and noise sources at 180° and 40° respectively are present, the same as in previous tests.

The plots show that both beamformers are able to place a '1'-response in the target direction for the whole spectrum. However, Figure 5.9 shows that due to the width of the beam at lower frequencies the DAS beamformer has difficulty suppressing the noise up to 1500 Hz. The MVDR on the other hand is able to place a '0'-response in the target direction for the whole spectrum.

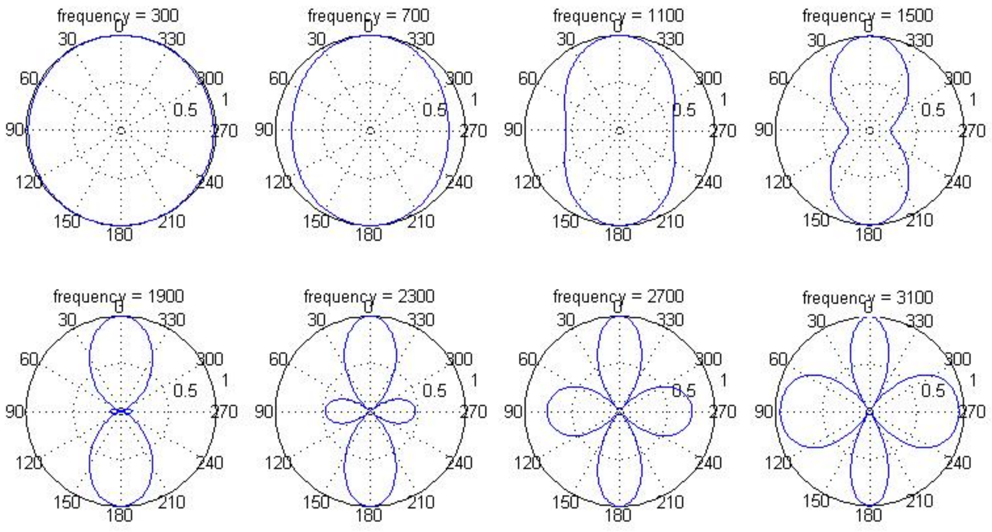


Figure 5.9: DAS: Polar plots for the frequency range 300-3100 Hz

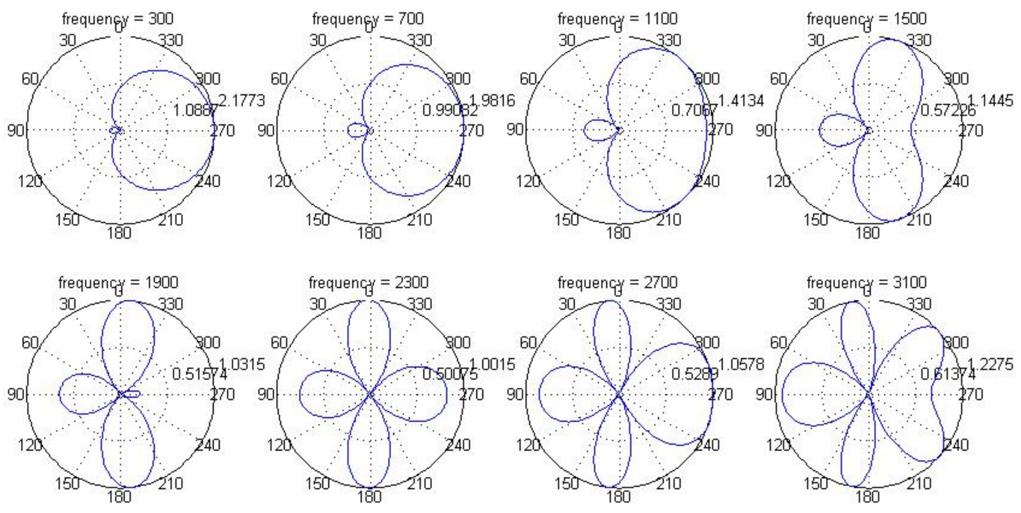


Figure 5.10: MVDR: Polar plots for the frequency range 300-3100 Hz

Table 5.1 gives an overview of the discussed algorithms with respect to the goals. Considering the goals the table shows that the MVDR achieves all of them, while the DAS requires certain conditions regarding frequencies and placement of the noise source.

Table 5.1: An overview of the DAS and MVDR beamformer based on the goals of Chapter 3

Goals	**DAS**	**MVDR**
Filter white noise (-3 dB)	yes	yes
Filter punctuated noise sources (-10 dB)	for certain frequencies and angles	yes
Filtering in the frequency range of 300 - 3400 Hz	for certain frequencies and angles	yes

Chapter 6

Conclusions and Recommendations

From this thesis we can conclude that beamforming on smartphones using a two microphone array is certainly possible. This chapter will describe our conclusions and recommendations, combining the results from Chapter 5 with the specifications from Chapter 3. Lastly, the design process is schematically depicted in Appendix B and the Matlab and Java codes can be found in Appendix C and D.

6.1 Conclusions

We have tested the feasibility of beamforming algorithms for two sensor array systems. From the results gathered in this thesis we can draw the following two conclusions.

Firstly, from the simulations for the DAS beamformer we can conclude that the algorithm succeeds in suppressing noise without doing the same for the target source in certain environments.

Drawbacks of the DAS beamformer are that the algorithm does not adapt to the direction of the noise signal, so for certain angles the suppression of noise will not be high or even negligible.

Another disadvantage of the DAS beamformer is that it operates in a limited frequency range. At a signal with low frequencies the DAS beamformer performs poorly. This performance improves for higher frequencies after which the performance declines again. Regarding the terms of reference in Appendix A the Delay and Sum does not achieve the goals we set out for our algorithm.

Secondly, even though the MVDR beamformer requires a higher degree of computational complexity, it does posses better beamforming qualities. This is due to its ability to block one noise source apart from steering a '1'-response to the target direction.

A drawback from the MVDR is that initialisation is required to estimate the noise signal. If this condition is satisfied however, the goals of 3 dB white and 10 dB punctuated noise reduction are both achieved. Also the MVDR beamformer operates within the frequency range 300-3400 Hz as specified in the terms of reference.

6.2 Recommendations and Discussion

With the test results from Chapter 5 we can conclude the MVDR is the beamformer we would recommend for implementation on a smartphone. Unfortunately we did not simulate the Generalized Sidelobe Canceller (GSC) due to time constraints. Performance is expected to be similar to the MVDR however, since its main purpose is to reduce computational complexity. Because this was not a bottleneck for our MVDR algorithm we decided to drop the GSC in testing.

When we tested the DAS algorithm on external microphones, we could not record with enough accuracy. After this test, we assumed the test set-up was not functional. Unfortunately we did not have the means nor the time to acquire a new test set-up. In this thesis we only simulated the algorithms in Matlab and further field testing is required.

We assumed a far and free field situation, meaning there would be no suppression or reflection and the received sound waves would be planar. Even though there are situations in which these assumptions hold, it is certainly not guaranteed for every situation in which a smartphone is involved. Therefore we recommend further research on the degeneration of the filter when these assumptions are invalid.

Bibliography

[1] A. Greensted, "Delay sum beamforming." [Online]. Available: http://www.labbookpages.co.uk/audio/beamforming/delaySum.html

[2] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855861

[3] H. Levitt, "Noise reduction in hearing aids: An overview," *Journal of Rehabilitation Research and Development*, vol. 38, no. 1, Jan. 2001.

[4] B. Van Veen and K. Buckley, "Beamforming: a versatile approach to spatial filtering," *ASSP Magazine, IEEE*, vol. 5, no. 2, pp. 4–24, Apr. 1988.

[5] G. W. Elko, "Microphone array systems for hands-free telecommunication," *Speech Commun.*, vol. 20, no. 3-4, pp. 229–240, Dec. 1996. [Online]. Available: http://dx.doi.org/10.1016/S0167-6393(96)00057-X

[6] S. Argentieri, P. Danes, and P. Soueres, "Modal analysis based beamforming for nearfield or farfield speaker localization in robotics," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct. 2006, pp. 866–871.

[7] S. A. Jacek Dmochowski, Jacob Benesty, "On spatial aliasing in microphone arrays," *Signal Processing, IEEE Transactions on*, vol. 57, no. 4, pp. 1383–1395, Apr. 2009.

[8] "Nyquist sampling theorem." [Online]. Available: http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

[9] "Voice frequency." [Online]. Available: http://en.wikipedia.org/wiki/Voice_frequency

[10] "Samsung galaxy s2 specifications." [Online]. Available: http://www.samsung.com/global/microsite/galaxys2/html/specification.html

[11] P. Kraniauskas, "A plain man's guide to the fft," *Signal Processing Magazine, IEEE*, vol. 11, no. 2, pp. 24–35, April 1994.

[12] J. Van de Sande, "Real-time beamforming and sound classification parameter generation in public environments," Master thesis, Delft University of Technology, Feb. 2012.

[13] D. Ba, D. Florencio, and C. Zhang, "Enhanced mvdr beamforming for arrays of directional microphones," in *Multimedia and Expo, 2007 IEEE International Conference on*, July 2007, pp. 1307–1310.

[14] M. Murthi and B. Rao, "All-pole modeling of speech based on the minimum variance distortionless response spectrum," *Speech and Audio Processing, IEEE Transactions on*, vol. 8, no. 3, pp. 221–239, May 2000.

[15] J. Bitzer, K. Simmer, and K. Kammeyer, "Multi-microphone noise reduction techniques for hands-free speech recognition -a comparative study-."

[16] C. Annis, "Correlation." [Online]. Available: http://www.statisticalengineering.com/correlation.htm

[17] L. Resende, R. Souza, and M. Bellanger, "Multisplit least-mean-square adaptive generalized sidelobe canceller for narrowband beamforming," in *Image and Signal Processing and Analysis, 2003. ISPA 2003. Proceedings of the 3rd International Symposium on*, vol. 2, Sept. 2003, pp. 976–980 Vol.2.

[18] J. McDonough, "Lecture slides, generalized sidelobe canceller," Jan. 2009. [Online]. Available: http://www.distant-automatic-speech-recognition.org/educational-materials/dsr-lecture-2008/lectures/2009-01-12/beamer-lecture.pdf

[19] Y. Chu and W.-H. Fang, "A novel wavelet-based generalized sidelobe canceller," *Antennas and Propagation, IEEE Transactions on*, vol. 47, no. 9, pp. 1485–1494, Sep. 1999.

[20] L. Griffiths and C. Jim, "An alternative approach to linearly constrained adaptive beamforming," *Antennas and Propagation, IEEE Transactions on*, vol. 30, no. 1, pp. 27–34, Jan. 1982.

[21] I. McCowan, "Robust speech recognition using microphone arrays," PhD thesis, Queensland University of Technology, Australia, 2001.

[22] N. Jablon, "Steady state analysis of the generalized sidelobe canceller by adaptive noise cancelling techniques," *Antennas and Propagation, IEEE Transactions on*, vol. 34, no. 3, pp. 330–337, Mar. 1986.

[23] L. Lepauloux, P. Scalart, and C. Marro, "Computationally efficient and robust frequency-domain GSC," in *12th IEEE International Workshop on Acoustic Echo and Noise Control*, Tel-Aviv, Israël, Aug 2010.

[24] G. Fudge and D. Linebarger, "A calibrated generalized sidelobe canceller for wideband beamforming," *Signal Processing, IEEE Transactions on*, vol. 42, no. 10, pp. 2871–2875, Oct. 1994.

[25] S. Rapuano and F. Harris, "An introduction to fft and time domain windows," *Instrumentation Measurement Magazine, IEEE*, vol. 10, no. 6, pp. 32–44, Dec. 2007.

[26] F. Harris, "On the use of windows for harmonic analysis with the discrete fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, Jan. 1978.

[27] P. Baggenstoss, "On the equivalence of hanning-weighted and overlapped analysis windows using different window sizes," *Signal Processing Letters, IEEE*, vol. 19, no. 1, pp. 27–30, Jan. 2012.

[28] "Spectral leakage." [Online]. Available: http://en.wikipedia.org/wiki/Spectral_leakage

[29] M. T. Flanagan, "Java scientific library." [Online]. Available: http://www.ee.ucl.ac.uk/~mflanaga/java/index.html

Appendix A

Program Requirements and Specifications

The project is split up in two parts, the beamforming algorithm and the application for the smartphone. The requirements discussed here are aimed at the beamforming algorithms. Therefore, we will not discuss the user interface or the implementation of the application in this thesis.

1 Requirements with Regards to the Intended Use of the Product.

[1.1] The algorithm will need to be able to filter at least 3 dB of white noise power.

[1.2] It should be possible to filter at least 1 punctuated noise source by decreasing its power by at least 10 dB.

[1.3] It should be possible to aim a beam with a '1' response in an indicated target direction.

[1.4] The noise source can be located anywhere, except at the target direction

[1.5] Filtering has to be possible on the spectrum used for human speech, i.e. 300-3400 Hz.

[1.6] All of the above should be achieved in situations where the far- and free field assumptions hold.

2 Requirements with Regards to the Design of the System.

[2.1] Sampling will be done at a frequency of 8 kHz to mitigate computational complexity, while avoiding frequency aliasing.

[2.2] The algorithm will sample in time-frames of 20 ms to ensure that enough samples are taken, while being short enough to enable smooth speech applications.

[2.3] The execution time of the algorithm can not exceed the length of the time frames, i.e. 20 ms.

3 Requirements with Regards to the Production Process.

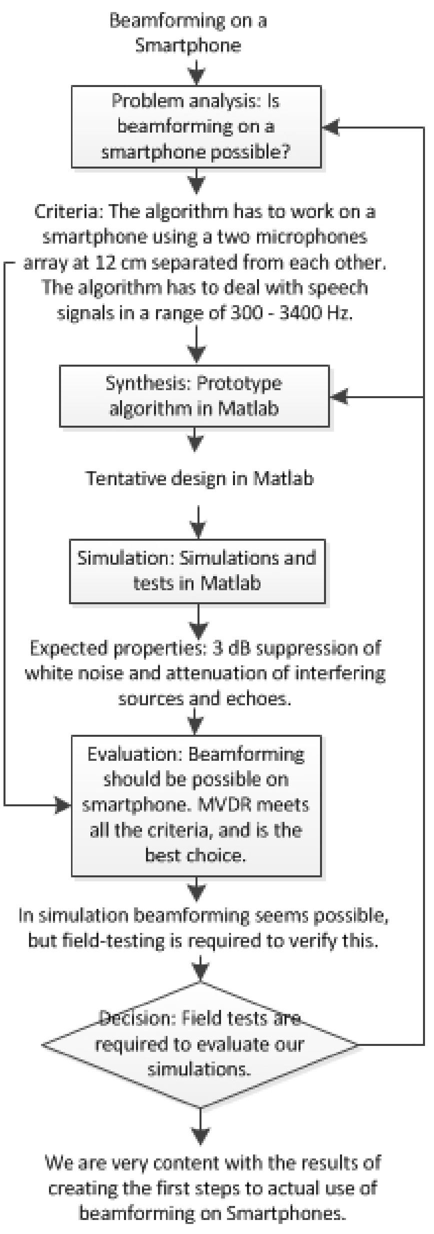
[3.1] Algorithms will be simulated and tested using Matlab.

[3.2] After testing, the algorithm will be implemented in Java, as to make it compatible with the Android operating system.

Appendix B

Design Scheme

Functional Block Diagram of the Design Process


Beamforming on a Smartphone
Problem analysis: Is beamforming on a smartphone possible?
Criteria: The algorithm has to work on a smartphone using a two microphones array at 12 cm separated from each other. The algorithm has to deal with speech signals in a range of 300 - 3400 Hz.
Synthesis: Prototype algorithm in Matlab
Tentative design in Matlab
Simulation: Simulations and tests in Matlab
Expected properties: 3 dB suppression of white noise and attenuation of interfering sources and echoes.
Evaluation: Beamforming should be possible on smartphone. MVDR meets all the criteria, and is the best choice.
In simulation beamforming seems possible, but field-testing is required to verify this.
Decision: Field tests are required to evaluate our simulations.
We are very content with the results of creating the first steps to actual use of beamforming on Smartphones.

Appendix C

Matlab Code

Delay and Sum

```
%% INITIALIZE
clear all; close all; clc;

% Properties of the phone
d = 0.12;                   % Distance between the mics
N = 2;                      % Amount of mics

% Physical constants
c = 343;                    % Speed of sound in air

% Properties of the signal
freq = 300;                 % The frequency of the source signal
freq2 = 600;                % The frequency of the noise signal
theta = pi;                 % Angle of the source signal
theta2 = 2 * pi / 9;            % Angle of the noise signal

% Our chosen properties
fs = 80000;                  % Sample frequency
time_frame = 0.02;          % Duration of 1 time frame
sptf = fs * time_frame;     % The amount of samples per time frame
noise_amp = 0.1;            % The noise amplitude
L = 2^(ceil(log2(sptf)));   % The power of 2 for the FFT
totaltime = 1;              % The total time of the signal

%% INPUT SIGNAL
tt = linspace(0, totaltime, fs * totaltime).';  % Samples till totaltime

tau1 = d / c * sin(theta);  % Time delay to mic 2 for signal
tau2 = d / c * sin(theta2); % Time delay to mic 2 for noise

signal = sin(2 * pi * freq * tt);           % Signal mic 1
signal2 = sin(2 * pi * freq * (tt - tau1)); % Signal mic 2

noise = 4 * sin(2 * pi * freq2 * tt) + noise_amp * randn(totaltime * fs, 1);           % ↩
    Noise mic 1
noise2 = 4 * sin(2 * pi * freq2 * (tt - tau2)) + noise_amp * randn(totaltime * fs, 1);  % ↩
    Noise mic 2

signal_total = signal + noise;      % Sum of the signal and noise at mic 1
signal_total2 = signal2 + noise2;   % Sum of the signal and noise at mic 2

x1 = [signal, signal2];
xsum = [signal_total, signal_total2];

f_center = linspace(0, fs - fs / (2 * L), L).';         % The centerfrequencies for each bin
```

```
zeta = −1i ∗ 2 ∗ pi ∗ f_center ∗ d ∗ sin(theta) / c;    % The phase shift of the signal for ↩
     mic 2

a_n = 1 / N;                                % The amplitude weights
w = a_n ∗ [(exp(zeta)), ones(L, 1)];    % Weights with delays

%% RECONSTRUCTING THE ORIGINAL SIGNAL
nr_frames = floor((length(signal) − sptf) / (sptf / 2));
rec_signal = zeros(size(signal(:, 1)));
rec_signal_total = zeros(size(signal_total(:, 1)));

testsignal1 = zeros(L, nr_frames);
testsignal2 = zeros(L, nr_frames);

for I = 1 : nr_frames
  win = [sqrt(hanning(sptf)), sqrt(hanning(sptf))];
  frame_x1 = x1((sptf / 2) ∗ (I − 1) + 1 : (sptf / 2) ∗ (I − 1) + sptf, :) .∗ (win);
  fft_x1 = fft(frame_x1, L);

  frame_xsum = xsum((sptf / 2) ∗ (I − 1) + 1:(sptf / 2) ∗ (I − 1) + sptf, :) .∗ (win);
  fft_xsum = fft(frame_xsum, L);
  fft_xsum(end, :) = real(fft_xsum(end, :));

  fft_x1([1, L / 2 + 1], :) = real(fft_x1([1, L / 2 + 1], :));
  part1 = fft_x1(1 : L / 2 + 1, :);
  fft_x1 = [part1; conj(flipud(part1(2 : end − 1, :)))];

  fft_xsum([1, L / 2 + 1], :) = real(fft_xsum([1, L / 2 + 1], :));
  part1 = fft_xsum(1 : L / 2 + 1, :);
  fft_xsum = [part1; conj(flipud(part1(2 : end − 1, :)))];
  estimate_fft_signal = w .∗ (fft_x1);
  estimate_fft_signal_2 = estimate_fft_signal(:, 1) + estimate_fft_signal(:, 2);

  estimate_fft_signal_total = w .∗ (fft_xsum);
  estimate_fft_signal_total_2 = estimate_fft_signal_total(:, 1) + estimate_fft_signal_total↩
      (:, 2);
  testsignal2(:, I)= estimate_fft_signal_total_2;
  testsignal1(:, I) = fft_xsum(:, 1);

  estimate_signal = real(ifft(estimate_fft_signal_2));

  estimate_signal_total = real(ifft(estimate_fft_signal_total_2));
  rec_signal((sptf / 2) ∗ (I − 1) + 1 : (sptf / 2) ∗ (I − 1) + sptf) = rec_signal((sptf / 2)↩
       ∗ (I − 1) + 1 : (sptf / 2) ∗ (I − 1) + sptf) + estimate_signal(1 : sptf) .∗ (sqrt(↩
      hanning(sptf)));
  rec_signal_total((sptf / 2) ∗ (I − 1) + 1 : (sptf / 2) ∗ (I − 1) + sptf) = ↩
      rec_signal_total((sptf / 2) ∗ (I − 1) + 1 : (sptf / 2) ∗ (I − 1) + sptf) + ↩
      estimate_signal_total(1 : sptf) .∗ (sqrt(hanning(sptf)));
end

%% PLOTS
% Time plot
t = linspace(0, totaltime, totaltime ∗ fs);

figure;
subplot(2, 1, 2)
plot(t, real(signal_total2(1 : totaltime ∗ fs)));
hold on
plot(t, real(rec_signal_total(1 : totaltime ∗ fs)), '.−r');
plot(t, real(signal2(1 : totaltime ∗ fs)), '−−g');
legend('noisy in', 'beamformed', 'desired in')
title('Incoming, beamformed and desired signals')
axis([0.200 0.210 −5 5]);
xlabel('Time (in seconds)');
ylabel('Amplitude');

% Frequency plot
subplot(2, 1, 1)
plot(linspace(1, fs, L), 10 ∗ log10(mean(abs(testsignal1).^2, 2))) ;
hold on
plot(linspace(1, fs, L), 10 ∗ log10(mean(abs( testsignal2).^2, 2)), 'r');
title('Spectrum plots of in signal and signal after beamforming in dB')
legend('in signal', 'beamformed signal')
```

```
axis([0 4000 10 50]);
xlabel('Frequency (in Hertz)');
ylabel('Magnitude (in dB)');

% Polar plot
figure;

polarfreq = [300, 600];
polarbin = zeros(1, length(polarfreq));

for I = 1: length(polarfreq);
  polarbin(I) = floor(polarfreq(I) / fs * L);

  delta = ((c / f_center(polarbin(I))) / d)^-1;
  [w_dakje polarhoek] = beam_resp(w(polarbin(I), :), L, delta);

  subplot(ceil(length(polarfreq) / 2), 2, I)
  polar90(polarhoek, abs(w_dakje));
  title(['Frequency = ', num2str(polarfreq(I))])
end
```

MVDR

```
%% INITIALIZE
clear all; close all; clc;

% Properties of the phone
d = 0.12;                   % Distance between the mics
N = 2;                      % Amount of mics

% Physical constants
c = 343;                    % Speed of sound in air

% Properties of the signal
freq = 1500;                % The frequency of the source signal
freq2 = 2500;               % The frequency of the noise signal
theta = pi;                 % Angle of the source signal
theta2 = 2 * pi / 9;        % Angle of the noise signal

% Our chosen properties
fs = 80000;                  % Sample frequency
time_frame = 0.02;          % Duration of 1 time frame
sptf = fs * time_frame;     % The amount of samples per time frame
noise_amp = 0.1;            % The noise amplitude
L = 2^(ceil(log2(sptf)));   % The power of 2 for the FFT
totaltime = 1;              % The total time of the signal

%% SIGNAL
tt = linspace(0, totaltime, fs * totaltime).';
tau1 = d / c * sin(theta);
tau2 = d / c * sin(theta2);

noise1 = sin(2 * pi * 2500 * tt) + noise_amp * randn(fs * totaltime, 1);
noise2 = sin(2 * pi * 2500 * (tt - tau2)) + noise_amp * randn(fs * totaltime, 1);

signal1 = sin(2 * pi * freq * tt) + noise1;
signal2 = sin(2 * pi * freq * (tt - tau1)) + noise2;

noise = [noise1, noise2];
signal = [signal1, signal2];

estimate_signal = zeros(L, 1);

nr_frames = floor((length(signal1) - sptf) / (sptf / 2));
rec_signal = zeros(size(signal(:, 1)));

%% CALCULATING THE NOISE CORRELATION MATRIX
for J = 1 : 10 + 1
  win = [sqrt(hanning(sptf)), sqrt(hanning(sptf))];
  frame_noise = noise((sptf / 2) * (J - 1) + 1 : (sptf / 2) * (J - 1) + sptf, :) .* (win);
  fft_noise = fft(frame_noise, L);

  fft_noise([1, L / 2 + 1], :) = real(fft_noise([1, L / 2 + 1], :));
  part1 = fft_noise(1: L / 2 + 1, :);

  for I = 1 : L / 2 + 1
    noise_cor(1, 1, I, J) = abs(part1(I, 1)).^2;
    noise_cor(2, 2, I, J) = abs(part1(I, 2)).^2;
    noise_cor(1, 2, I, J) = part1(I, 1) * conj(part1(I, 2));
    noise_cor(2, 1, I, J) = part1(I, 2) * conj(part1(I, 1));
  end

end
noise_cor = mean(noise_cor, 4);

%% RECONSTRUCTING THE ORIGINAL SIGNAL
for J = 1 : nr_frames + 1
  win = [sqrt(hanning(sptf)), sqrt(hanning(sptf))];
  frame_signal = signal((sptf / 2) * (J - 1) + 1 : (sptf / 2) * (J - 1) + sptf, :) .* (win);
  fft_signal = fft(frame_signal, L);
  fft_signal([1, L / 2 + 1], :) = real(fft_signal([1, L / 2 + 1], :));
  part1 = fft_signal(1: L / 2 + 1, :);
  fft_signal = [part1; conj(flipud(part1(2: end - 1, :)))];
  fft_signal_mat(:, J) = fft_signal(:, 1);

  for I = 0 : L / 2
    f_center = I * fs / L;
    zeta = - 1i * 2 * pi * f_center * d * sin(theta) / c;

    a_n = 1 / N;
    c_ = [1; exp(zeta)];

    w = (noise_cor(:, :, I + 1) \ c_) / ((c_' / noise_cor(:, :, I + 1)) * c_);
    w = conj(w);
    testsignal(:, I + 1) = w;

    estimate_signal(I + 1) = w.' * fft_signal(I + 1, :).';
  end

  fft_estimate_signal_mat(:, J) = estimate_signal;
  estimate_signal = estimate_signal(1: L / 2 + 1);
  estimate_signal([1, L / 2 + 1], :) = real(estimate_signal([1, L / 2 + 1], :));
  estimate_signal = [estimate_signal; flipud(conj(estimate_signal(2 : end - 1)))];
  td_estimate_signal = real(ifft(estimate_signal(1 : L)));
  rec_signal((sptf / 2) * (J - 1) + 1: (sptf / 2) * (J - 1) + sptf) = rec_signal((sptf / 2) ↩
      * (J - 1) + 1 : (sptf / 2) * (J - 1) + sptf) + td_estimate_signal(1 : sptf) .* (sqrt(↩
      hanning(sptf)));
end

%% PLOTS
% Time plot
figure;
subplot(2, 1, 2)
plot(linspace(0, totaltime, length(signal1)), signal1, 'b');
hold on
plot(linspace(0, totaltime, length(rec_signal)), rec_signal, '--r');
plot(linspace(0, totaltime, length(signal1)), sin(2 * pi * freq * tt), '--k')
legend('noisy in', 'beamformed', 'desired in')
title('Incoming, beamformed and desired signals')
axis([0.200 0.204 -2 2]);
xlabel('Time (in seconds)');
ylabel('Amplitude');

% Frequency plot
subplot(2, 1, 1)
plot(linspace(1, fs, L), 10 * log10(mean(abs(fft_signal_mat).^2, 2)));
hold on
plot(linspace(1, fs, L), 10 * log10(mean(abs(fft_estimate_signal_mat).^2, 2)), 'r');
title('Spectrum plots of in signal and signal after beamforming in dB')
legend('in signal', 'beamformed signal')
axis([0 4000 -10 40]);
xlabel('Frequency (in Hertz)');
ylabel('Magnitude (in dB)');

% Polar plot
figure;

f_center = linspace(0, fs - fs / (2 * L), L).';
polarfreq = [1500, 2500];
polarbin = zeros(1, length(polarfreq));

for I = 1: length(polarfreq);
  polarbin(I) = floor(polarfreq(I) / fs * L);

  delta = ((c / f_center(polarbin(I))) / d)^-1;
  [w_dakje polarhoek] = beam_resp(testsignal(:, polarbin(I)), L, delta);

  subplot(ceil(length(polarfreq) / 2), 2, I)
  polar90(polarhoek, abs(w_dakje));
  title(['Frequency = ', num2str(polarfreq(I))])
end
```

Appendix D

Java Code

Delay and Sum

With thanks to [29]

```java
import flanagan.complex.*;
import flanagan.math.*;

/**
 * @author Mark Aarts, Hendrik Pries, Arjan Doff
 * This class is used to filter signals using a beamforming technique called Delay-And-Sum (↩
    DAS)
 */
public class DelayAndSum
{
    ///////////////////////////////// Variables //////////////////////////////////
    float distance = 0.1f;                                                      // Distance ↩
        between the mics
    int N = 2;                                                                  // Amount of↩
         mics
    int c = 343;                                                                // Speed of ↩
        sound
    int fs = 8000;                                                              // Samples ↩
        per second (or sample frequency)
    float timeFrameDuration = 0.02f;                                            // The ↩
        duration of 1 timeframe
    int sptf = (int) (fs * timeFrameDuration);                                  // Samples ↩
        per timeframe
    int pow2 = (int) Math.pow(2, 32 - Integer.numberOfLeadingZeros(sptf - 1));  // For the ↩
        zero padding of the Fourier transform
    int pow = pow2 / 2 + 1;                                                     // For the ↩
        use of a mirrored spectrum
    Complex[][] weights;                                                        // The ↩
        weights to be calculated
    double[] estimatedSignal;                                                   // The ↩
        estimated signal to be calculated
    int nrOfFrames;                                                             // The ↩
        amount of timeframes
    /////////////////////////////////////////////////////////////////////////////

    /**
     * The constructor for estimating the original signal.
     * @param signalMic1
     * @param signalMic2
     * @param theta
     */
    public DelayAndSum(double[] signalMic1, double[] signalMic2, double theta)
    {
        weights = calculateWeights(theta);
        nrOfFrames = (int) Math.floor(2*(signalMic1.length - sptf) / sptf) + 1;

        double[][] tempSignal = new double[nrOfFrames][sptf];
        estimatedSignal = new double[signalMic1.length];

        // De aparte frames van het signaal halen
        for (int  i = 0; i < nrOfFrames; i++) {
            double[] signalMic1SubArray = new double[sptf];
            double[] signalMic2SubArray = new double[sptf];

            System.arraycopy(signalMic1, i * sptf / 2, signalMic1SubArray, 0, sptf);
            System.arraycopy(signalMic2, i * sptf / 2, signalMic2SubArray, 0, sptf);

            tempSignal[i] = calculateEstimatedOriginalSignal(signalMic1SubArray, ↩
                signalMic2SubArray);
        }

        // Windows laten overlappen
        for (int i = 0; i < nrOfFrames; i++) {
            for (int  j = 0; j < sptf; j++) {
                estimatedSignal[i * sptf / 2 + j] += tempSignal[i][j];
            }
        }
    }

    /**
     * @return The estimated signal after having done all necessary calculations
     */
    public double[] getEstimatedSignal()
    {
        return estimatedSignal;
    }

    /**
     * Calculate the fourier transform of the input, which should be a double and in the ↩
         time domain.
     * @param signal
     * @return A complex signal in the frequency domain.
     */
    public Complex[] calculateFourierTransform(double[] signal)
    {
        // Taking the square root in order to create a constant one when the signals overlap↩
            .
        double[] hannSignal = useSQRTHannWindow(signal);

        // Zero padding before fourier transforming
        double [] hannSignalPadded = new double[pow2];

        for (int i = 0; i < pow2; i++) {
            if (i < sptf)
                hannSignalPadded[i] = hannSignal[i];
            else
                hannSignalPadded[i] = 0;
        }

        FourierTransform fftSignal = new FourierTransform(hannSignalPadded);
        fftSignal.transform();

        return fftSignal.getTransformedDataAsComplex();
    }

    /**
     * Calculate the inverse fourier transform of the input, which should be complex and in ↩
         the frequency domain.
     * @param signal
     * @return A complex signal in the time domain
     */
    public Complex[] calculateInverseFourierTransform(Complex[] signal)
    {
        FourierTransform inverseFFTSignal = new FourierTransform(signal);
        inverseFFTSignal.inverse();

        return inverseFFTSignal.getTransformedDataAsComplex();
```

```java
    }

    /**
     * A function to create the square root of a Hann window and multiply it with the signal
     * @param signal
     * @return The signal multiplied by the square root of a Hann window.
     */
    public double[] useSQRTHannWindow(double[] signal)
    {
        double[] hannSignal = new double[sptf];

        for (int i = 0; i < sptf; i++) {
            hannSignal[i] = Math.sqrt(0.5 − 0.5*Math.cos((2 * Math.PI * i) / (sptf − 1))) * ↩
                signal[i];
        }

        return hannSignal;
    }

    /**
     * This method will estimate what the original signal is.
     * The input signals should already be sampled, so this calculation is for 1 timeframe.
     * @param signalMic1
     * @param signalMic2
     * @return the estimated signal
     */
    public double[] calculateEstimatedOriginalSignal(double[] signalMic1, double[] ↩
        signalMic2)
    {
        // Taking the FFT of the incoming signal
        Complex[] fftSignalMic1 = calculateFourierTransform(signalMic1);
        Complex[] fftSignalMic2 = calculateFourierTransform(signalMic2);

        Complex[] estimateFFTSignal = new Complex[pow2];

        // Multiplying with the beamforming weights
        for (int i = 0; i < pow; i++) {
            estimateFFTSignal[i] = weights[0][i].times(fftSignalMic1[i]).plus(weights[1][i].↩
                times(fftSignalMic2[i]));
        }

        //Mirror the spectrum
        for (int i = 0; i < pow − 2; i++)
        {
            estimateFFTSignal[i+pow] = estimateFFTSignal[pow − 2 − i].conjugate();
        }

        //Force the middle frequency bin to be real
        estimateFFTSignal[pow − 1] = new Complex(estimateFFTSignal[pow − 1].getReal(), 0);

        Complex[] complexEstimatedSignal = calculateInverseFourierTransform(↩
            estimateFFTSignal);

        return useSQRTHannWindow(new ArrayMaths(complexEstimatedSignal).↩
            subarray_as_real_part_of_Complex(0, sptf − 1));
    }

    /**
     * Calculating the weights corresponding to the signals.
     * @param signal
     * @return the weights
     */
    public Complex[][] calculateWeights(double theta)
    {
        double[] fCenter = new double[pow];
        double[] zeta = new double[pow];

        for (int i = 0; i < pow; i++) {
            // The center frequency of the bands
            fCenter[i] = ((double) fs / pow2) * i;

            // The delay
            zeta[i] = 2 * Math.PI * fCenter[i] * distance * Math.sin(theta) / c;
        }

        // Calculating the weights per frequencyband
        Complex[][] weights = new Complex[N][pow];
        Complex alpha0 = new Complex(1/((double)N), 0);

        for (int i = 0; i < pow; i++) {
            weights[0][i] = new Complex(Math.cos(−zeta[i]), Math.sin(−zeta[i])).times(1/((↩
                double)N));
            weights[1][i] = alpha0;
        }

        return weights;
    }
}
```