Smartphone Audio Acquisition and Synchronization Using an Acoustic Beacon

With Application to Beamforming

Roy Smeding (4082192)
Sjoerd Bosma (4088379)
Supervisor: dr. Jorge Martínez Castañeda

Abstract

A time-domain (TD) and a frequency-domain (FD) method for time offset (TO) compensation between received audio data are presented and implemented on Android smartphones and a processing computer. Simulations show the TD method is superior in almost all applications. Real-life experiments show the TD synchronization error is below 6 samples. A Java and MATLAB server application and an Android application are presented to acquire audio and orientation data with the intent of applying it to beamforming algorithms. Orientation data from Android smartphones is shown to be inconsistent among devices and is therefore inadequate for beamforming applications. Sampling rate offset (SRO) on Android smartphones is characterized by analyzing long duration audio recordings from the devices. Future work could include an SRO compensation algorithm, self-localization of the smartphones and distributed synchronization.

***Keywords* – Acoustic synchronization, sampling rate offset, smartphone orientation, ad-hoc beamforming**

Delft University of Technology

July 4, 2015

Acknowledgments

Succesful completion of this project would never have been possible without the excellent guidance of our supervisor, dr. Jorge Martínez Castañeda. We would like to thank him for his support, advice and guidance. Furthermore, professor Bastiaan Kleijn of the Delft University of Technology has been very helpful with his insights. Henry den Bok at the Faculty of Applied Sciences was very helpful in preparing measurement setups and dr. Jan Skoglund at Google inspired us to get the most out of this project. Professor Koen Bertels was of great help in compiling our accompanying business plan. It was a pleasure working together with our fellow students Erik, Niels, Rosalie and Tim on the encompassing system.

Finally, we would like to thank dr. Ioan Lager for organizing the Bachelor graduation project this year.

Roy Smeding
Sjoerd Bosma

Delft University of Technology

List of figures

1 Typical usage scenario for the beamforming system. 2
2 Global and local overview of the beamforming system. 3

3 TO and SRO correction stages. 6
4 Measurement setup used for SRO measurements. 8
5 SRO measurement setup in the anechoic chamber. 9
6 SRO measurement result. 10
7 TD and FD synchronization simulation results. 13

8 Android Activity user interface. 16
9 Server and Android app FSM. 19
10 Measurement of smartphone orientations. 21

11 Measurement results of synchronization. 24

12 Complete orientation measurement results. 30
13 Full TD and FD synchronization simulation results. 31
14 Complete SRO measurement results. 32
15 Synchronization in a real-world scenario. 33

16 Global coordinate system for smartphones. 34

Table of contents

List of figures **ii**

I Introduction **1**
- I.1 Problem formulation . . . 2
- I.2 Definitions and symbols . . . 2
- I.3 Requirements . . . 3

II Synchronization **5**
- II.1 Problem formulation . . . 5
- II.2 State of the art . . . 6
- II.3 SRO characterization . . . 7
- II.4 Algorithm . . . 10
- II.5 Simulation . . . 12

III System implementation **15**
- III.1 State of the art . . . 15
- III.2 Android app . . . 16
- III.3 MATLAB implementation . . . 17
- III.4 System performance . . . 19
- III.5 Orientation measurement . . . 20

IV System test **23**
- IV.1 Set-up . . . 23
- IV.2 Results . . . 24

V Discussion and recommendations **25**
- V.1 Discussion . . . 25
- V.2 Future work . . . 26
- V.3 Conclusion . . . 27

Bibliography **28**

A Full simulation and experiment results **30**
- A.I Orientation measurements . . . 30
- A.II Time offset simulation . . . 31
- A.III Sampling rate offset measurements . . . 32
- A.IV Real-life synchronization results . . . 33

B Global coordinate system used by smartphone **34**

Chapter I

Introduction

In many larger businesses, conference calls are used to communicate between different teams, or with other teams in different locations. The systems for these conference calls are generally complicated to configure, and once configured it can still be difficult to make every participant heard. One potential solution is to use a network of smartphones, which most conference call participants already possess. Each smartphone has at least one microphone, meaning a network of these microphones can be processed as a microphone array. As a result, expensive conferencing hardware can be replaced by a network of smartphones running an application. This smartphone microphone array could also be used for other applications, such as speaker identification and improving the accuracy of speech recognition systems.

In general, the applications presented above rely on spatial filtering of acoustic data. In other words, they strive to enhance speech by taking into account the locations of the microphones and sources. This technique is commonly known as beamforming [1]. In these examples, the smartphones can be seen as ad-hoc arrays of wireless acoustic sensors [2]. Further enhancement of audio may be accomplished by incorporating microphone directivities into the beamforming algorithms [3, 4]. Beamforming with ad-hoc microphone arrays requires known locations of microphones [5] and synchronization of sensor data [6, 7]. Using directivities further requires the orientation of the smartphones to be known [2].

Acoustic self-localization of smartphones is possible, though with limited accuracy [6, 8]. Using external loudspeakers, sub-5 cm variances of positioning algorithms have been realized on a variety of wireless sensors [9]. Synchronization of the microphone data can be achieved using correlation processing for both time offset [10, 11] and sampling rate offset [7, 12] correction. Orientation estimates may be directly obtained from Android smartphones, but its accuracy is severely limited [13].

Solutions to each individual challenge stated above are described in the literature, but no working beamforming system including directivities using smartphone microphones is known to the authors. This work, in conjunction with the work of Van Wijngaarden and Wouters [14] and Brinkman and De Rooij [15], strives to form the basis of such a system.

The rest of this document is structured as follows: the current chapter further specifies the exact problem addressed in this work, followed by a list of requirements on the resulting system. Chapter II details the challenges faced in synchronizing the audio signals, along with solutions for these challenges and evaluation of these solutions. Chapter III details the data acquisition system that was built, and gives quantitative performance results for orientation measurement in the context of the target application. Chapter IV describes experiments done on the resulting subsystem. Finally, the document is concluded by a discussion of the achieved performance, as well as recommendations for future work, in chapter V.

I.1 Problem formulation

The aim of the complete system is to perform beamforming on smartphone microphone signals. A diagram of an example usage scenario is shown in Fig. 1. A number of smartphones are located on a table in a normal office environment. Located around the table are a number of desired audio sources (e.g. people speaking) and noise sources (e.g. background conversation or traffic outside). The position of all sources and microphones is assumed known. By combining these recordings, taking into account the location information, an output audio signal is produced that ideally has better perceived quality than the individual microphone signals. This work specifically focuses on the overall signal acquisition and audio synchronization aspects of the system. Application of the actual beamforming algorithms is left to Van Wijngaarden and Wouters [14] who also incorporate work done on microphone directivity measurements by Brinkman and De Rooij [15] (see Fig. 2a).

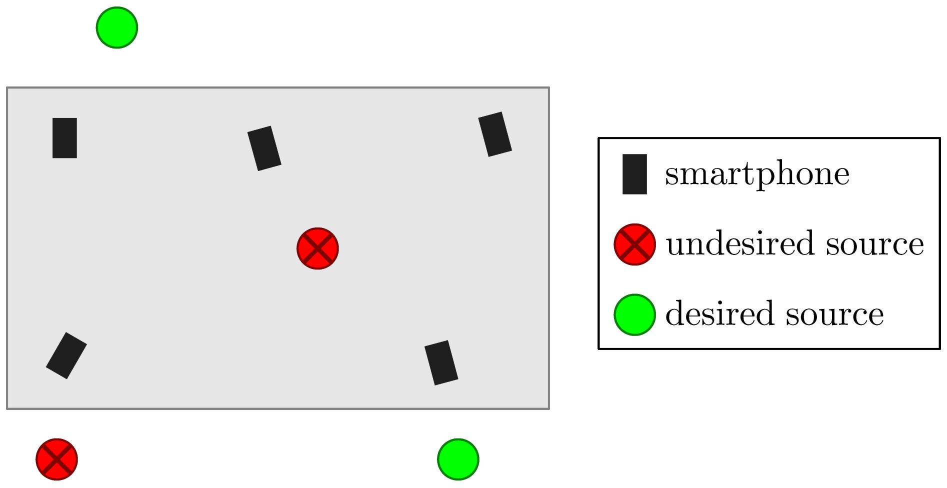


Fig. 1. Typical usage scenario for the beamforming system. Several smartphones are positioned on a table to capture desired audio sources, while rejecting sources of noise.

I.1.1 Scope

This work describes a complete system for sensor data acquisition, taking the form of an Android smartphone application and a MATLAB and Java processing server. An overview of the components of this system is shown in Fig. 2b. The choices for these platforms will be outlined in sections III.2 and III.3 respectively.

This work will also look at the state of the art of both orientation estimation and localization and make some recommendations for future work. It was chosen not to incorporate this functionality in the application, but instead assume the smartphone orientations and positions are known because current localization and orientation measurements do not give sufficient accuracy (see sections III.1.2 and III.1.2).

The following section details the definitions and mathematical symbols used in the rest of this work. A set of requirements for this subsystem follows in section I.3.

I.2 Definitions and symbols

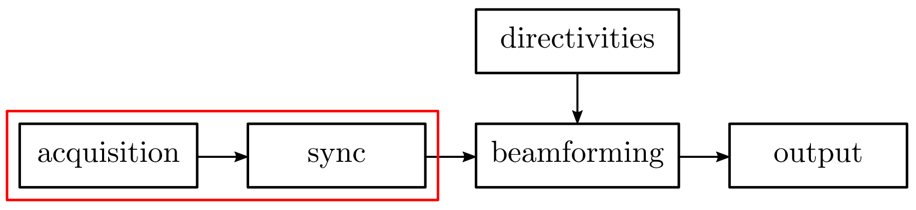
Throughout the rest of this document, some shorthand form of long product names will be used. Below, these are enumerated and explained. The shorthand forms are also introduced.

Android™ platform	Smartphone operating system – abbreviated as Android.
Google Nexus 5™	Smartphones used in this work – abbreviated as Nexus 5.
MATLAB Student R2014b	Numerical computing environment – abbreviated as MATLAB.
Java®	Programming language maintained by Oracle Corporation – abbreviated as Java.

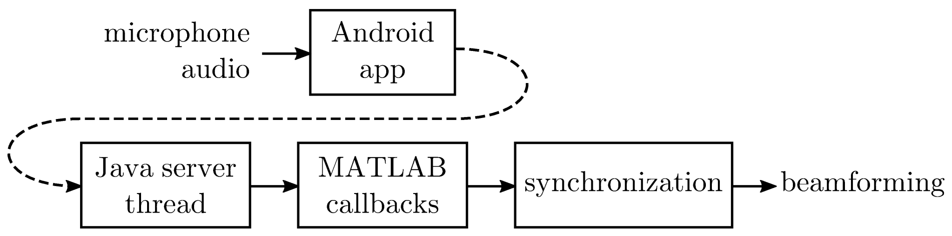
I.2.1 Mathematical symbols

The following mathematical symbols are used throughout this document:

$\boldsymbol{a}, \boldsymbol{A}$	time-domain and discrete-frequency-domain vector, respectively
$\mathscr{F}$	the discrete Fourier transform (DFT) operator, i.e. $\mathscr{F}\{\boldsymbol{f}\,[t]\} = \boldsymbol{F}\,[2\pi k/N]$
$\circ\!\!-\!\!-$	the right-side element is the DFT of the left-side element, i.e. $\boldsymbol{f}\,[t] \circ\!\!-\!\!- \boldsymbol{F}\,[2\pi k/N]$.
$\boldsymbol{a} \star \boldsymbol{b}$	the cross correlation between $\boldsymbol{a}$ and $\boldsymbol{b}$.
$\boldsymbol{a} * \boldsymbol{b}$	the convolution of $\boldsymbol{a}$ and $\boldsymbol{b}$.
$a \cdot b$	the product of a and b.
$\boldsymbol{a} \odot \boldsymbol{b}$	the element-wise multiplication of $\boldsymbol{a}$ and $\boldsymbol{b}$.
$\overline{a}$	the complex conjugate of a.
$0.\overline{1}$	a recurring decimal, i.e. $0.111\ldots$
$\hat{a}$	an estimate for a.



(a) A global overview of the beamforming system structure. This work describes the stages highlighted in red.



(b) A closer look at the general outline of the system described in this work.

Fig. 2. Global and local overview of the beamforming system.

I.3 Requirements

A set of requirements has been established before the subsystem was implemented, in order to judge if the finished subsystem complies with expectations. In the rest of this chapter, *system* will refer to the complete, integrated system of smartphones and processing computer.

I.3.1 Requirements concerning the intended use

Requirements concerning the Android application.

[1] The Android application must have an easy-to-use graphical interface.
[2] The Android application must connect using TCP/IP to a running server that will accept the recorded audio data.
[3] The Android application must stream the data acquired from its microphone after receiving the corresponding signal from the central computer.
[4] The Android application must stop streaming the recorded sound after receiving a stop signal.
[5] The Android application must adhere strictly to the defined communication protocol with the central computer.

Requirements concerning the MATLAB application.

[1] The MATLAB program must accept incoming network connections from smartphones so data can be transmitted between them.
[2] The MATLAB program must be capable of handling 8 simultaneous phone connections.
[3] The MATLAB program must adhere strictly to the defined communication protocol with the phones.
[4] The MATLAB program must provide the beamforming algorithms with discrete, synchronized ‘blocks’ with predefined size of sampled audio data from all phones.
[5] The MATLAB program must also provide a unique phone identifier to the beamforming algorithms along with the aforementioned audio data.

I.3.2 Requirements concerning the system

[1] The system must be usable in a typical office environment (e.g. a meeting room).
[2] The Android application must run on Nexus 5 smartphones.
[3] The computer program must run on computers running MATLAB version R2014b with networking capability.

I.3.3 Requirements regarding the production

[1] The Android application will be developed using version 24.2 of the Android SDK[1].
[2] The MATLAB code will be developed using MATLAB version R2014b.
[3] Java applications will be developed using version 1.7.0 of the JDK[2].

[1] Software Development Kit
[2] Java Development Kit.

Chapter II

Synchronization

In traditional multichannel analog-to-digital converters (ADCs) multiple sources are sampled using the same reference clock signal. As a result, temporal and frequency synchronization between the channels is guaranteed. However, when audio signals are received over a (wireless) network of independent ADCs, the samples are not synchronized in either time or frequency. For the beamforming algorithms supplied with data by this system, it is imperative that the signals are synchronized [12]. As a result, one or more synchronization algorithms are needed. This chapter describes these problems in a more systematic manner, explores the literature for solutions and describes the algorithms used to accomplish synchronization, concluding with both simulated and experimental performance results for these algorithms.

II.1 Problem formulation

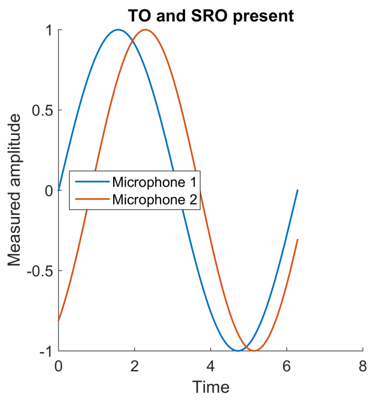
Two distinct aspects of the received signal samples require synchronization. The first is a fixed time offset between received signals, caused by network lag and latency in handling the commands sent from the central computer. The second is due to frequency deviations from the intended sampling rate between different analog-to-digital converters.

II.1.1 Time offset

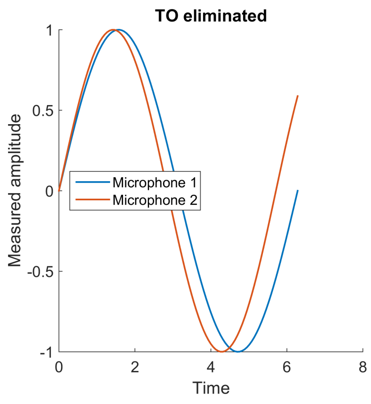
Formally, the time offset (TO) between two receivers may be described by the following formula [16]:

$$\Delta t = \Delta t_{\text{send}} + \Delta t_{\text{access}} + \Delta t_{\text{prop}} + \Delta t_{\text{receive}} \tag{II–1}$$

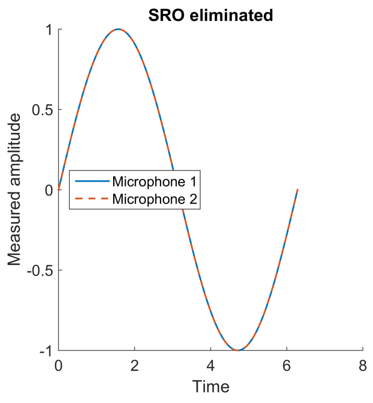
In this case, Δt_{send} and Δt_{access} are equal for all smartphones because they consist of the shared loudspeaker and the acoustic channel which is common to all receivers. $\Delta t_{\text{receive}}$ is prone to the random variation associated with audio recording, wireless networking, and processing and Δt_{prop} is due to the propagation time of sound in air. To correct for TO, the propagation delay and reception delay must both be known; in the first step of synchronization, the sum $\Delta t_{\text{prop}} + \Delta t_{\text{receive}}$ is compensated as exemplified in Fig. 3a-3b. In a second step the propagation delay is added back to the sampled signal, effectively compensating each smartphone for its expected time-difference of arrival (TDOA). This process is illustrated in Fig. 3c-3d.



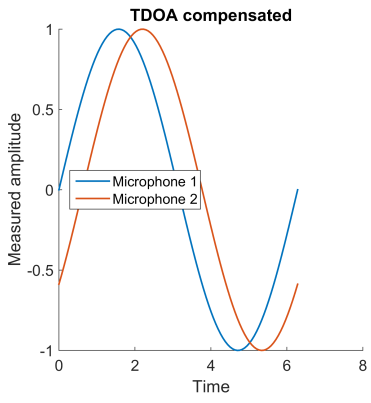
(a) Both TO and SRO are present in the measured microphone samples.



(b) TO eliminated, SRO is still present.



(c) TO and SRO have now been eliminated, fully synchronizing the signals.



(d) Samples corrected for TDOA from loudspeaker to microphone location.

Fig. 3. Example of TO and SRO correction stages.

II.1.2 Sampling rate offset

The second synchronization requirement is due to clock skew in the individual ADCs used to record audio. Since the ADCs used in smartphones are not typically designed for high-precision sampling, a sampling rate offset (SRO) up to even 10 Hz may be present [17]. The detrimental influence of sampling rate offset on signal processing performance has been demonstrated by Cherkassky and Gannot [12].

If left uncompensated, sampling rate offset results in a time offset that changes with time. In order to correct for this, the sample rate offset must be estimated, after which the microphone signals can be resampled to conform to a precise global sample rate [7].

The following section describes some possible solutions to these two synchronization problems.

II.2 State of the art

II.2.1 Time offset correction

The Network Time Protocol (NTP) is used worldwide for clock synchronization, but generally has errors on the order of tens of milliseconds, too much for signal processing applications. The Global Positioning System

(GPS) offers superior time stamp capability, in the order of 340 ns but is unreliable indoors [11]. Thus, for the purposes of this work, methods relying on an external clock were deemed inappropriate.

Ballew, Kuzmanovic and Lee have explored the fusion of audio recordings of concerts made by smartphones [18]. Of special interest for this work is the chosen synchronization algorithm, which uses cross correlation between different audio recordings to determine relative time offset of recorded sounds. However, this cross correlation is not performed with the intent of real-time synchronization but is performed in post-processing. As shown by Knapp and Carter [10], the maximum value of the cross correlation is a maximum-likelihood estimator for the time delay between two input signals.

II.2.2 Sampling rate offset correction

Two methods for sampling rate offset correction were found in the literature, both relying on correlation processing. The first, proposed by Markovich-Golan, Gannot and Cohen [7] is based on the drift of the cross spectrum of two signals sampled at different sampling rates. The second, by Cherkassky and Gannot [12], derives an estimator for the relative sample rate offset between two signals based on a cross-wavelet transform. Both of these approaches are independent of resampling implementations and thus may use a range of resampling methods, including Lagrange polynomial interpolation [7, 17] and spline interpolation [12].

II.3 SRO characterization

An attempt has been made to characterize the degree of sampling rate offset in the used smartphones. Based on the literature (see section II.1.2), it is expected that the offset is quite low compared to the used sampling rate. Therefore, this work has focused on providing quantitative measurements of sample rate offset. Implementation and comparison of correction algorithms was left as a topic of future research.

II.3.1 Measurement setup

In order to measure SRO, the anechoic chamber at the Delft University of Technology was set up with three identical Nexus 5 smartphones streaming audio to a computer at a nominal sampling frequency of 48 kHz. The audio source consisted of a loudspeaker connected to a high-fidelity audio interface, RME Fireface 800. The measurement setup is depicted in Fig. 4 and a picture of the actual setup is shown in Fig. 5.

The audio interface produced an hour-long continuous sine waveform of 15 kHz which was recorded by the smartphones and sent to the computer, with the intent of computing the sample rate offset from the ideal 48 kHz value.

As will become clear in the next section, this offset is better detected when a higher-frequency sine wave is used. However, if the frequency is chosen too high, the microphone gain is reduced to nearly zero. Therefore, 15 kHz was chosen to strike a balance between high frequency and and good microphone gain. The microphone response was taken from the results of Gaubitch, Martinez, Kleijn and Heusdens [2] and preliminary results from Brinkman and De Rooij [15] for Nexus 5 microphone directivities.

II.3.2 SRO estimation

The resolution at which the frequency can be estimated can be calculated from the properties of the discrete Fourier transform (DFT). Let N be the number of time-domain samples in the series $s[t]$, $t \in \{0, 1, \ldots N-1\}$.

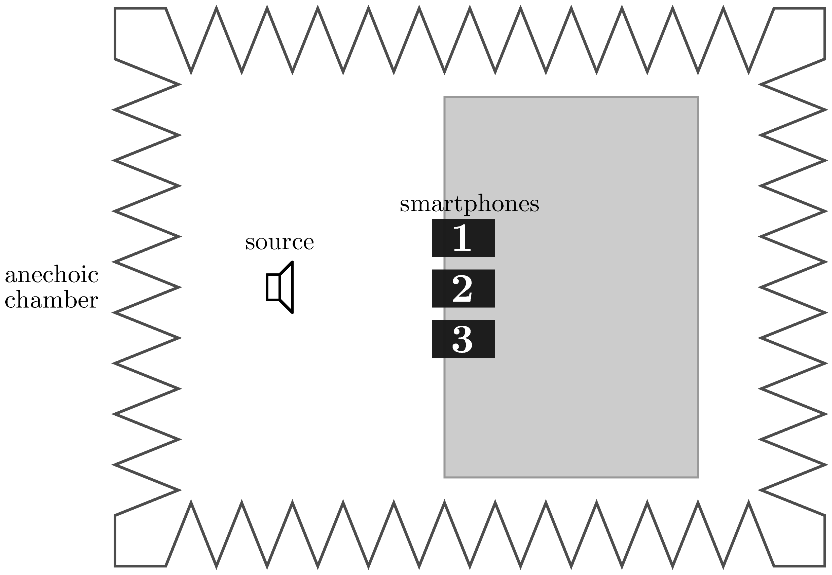


Fig. 4. Schematic representation of the measurement setup used for SRO measurements. The figure shows the anechoic chamber at the Delft University of Technology with three smartphones and a loudspeaker. Not depicted: computer connected via Wi-Fi to receive the sent audio signals.

The DFT is then the frequency-domain representation of the periodically extended discrete-time signal $\boldsymbol{s}[t]$, and is given by $\boldsymbol{S}[2\pi k/N],\ k \in \{0, 1 \ldots N-1\}$. Thus, we also have N samples in the frequency domain [19, p. 421-422]. Recalling that the DFT consists of frequencies from $-f_s/2$ to $f_s/2$ (assuming the time-domain signal was bandlimited below $f_s/2$), the frequency resolution can be calculated from $\Delta f = f_s/N$. But since $N = Tf_s$, $\Delta f = 1/T$ where T represents the duration in seconds of the sampled signal $s[t]$.

It can be seen that a short DFT window provides fine time resolution but coarse frequency resolution, and conversely a long DFT window provides fine-grained frequency resolution at the expense of coarse time resolution. Put differently, a long DFT window provides more detailed information about the frequencies present in a signal, but provides less detail about *when* these frequencies are present in time.

For these experiments, a sampling rate of 48 kHz was chosen as this is the highest widely supported sampling rate on Android smartphones. Window lengths varying from 5 minutes ($\Delta f = 3.3$ mHz) to 30 seconds ($\Delta f = 33$ mHz) were chosen for performing the DFT.

Once the recorded frequency is known, the SRO can be determined:

$$\text{SRO} = \frac{\left(f_{\text{play}} - f_{\text{detected}}\right) \cdot f_s}{f_{\text{play}}}$$

where $f_{\text{play}}, f_{\text{detected}}, f_s$ indicate played audio frequency, detected audio frequency from the recording and sampling frequency, respectively.

II.3.3 Results

The SRO for these smartphones is lower than predicted by the literature, in the order of 0.2 Hz (Fig. 6) and does not change rapidly with time. Full DFT plots for all window lengths may be found in Fig. 14 in Appendix

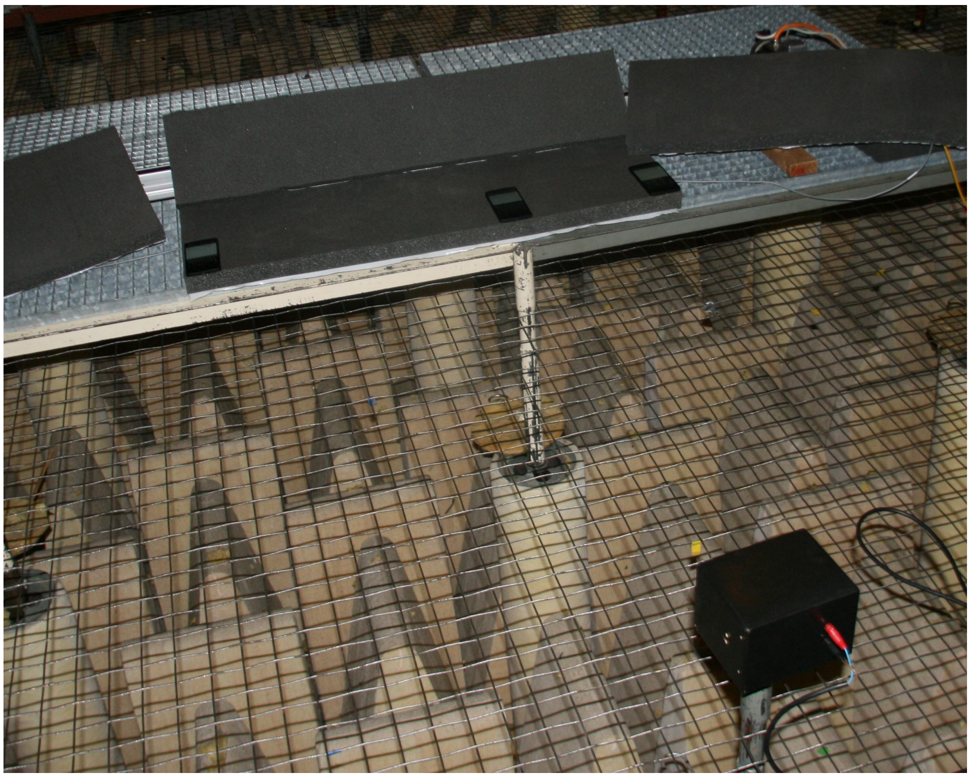
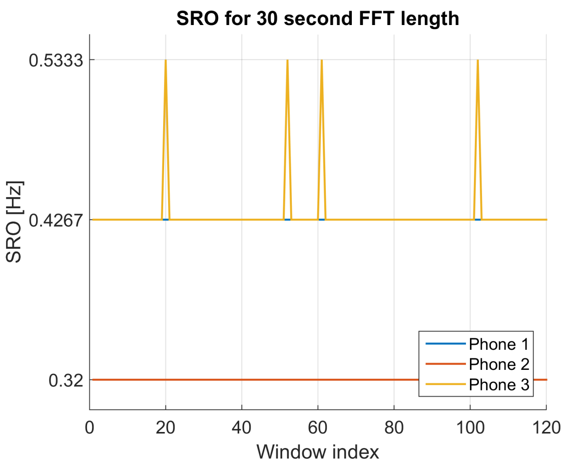
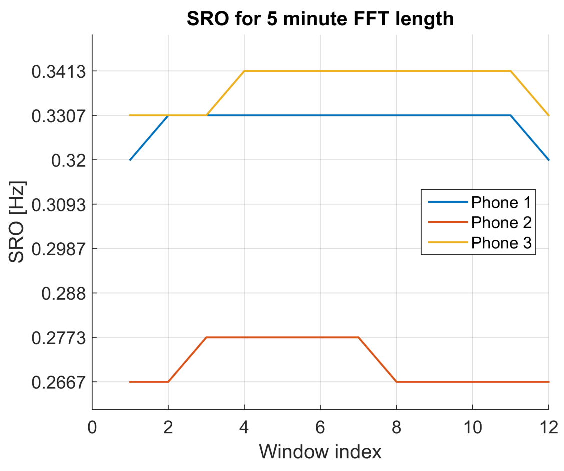


Fig. 5. An actual picture of the setup in the anechoic chamber. The three phones are resting on a piece of foam in the top half of the image, with the speaker visible in the lower right.

A and a discussion of the results in section V.1.1.



(a) 30 second DFT window, $\Delta f = 33$ mHz.



(b) Five minute DFT window, $\Delta f = 3.3$ mHz

Fig. 6. Sampling rate offset measurements for two DFT window lengths. A larger DFT window gives a higher frequency resolution Δf but lower time resolution.

II.4 Algorithm

This section presents the two approaches for time offset correction that were implemented for this work. Both of these approaches rely on the transmission of a 'beacon' signal with known properties, then correlating the received signals with this beacon. This technique is widely used in computing acoustic impulse responses [20, 21], and is a maximum-likelihood estimator for the delay between two sequences [10].

II.4.1 Time offset correction

Beacon signal

In order to robustly estimate the time delay using cross correlation, it is desirable for the beacon signal to have four properties: first, the correlation operation should yield a prominent maximum when performed on the beacon signal, even when noise is introduced into the system. Second, the beacon should emit a wideband signal such that e.g. a resonant dip in acoustic frequency response will not disproportionately affect the synchronization procedure. Third, a long time support for the beacon lessens the effect of transient disturbances on the synchronization system. Finally, a low ratio of peak-to-average power is desirable, as it lowers demands on the audio playback and recording systems.

For this work, the decision was made to use a maximum length sequence (MLS). A maximum length sequence is a waveform generated by a linear feedback shift register (LFSR) designed to produce the longest possible sequence for a given amount of state information: with N bits of state, it yields a sequence of length $2^N - 1$. This sequence fits all four criteria listed above: its autocorrelation approaches the Kronecker delta function, it is spectrally flat in a wide frequency range (restricted by the signal's finite length and sample rate), the signal has configurable time support and finally a low ratio of peak-to-average power [21].

These properties of MLS make it a good candidate for (cross-)correlation processing, which is described below.

Cross correlation

Both the time- and frequency-domain synchronization algorithms rely on the operation of cross correlation. For discrete-time, real-valued signals, cross correlation is defined as follows [22, p. 228]:

$$(\boldsymbol{s} \star \boldsymbol{p})[n] = \boldsymbol{s}[-n] * \boldsymbol{p} = \sum_{m=-\infty}^{\infty} \boldsymbol{s}[m] \cdot \boldsymbol{p}[m+n] \tag{II–2}$$

If $\boldsymbol{p}$ and $\boldsymbol{s}$ are the same maximum length sequence, this cross correlation approaches a delta function [21]:

$$(\boldsymbol{s} \star \boldsymbol{p})[n] = (\boldsymbol{p} \star \boldsymbol{p})[n] \approx c \cdot \boldsymbol{\delta}[n]$$

where c is a scaling factor dependent on the MLS amplitude and length.

If $\boldsymbol{s}$ is now convolved with an impulse response $\boldsymbol{h}$, the associativity and commutativity properties of convolution [22, p. 169] can be used to show that the cross correlation approaches that same response:

$$((\boldsymbol{p} * \boldsymbol{h}) \star \boldsymbol{p})[n] = (\boldsymbol{p} * \boldsymbol{h}) * \boldsymbol{h}[-n] = \boldsymbol{p}[-n] * \boldsymbol{p} * \boldsymbol{h} \approx c \cdot \boldsymbol{\delta} * \boldsymbol{h} = c \cdot \boldsymbol{h}$$

To implement both algorithms, a pulse signal $\boldsymbol{p}$ is first generated. This pulse is played back over a speaker to the phones, where each phone i records an acquired signal $\boldsymbol{s}_i$. Each acquired signal is then the original pulse $\boldsymbol{p}$, convolved with an impulse response $\boldsymbol{h}_i$. This $\boldsymbol{h}_i$ represents the effects of the audio playback system, the room acoustics, the microphone response and the unsynchronized recording time offset:

$$\boldsymbol{s}_i = \boldsymbol{p} * \boldsymbol{h}_i = \boldsymbol{p} * \boldsymbol{h}_{\text{speaker}} * \boldsymbol{h}_{\text{i,acoustic}} * \boldsymbol{h}_{\text{i,mic}} * \boldsymbol{h}_{\text{i,delay}}$$

It is assumed that each of the speaker and microphone impulse responses and the room impulse responses $\boldsymbol{h}_i$ consist of damped LTI systems. Although this assumption does not model all characteristics of room impulse responses, it is a common simplifying assumption in other literature [23].

The LTI model predicts a prominent initial peak in $\boldsymbol{h}_i$. The acoustic impulse response is modelled with a propagation delay t_p, as well as damped repetitions resulting from surfaces in the room reflecting sound waves. The delay impulse response $\boldsymbol{h}_{\text{i,delay}}$ is assumed to contain only a time delay of $\Delta t_{\text{receive}}$, as explained in section II.1.

Based on these assumptions, the expected compound, time-domain impulse response consists of a prominent global maximum, and its scaled repetitions, smeared by the speaker and microphone impulse responses, and delayed by the combined propagation delay and time delay.

Time-domain cross correlation

The first implemented algorithm performs a time-domain cross correlation of $\boldsymbol{p}$ and $\boldsymbol{s}_i$ to obtain a signal $\boldsymbol{c}_i$ as in equation II–2. This signal constitutes an estimate of the total system impulse response $\boldsymbol{h}_i$, and is expected to contain the same delayed global maximum described above. This maximum is located in each source signal $\boldsymbol{c}_i$ by a search, and its index $\hat{\tau}$ is used as an estimate of the total time delay $t_p + t_d$:

$$\hat{\tau} = \arg\max_n \ (\boldsymbol{p} \star \boldsymbol{s}_i)[n] \tag{II–3}$$

Frequency-domain cross correlation

In addition, a similar algorithm was implemented in the frequency domain to evaluate potential benefits.

First, both the pulse $\boldsymbol{p}$ and each recorded signal $\boldsymbol{s}_i$ are transformed by means of a fast Fourier transform (FFT) to yield signals $\boldsymbol{P}$ and $\boldsymbol{S}_i$ in the discrete-frequency domain. Next, these signals are cross correlated in the frequency domain. In the discrete-frequency domain, cross correlation can be written as an element-wise multiplication of the arguments [24, p. 310]:

$$\boldsymbol{s}_i \star \boldsymbol{p} = \boldsymbol{s}_i[-n] * \boldsymbol{p} \circ\!\!-\!\!- \mathscr{F}\{\boldsymbol{s}_i \star \boldsymbol{p}\} = \boldsymbol{S} \odot \overline{\boldsymbol{P}} \tag{II–4}$$

Since a time-domain delay τ maps to a linear phase shift $-2\pi k\tau/N$ in the discrete-frequency domain, the phase of the resultant signal consists of this frequency-dependent phase shift plus the phase of the input signal:

$$\boldsymbol{f}[t-\tau] \circ\!\!-\!\!- e^{j\omega\tau}\mathscr{F}\{\boldsymbol{f}[t]\} \Rightarrow \arg(\mathscr{F}\{\boldsymbol{f}[t-\tau]\}) = \arg(\mathscr{F}\{\boldsymbol{f}[t]\}) - \frac{2\pi k}{N}\tau \tag{II–5}$$

where $2\pi k/N,\ k \in \{0, 1 \ldots, N-1\}$ are the discrete frequencies. As a result, it is possible to estimate the delay τ by estimating the linear dependence of the phase shift on the frequency, so long as the phase contribution of the room impulse response, $\arg(\mathscr{F}\{\boldsymbol{f}[t]\})$, is low compared to the delay-induced phase shift.

Therefore, the frequency-domain algorithm computes the phase angle of the frequency-domain cross correlation (equation II–4). Since this angle is confined to the interval $[-\pi, \pi]$, it is first 'unwrapped' around each discontinuity to provide a linear phase $\boldsymbol{\phi}[n]$ as function of discrete frequencies. Then, the algorithm attempts a least-squares fit of a line $\hat{\boldsymbol{\phi}}[n] = n\hat{\Gamma} + \hat{\Theta}$. Γ is then a measure for linear phase shift is computed as a slope in radians per FFT index. Finally, it is divided by 2π and multiplied by the FFT size L, to obtain corresponding time shift $\hat{\tau}$ measured in samples:

$$\hat{\tau} = \hat{\Gamma} \cdot \frac{L}{2\pi}$$

II.5 Simulation

To characterize the performance of just the synchronization subsystem, a simulation was constructed that measures the error of the estimate $\hat{\tau}$ in different scenarios. This simulation is performed for different values of synchronization delay, reverberation and signal-to-noise ratio.

II.5.1 Scope and implementation

First, the simulation generates the MLS signal that is used for all simulations. Based on this, a number of microphone signals are generated. Each signal is convolved by K different FIR filters, each representing a room impulse response for a 5 by 6 by 2.5 metre cuboid-shaped room with a variable wall acoustic reflection coefficient[1]. Each of these signals is delayed by L different delays, yielding a $K \times L$ set of convolved and delayed sample signals.

Then, for each delay and reflection value, M different signal-to-noise ratios are evaluated. For each SNR, N realizations of Gaussian noise at that magnitude are generated so that, in total, $K \times L \times M \times N$ simulation runs were performed. This noise realization is added to the pre-generated sample signal and both the time- and frequency-domain correlation algorithms described above are run on this noisy signal. The difference between the actual inserted delay and the estimated delay is computed and stored in a result vector.

[1]These FIR filters were computed by the rir.m file available from the MATLAB Central File Exchange at `https://www.mathworks.com/matlabcentral/fileexchange/5116-room-impulse-response-generator/content/rir.m`

The parameters used for this simulation are as follows: $K = 10$ different wall reflection coefficients from 0 to 1 (unitless), $L = 10$ different delays from 0 s to 0.2 s, $M = 20$ different noise signal-to-noise ratios from 60 dB to -34 dB and $N = 100$ noise realizations for each parameter combination.

II.5.2 Expected results

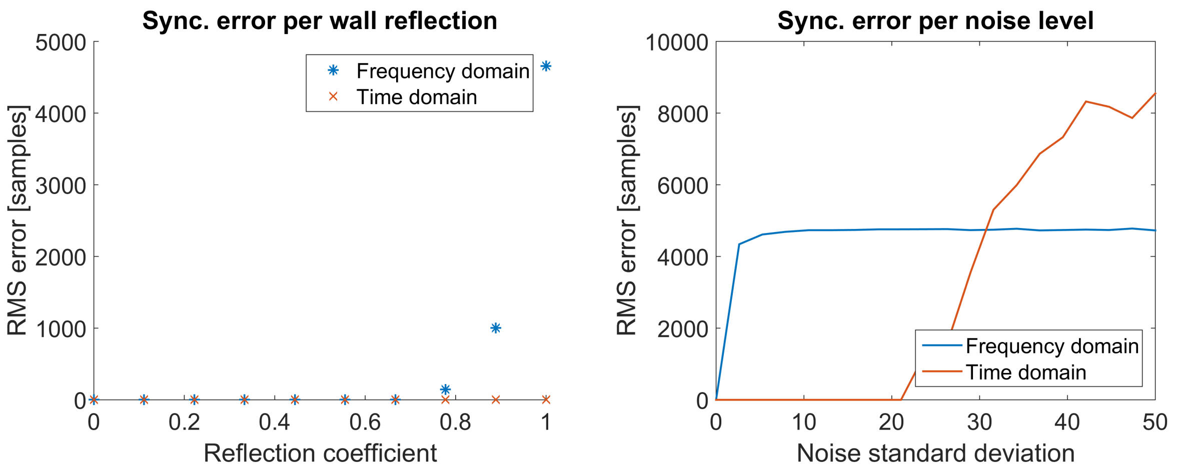
Due to the favorable properties of the beacon signal described in II.4.1, it is expected that the time-domain algorithm will be robust in the face of introduced noise. As this algorithm relies on finding a global maximum in the impulse response, however, it is suspected that for high values of wall reflectance, combined with multipath interference, a maximum in the impulse response might arise that does not correspond to the direct acoustic path. This would lead to a large estimation error beyond a certain critical wall reflection coefficient. In addition, the time-domain algorithm is only accurate to integer samples, leading to an expected round-off error of less than one sample.

The frequency-domain algorithm is expected to provide sub-sample accuracy due to its least-squares estimation of the delay in the frequency domain. However, performance is reliant on low phase contribution of the acoustic impulse response compared to the contribution of the delay that must be estimated (equation II–5). It is unknown whether this assumption will hold throughout the simulation scenarios.

It is not expected that the performance of either algorithm will depend on the amount of introduced delay.

II.5.3 Results

The reflection coefficient has no influence on the performance of the time-domain algorithm, but the frequency-domain algorithm is crippled at reflection values of 0.8 or higher (Fig. 7a). The frequency-domain algorithm is much more susceptible to noise than the time-domain algorithm (Fig. 7b). Neither algorithm is affected by the amount of delay (Fig. 13c in Appendix A.II). A discussion of the results may be found in section V.1.3.



(a) Delay estimation error as a function of wall reflection. **(b)** Delay estimation error as a function of noise power

Fig. 7. Simulated results for TD and FD synchronization algorithm.

Chapter III

System implementation

This chapter describes the implementation details of the audio acquisition subsystem. The purpose of this subsystem is to acquire the microphone audio from the smartphones and bring it to a centralized location where it can be processed and combined.

As was explained in chapter I, the subsystem in question consists of two parts. The first is a server application that accepts audio data from a number of smartphones, synchronizes it, then makes it available to the subsequent beamforming system. The second is an Android application that connects to this server application, after which it can be instructed to start streaming audio data captured from its microphone(s), as well as orientation data that can be used in the audio processing algorithms.

III.1 State of the art

III.1.1 Android OS

The Android operating system is the most popular mobile platform globally, with a market share of nearly 80% at the time of writing [25]. Android is not a real-time operating system (RTOS) [26], and thus has no timing guarantees on task completion. However, Moore's Law scaling has led to huge performance boosts in computers in general and smartphones in particular in the last five years which might mean smartphones are already fast enough for real-time audio streaming. Since Google provides a development kit in the form of the SDK [27], performance issues can easily be assessed on the used hardware. Furthermore, if performance remains an issue, Android has a Native Development Kit available to access low-level functionality in C or C++ [28]. Android native development may have performance increases, but is not recommended as a starting point for general-purpose applications [29].

III.1.2 Smartphone localization

While necessary for synchronization, the smartphone locations are also required to perform beamforming and calculate directivities of the microphones. There are several ways of localization for smartphones: GSM multilateration and GPS sensors use hardware supported by a broad range of smartphone models. GSM based localization is accurate up to 5 meters indoors or 75 meters outdoors [30] while GPS typically does not work indoors [11]. A solution for indoor acoustic self-localization on smartphones is discussed by Henneke and Fink, who achieve root-mean-squared errors of 6 cm based on an array with 40 cm diameter [8].

III.1.3 Orientation

For gain calculation from directivity measurements, the orientation of the microphone is needed as an input. Smartphones typically have two sensors to measure orientation relative to the Earth: a compass and an accelerometer [31]. These measurements provide an azimuth, pitch and roll angle, defined in Fig. B. Even though the raw output obtained when reading these sensors on Android is inaccurate, an extended Kalman filter (EKF) may be applied to correct some noise, leading to an error of around 10 degrees [13]. As the measurement setup described in [13] is not necessarily representative of the usage scenario covered in this work, it was decided to evaluate the performance of Android's orientation estimation functionality in a more representative environment.

III.2 Android app

For acquiring the microphone signals, as well as auxiliary data such as the smartphone orientation, a smartphone application (or 'app') was developed on the Android platform.

In the context of the Android Application Programming Interface[1], this application consists of two parts. The first is an Activity, which is started by the user and can be interacted with using the phone's display, and the second is a Service, which runs independently from the Activity so that its operation can continue if the phone's display is off or another Activity has focus.

A screenshot of the Activity is shown in Fig. 8. It shows the phone ID as sent along when connecting to the MATLAB server, and provides fields for setting the hostname and port of that server. After this information has been entered, the user can press the 'Start service' button to start the Service in the background, and the 'Stop service' button to terminate it again.

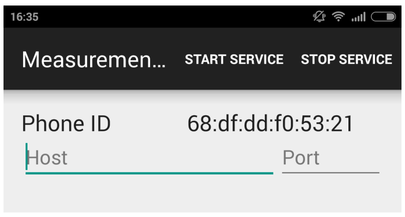


Fig. 8. Partial screenshot of the Activity user interface

The Service consists of four separate classes. There is a RecordService that provides the main Service seen by the Android OS. It handles start and stop requests from the Activity or the OS itself, as well as some convenience methods for notifying the user and interacting with the Android user interface. The Service sets itself up as a so-called 'foreground service', which implies that it is a purposefully started activity that should take priority over background processes when the system is low on resources.

The bulk of the application is distributed over three threads started by this Service. The NetworkInterfaceThread handles the connection to the MATLAB server and loops to receive new messages, as well as sending messages generated by the other threads. As there is only one connection to the server, it was decided to use Java's normal blocking socket system, meaning the thread blocks (and other threads can be run) while no new data has been received.

[1]The Android Application Programming Interface (or 'API') is referenced throughout this section. Documentation for this API is available on-line at `https://developer.android.com/reference/packages.html`.

The NetworkInterfaceThread follows the same state machine described in section III.3.1 for the MATLAB server. When the thread is started, its state is SETUP. When it has received a packet containing the audio settings for this session, it transmits an acknowledgement packet and changes state to IDLE. From this point on, it will process a command to start streaming by going into the STREAMING state, at which point it launches an AudioThread to acquire data from the on-board microphone. During both the IDLE and STREAMING states, an OrientationThread is run to acquire orientation data, which is also transferred to the MATLAB server.

The AudioThread uses the Android AudioRecord API to initialize the device microphone. As the Android API defines a number of different audio sources meant for different applications [2], it accepts an input argument specifying the audio source to use as a byte. Once the audio source has been successfully initialized, the thread continuously does a blocking read from the audio device into a buffer of its own. Once it has enough samples to transmit a full block (of the length sent by the server), it calls a method in the NetworkInterfaceThread to initiate this transmission.

The OrientationThread uses the Android SensorManager API to request access to the device magnetometer and accelerometer. The magnetometer produces a vector in the direction of the local magnetic field, and the accelerometer produces a vector in the direction of the so-called 'proper acceleration', taking into account the balance of gravitational force and restoration force of the table surface. As such, the magnetometer should produce a vector that points north, and the accelerometer should, at rest, produce a vector that points down.
These vectors are then processed using Android's provided SensorManager.getRotationMatrix to yield a rotation and inclination matrix. These matrices represent the transformation from the device coordinate system (in which the accelerometer and magnetometer measure) to a global coordinate system where X points east, Y points towards magnetic North, and Z points towards the ground. In Appendix B Fig. 16, this coordinate system is shown graphically. Finally, the SensorManager.getOrientation method is then called on this matrix to express this transformation in an azimuth, pitch and roll angle that can be sent to the server.

III.3 MATLAB implementation

The possibility to call compiled Java `.class` files from within MATLAB can be used to leverage Java's asynchronous networking capabilities from within a MATLAB script. In this section, the implemented Java/MATLAB host application will be detailed and design choices will be explained.

The choice for MATLAB was made based on the ease of prototyping, prior experience and interconnections with the subsystem built by van Wijngaarden and Wouters [14]. Furthermore, compiled Java `.class` files can be called from MATLAB to extend default MATLAB functionality.

Although MATLAB was found to provide a suitable environment for developing this application, it was found that interoperation between Java code and MATLAB itself is not entirely trivial. The largest problem is that the ability to call back arbitrary MATLAB functions from Java, while present, is not officially supported or documented[3]. Using this functionality is therefore somewhat risky, since MathWorks may remove this functionality without any indication to users.

[2]For more information, see `https://developer.android.com/reference/android/media/MediaRecorder.AudioSource.html`.
[3]Yair Altman keeps a blog called Undocumented Matlab where he explains how MATLAB callbacks can be implemented in Java, among other undocumented features: `http://www.undocumentedmatlab.com`

Despite previously mentioned concerns, it was decided to use Java for this project because MATLAB does not natively provide non-blocking networking to multiple clients. In Java, the `java.nio` package developed by Oracle[4] allows multiple simultaneous transmission control protocol (TCP) connections over a single port, enabling multiple smartphones to connect to the Java program. The connections can then be multiplexed using `selector`. The reason TCP was chosen instead of the user datagram protocol (UDP) is the connection-oriented nature of TCP, which provides guaranteed in-order reception of packets. This simplifies consequent signal processing, which no longer has to account for possibly missing audio data.

Using a single processing thread would be detrimental to this server application since the application could then either send/receive data to connected clients *or* process the received audio. In order to parallelize these concurrent activities, the Java classes utilized parallelization in the form of `java.lang.Thread` threads. One thread is used to perform non-blocking I/O and the other thread is the default MATLAB thread used to process incoming audio samples.

III.3.1 I/O finite state machine

Since the I/O is non-blocking, several clients (in this case, smartphones) can read and write to the server simultaneously. Therefore, to facilitate communicating with multiple devices simultaneously, the server application represents each client as a finite state machine (FSM). From the server's point of view, each client is in one of the following states (see also Fig. 9):

CONFIGURING	The client has initiated a connection to the server and the client and server are currently negotiating settings[5].
IDLE	The client is connected and fully configured. It is awaiting a start streaming command to start streaming its audio recordings to the server. Additionally, the settings may be renegotiated to change the state back to CONFIGURING.
STREAMING	The client is currently streaming audio to the server. Sending a stop streaming signal will change the client state to IDLE.

Additionally, because a complete packet sent by a client may arrive as several frames on the server side, each client can be in one of two reading states:

RECEIVINGDATA	This state indicates to the server that the client is somewhere along sending a packet to the server. The next read frame should be interpreted as a continuation of the previous frame(s).
IDLE	This state indicates the next received frame should be interpreted as the start of a new packet.

III.3.2 Communication protocol

On top of the TCP interface provided by the Java `SocketChannel`[6], a custom communication protocol between the MATLAB application and the clients was defined. Each packet starts with one byte identifying its

[4]Documentation for the NIO package can be found at
`https://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html`

[5]The negotiated settings are sample rate, block length (number of bytes sent at once) and mono/stereo recording.

[6]Documentation for the SocketChannel can be found at `https://docs.oracle.com/javase/7/docs/api/java/nio/channels/SocketChannel.html`

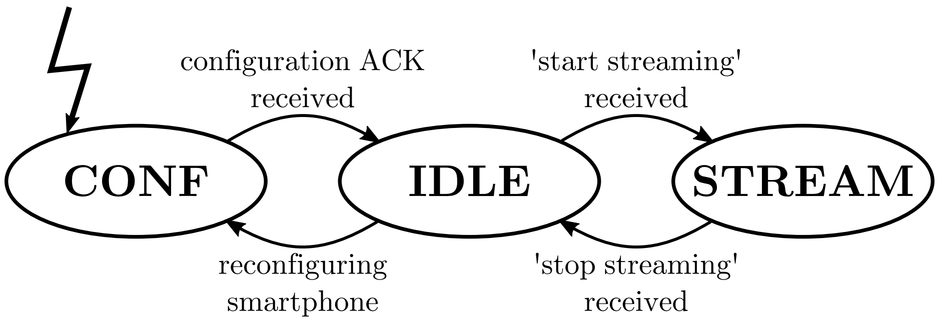

configuration ACK received
'start streaming' received
CONF
IDLE
STREAM
reconfiguring smartphone
'stop streaming' received

Fig. 9. Diagram representing the finite state machine implemented by both the server and the smartphone application.

type to the recipient. A list of each byte-identifier and packet type in the communication link along with their content is listed below.

1: SETUP	Sent from server to client. Contains 4 bytes specifying the sampling rate to use, 4 bytes containing the block length to use and 1 byte indicating whether to record in mono or stereo.
2: ACK_SETUP	Sent from client to server after receiving SETUP. Contains a string with the hardware (MAC) address of the client preceded by 4 bytes specifying, in bytes, the length of the string.
3: START_STREAMING	Sent from the server to the client, indicating the client should start streaming its audio recordings to the server. Contains 1 byte specifying which audio source to use, as explained in section III.2.
4: STOP_STREAMING	Sent from the server to the client, indicating the client should cease streaming its audio recordings to the server. Packet has no content.
5: ACK_STOP	Sent from the client to the server to acknowledge it has received the stop packet and will stop streaming audio. Packet has no content.
6: STREAM_DATA	Sent from client to server, contains recorded audio samples. The length of this packet is block_length $*$ 16 bits (each sample has 16 bit resolution). The block length is negotiated in the SETUP packet.
7: ORIENTATION_UPDATE	Sent from client to server, contains 12 bytes (3 floating point numbers) with measured azimuth, pitch and roll. A final byte indicates whether the phone is moving[7].

III.4 System performance

III.4.1 Android

Despite the lack of real-time guarantees from the Android kernel and the overhead inherent to Java applications, no performance problems were encountered in the Android data acquisition application.

Unfortunately, the Android `AudioRecord` API does not make adequate provisions for querying the supported audio recording settings for a given device. The suggested way of determining whether a given mode

[7]This byte is not used on the MATLAB side.

is supported is to attempt to initialize using these settings, and catching any thrown exceptions as well as detecting whether the resulting object was successfully initialized. Unfortunately, this process is unreliable, and it was found that while testing, certain phones successfully initialized the AudioRecord object but e.g. yielded duplicated mono data when a stereo signal was requested, or yielded corrupted data.

In addition, the processing performed for each configurable audio source varies per device. Certain devices turn on specialized signal processing that negatively affects the beamforming processing, such as automatic gain control or noise reduction algorithms. Android unfortunately lacks an adequate API for requesting audio sources with a certain amount of processing performed or querying the processing performed for each audio source. Experimentation showed the voice recognition input (`AudioSource.VOICE_RECOGNITION`) generally yielded the least amount of processing, but even this was not guaranteed in all cases.

III.4.2 Server application

Although no attempt was made to quantify the performance of the server application, it can be said that no problems were encountered with the server application, with up to 6 simultaneous phones streaming their audio recordings at once. Furthermore, all packets described in section III.3.2 are correctly encoded, decoded and interpreted. The orientation measurements are passed on to a function translating them into azimuth and elevation estimates. Recorded audio data is passed on to a buffer structure used in the beamforming system [14].

One feature of the server implementation is that it does not use any functions specific to MATLAB or Java (e.g. functions that are not available in other common programming languages). This ensures the application may be ported to a different programming language, although performance was found to be adequate for the purposes of this system. A major advantage of using MATLAB was the ease of interfacing with the beamforming subsystem, which also runs in MATLAB [14].

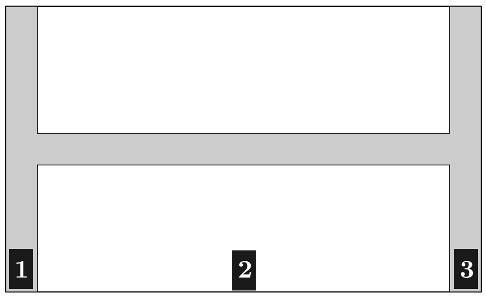
For a future, more fully-fledged product, the use of MATLAB should be reconsidered. Performance issues might arise when more complex operations are performed simultaneously within the server application, and the interface between MATLAB and Java that was used is not officially supported.

III.5 Orientation measurement

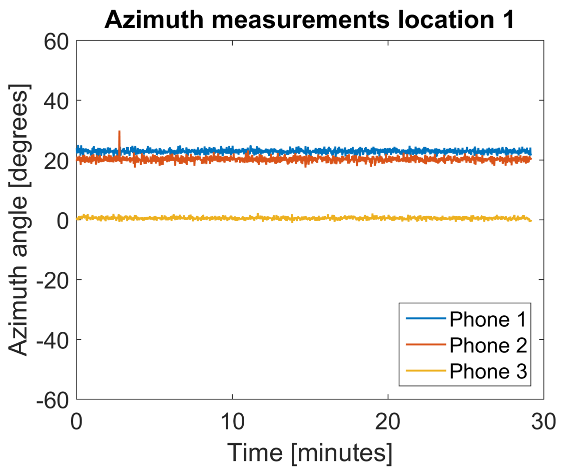
As the beamforming algorithm relies on directivity data that varies with the orientation of the phone, it is necessary to determine the orientation of each smartphone. As no research was found detailing the accuracy of smartphone orientation sensors on a table in an office environment, it was deemed necessary to characterize the accuracy of the orientation sensors in the smartphones in this situation.

The experimental setup is shown in Fig. 10a. A table was set up in an office environment. Three smartphones were placed in three locations on the table, two of which were located above the metal leg assembly of the table. For each location, the smartphones were placed in the same orientation by aligning them with the edges of the table. Every second, the orientation of each phone was computed based on its internal sensors, in the manner described in section III.2. Next, this data was transmitted to a computer running the server application described in section III.3. This orientation was logged for 30 minutes per run, after which the phones were switched to another of the three locations. The experiment was repeated three times to acquire data for each phone in each position.

Representative results for the computed azimuth are shown in Fig. 10b. The reported azimuths are reasonably constant over the measurement interval but individual computed azimuths already vary by over 20 degrees. Full results may be found in Fig. 12 in Appendix A.I. A discussion of the results may be found in section V.1.2.



(a) Schematic representation of the measurement setup used for orientation measurements. The figure shows the top view of a table with measurement locations indicated by the numbers. The grey area indicates the location of metal support bars of the table.



(b) Azimuth angles for location 1.

Fig. 10. Measurement setup for orientation estimation and azimuth angles from three smartphones. The azimuth is quite constant, but a large offset is present between different sensor readings at the same location. Full results for all smartphones and locations can be found in Fig. 12 in Appendix A.I.

Chapter IV

System test

In order to evaluate the performance of the complete system, an experiment was devised to quantify the error in synchronization between different smartphones. This chapter describes the experimental setup and results.

IV.1 Set-up

The setup for the experiment is shown in Fig. 11a: two smartphones are placed on a table in a room representative of a reverberant office environment. A loudspeaker was also placed in the room, acting as a source of the synchronization signal and an additional background signal. The placement of the loudspeaker, l_1, was chosen such that smartphones m_1, m_2 were equally spaced (within measurement error) from speaker l_1.

To perform an experiment, the smartphones were first placed in the STREAMING state using the MATLAB server application. Next, a background signal (silence, voice or music) was played over speaker l_1. After several seconds, the beacon signal (see also section II.4.1) was played, and the time-domain synchronization algorithm (described in section II.4) was invoked on the audio recordings transmitted to the MATLAB application up to that point. Recording was continued for approximately ten seconds after the pulse, then halted by sending a STOP_STREAMING packet to both smartphones.

This experiment was repeated for three background signals (silence, voice and music) and two amplitudes of the beacon signal (100% and 20% of full scale, respectively). For both beacon signal amplitudes, the amplitude of the background signal was left at the same level.

IV.1.1 Expected results

Based on the simulation data described in section II.5, the recordings on m_1 and m_2 played from l_1 are expected to be synchronized within one sample of one another. However, error in the placement of the loudspeaker and smartphones causes an unknown arrival time difference that contributes linearly to the synchronization error. For example, if the accuracy of placement of the smartphones and loudspeakers is $\Delta d = \pm 1$ cm, the maximum synchronization error ϵ is

$$\epsilon = \pm f_s \frac{4\Delta d}{v} = \pm 48 \cdot 10^3 \frac{0.04}{343} = \pm 5.6 \text{ samples}$$

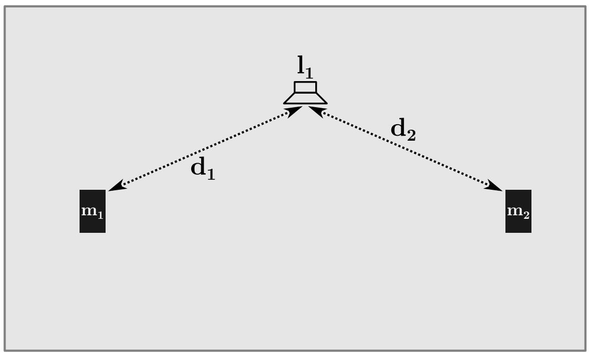
with $v = 343$ m/s the speed of sound in air and $f_s = 48$ kHz the used sampling frequency.

Furthermore, an error in the speed of sound leads to a TO mismatch linear with the TDOA error. For example, let the expected TDOA be 10 ms, (corresponding to a distance of approximately 3.4 m) and let the error in speed of sound be 1%. Then, the actual TDOA lies between 9.9 and 10.1 ms. This 0.2 ms margin leads to a margin of 9.6 samples. TDOA compensation is not performed in this experiment.

Based on the above remarks, it is expected that the synchronization will be up to 15 samples off, assuming a 1 cm error in position per smartphone and loudspeaker and perfect synchronization algorithm. It is also expected that playing the beacon signal at a higher volume will yield a lower error, as the ratio of the beacon amplitude to the background amplitude is higher.

IV.2 Results

Results of the system test are shown in Fig. 11b. It can be seen that contrary to the expected results, the algorithm performed better (with errors of 5 and 6 samples) in the low-volume case than in the high-volume case (where the synchronization error was 26 samples). Further discussion of these results may be found in section V.1.4.



(a) An overview of the system test setup.

Description of background noise	**Error** [samples]
Silence	26
Voice	26
Voice (low MLS volume)	5
Music (low MLS volume)	6

(b) Synchronization error for different scenarios.

Fig. 11. Measurement results for real-life synchronization tests. Fig. 11a shows the set-up.

Chapter V

Discussion and recommendations

This chapter contains a discussion of the various results obtained in this work, followed by a number of suggestions to improve the various aspects of the system, as a basis for future research. Finally, an overarching conclusion is given regarding the presented results.

V.1 Discussion

V.1.1 SRO measurements

A measurement of the sample rate offset (SRO) present in the smartphone was performed. This test is described in section II.3. Complete results are shown in Fig. 14 in Appendix A.

The effect of sampling rate offset on the audio recordings is quite evident. After an hour of recording – a typical time for a conference call – the phones with the most diverging SRO (phone 2 and 3 in Fig. 6a) will have drifted apart by an order of $60 \cdot 60 \cdot 0.2 = 720$ samples.

The need for resampling is more pronounced than expected and should be compensated in future work. Alternatively, more frequent time-offset correction could be applied. The disadvantage of this approach is that periodic playback of the beacon signal (section II.4.1) could be considered unpleasant. Furthermore, using current smartphone microphone directivities, ultrasonic synchronization beacons are not realistic since the responses of the microphones are near zero at these frequencies.

V.1.2 Orientation measurement

As described in section III.5, the orientation of the smartphones was computed over a period of time to determine the accuracy and stability of these estimates. Results for this rest are shown in Appendix A, Fig. 12.

It can be seen that the orientation estimation problem is exacerbated for all different positions on the table. It is hypothesized that this is due to the difference in metal content near the smartphone, as well as the variable magnetic environment in an indoor environment in general. As the offset for each smartphone is different for each different position, it is expected that this cannot be reliably accounted for by processing the sensor signals.

The pitch and roll angles (Figs. 12d and 12g), defined in Appendix B, show similar offset between phones, but on a different scale. Pitch offset is on the order of 1 degree and roll offset on the order of 0.5 degree. These measurements are more usable for orientation estimation applied to beamforming, but further filtering may be used to compensate for the measured noise levels [13].

V.1.3 TO simulation

In section II.5, a simulated experiment was performed to characterise the performance of the proposed synchronization algorithms in a variety of different scenarios. The results of this simulation can be found in Appendix A, Fig. 13.

In Fig. 7a, the root-mean-square (RMS) error over a number of runs is plotted as a function of the different reflection coefficients used in the simulation. This plot was made for a simulated SNR of 60 dB and a delay of 0.1 s. It can be seen that the reflection coefficient has no influence on the performance of the time-domain algorithm. The effect on the frequency-domain algorithm is similarly small, until the reflection coefficient reaches a value of about 0.8. At this point, the amplitude of the initial peak is about 2.8 times the amplitude of the largest reflection and the least-squares fit in the frequency domain no longer finds the true delay. It is hypothesized that this performance degradation occurs because, as described in II.4.1, for this algorithm it is assumed that the phase contribution of the acoustic impulse response is low compared to the phase contribution of the delay.

Fig. 13c shows the RMS error over N runs versus the delay that was inserted in the simulation. This plot was made at a simulated signal-to-noise-ratio of 60 dB and a wall reflection coefficient of 0. In this case, it can be seen that the error is limited to within one sample, showing that while the algorithm does not perform with sub-sample accuracy, in an idealized scenario it performs with error close to this one-sample limit.

Finally, Fig. 7b shows the RMS error as a function of the standard deviation of the added Gaussian noise. The frequency-domain algorithm quickly gains a large error as the noise power increases. In contrast, the error of the time-domain estimator only starts to increase around a noise variance of 20 – as the variance in the original pulse is 1, this is equivalent to a signal-to-noise ratio of -26 dB. It is not expected that such low signal-to-noise ratios will be encountered during normal system operation.

V.1.4 System test

A test of the entire subsystem was performed in order to test the synchronization algorithm in a more realistic scenario. This test is described in chapter IV. The results of this test are shown in table 11b of that chapter.

The magnitude of the error was close to the expected results. However, while it was expected that a quieter beacon would yield a higher error, the results indicate that the algorithm performs better (showing a synchronization error of approximately 5 samples instead of 26 samples) when the beacon signal is played back at 20% of full scale. It is hypothesized that this is due to distortion occurring in the signal path at full-scale playback, yielding a distorted pulse that does not match the intended reference signal. When the signal is later correlated with this intended reference signal, the performance is degraded.

As such, it is recommended that future tests of this system account for the effect of distortion on the pulse signal.

V.2 Future work

In light of the results presented here, there are a variety of possible future projects that could be based on this work.

First of all, the Java server application and MATLAB signal processing are not yet optimized for performance. As such, they could be reimplemented in for example the C programming language to run natively on the host computer. This would likely improve the power efficiency of the software, as well as facilitating a real-time reimplementation of the beamforming system described by Van Wijngaarden and Wouters [14].

In addition, while it was decided not to compensate for sample rate offset in this work, such compensation could be integrated in the system to evaluate performance gains.

It may be worth evaluating the merit of cross correlating the different smartphone microphone signals instead of correlating each signal with a known reference. The advantage of this is that in this case, it may be possible to use arbitrary audio as a synchronization reference instead of a predetermined beacon signal. The disadvantage is the increased complexity caused by the unknown correlation properties of arbitrary audio.

It was also shown that the orientation sensing features present in the Android application programming interface were not suitable for our assumed usage scenario. As such, further research into these sensor technologies, signal processing algorithms to improve their precision, and alternative methods of orientation estimation are warranted.

As the smartphone locations were presumed known, another avenue for improvement lies in integrating existing audio-based localization algorithms [8] into the system.

Finally, the need for a centralized processing server could be avoided by instead implementing a set of algorithms on the smartphones which, using message passing, distributes the processing over the smartphones themselves. This approach, however, would require a significantly different architecture from the model presented in this work.

V.3 Conclusion

A time offset compensation system was designed and implemented on smartphones and in MATLAB with application to beamforming. Two strategies for time offset compensation were implemented: one based on time-domain cross correlation and a maximum search, the other based on frequency-domain cross correlation and least-squares estimation.

A simulation was made to evaluate these algorithms, showing that the potential of the frequency-domain method for sub-sample accuracy was not realized and the time-domain method was more robust in highly noisy or reverberant situations. Real-life experiments show a synchronization offset of 5 to 6 samples for the time-domain algorithm.

Orientation measurements obtained from Android smartphones were shown to be unsuitable for use in this work. On a table in an office environment, the reported orientations were found to be inconsistent both between different devices and between the same device in different locations.

In addition, the sampling rate offset of Nexus 5 phones was characterized and found to be on the order of 10 mHz. Based on this result, it was decided not to implement a compensation algorithm in this work.

The Android platform was used extensively to record audio, but it was found to lack adequate provisions to query the supported audio recording settings of a device. Further, it is difficult to obtain raw audio recordings for use in signal processing, as the amount of preprocessing done on the microphone signal is device-dependent and opaque.

Bibliography

[1] B. D. Van Veen and K. M. Buckley, “Beamforming: A versatile approach to spatial filtering.” *IEEE ASSP magazine*, vol. 5, no. 2, pp. 4–24, 1988.

[2] N. Gaubitch, J. Martinez, W. Kleijn, and R. Heusdens, “On near-field beamforming with smartphone-based ad-hoc microphone arrays,” in *Acoustic Signal Enhancement (IWAENC), 2014 14th International Workshop on*, Sept 2014, pp. 94–98.

[3] M. R. P. Thomas, J. Ahrens, and I. Tashev, “Optimal 3d beamforming using measured microphone directivity patterns,” in *Acoustic Signal Enhancement; Proceedings of IWAENC 2012; International Workshop on*, Sept 2012, pp. 1–4.

[4] 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.

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

[6] M. Parviainen, P. Pertila, and M. Hamalainen, “Self-localization of wireless acoustic sensors in meeting rooms,” in *Hands-free Speech Communication and Microphone Arrays (HSCMA), 2014 4th Joint Workshop on*, May 2014, pp. 152–156.

[7] S. Markovich-Golan, S. Gannot, and I. Cohen, “Blind sampling rate offset estimation and compensation in wireless acoustic sensor networks with application to beamforming,” in *Acoustic Signal Enhancement; Proceedings of IWAENC 2012; International Workshop on*, Sept 2012, pp. 1–4.

[8] M. Hennecke and G. Fink, “Towards acoustic self-localization of ad hoc smartphone arrays,” in *Hands-free Speech Communication and Microphone Arrays (HSCMA), 2011 Joint Workshop on*, May 2011, pp. 127–132.

[9] V. Raykar, R. Lienhart, and I. Kozintsev, “Method for three-dimensional position calibration of audio sensors and actuators on a distributed computing platform,” Sep. 6 2005, US Patent 6,941,246. [Online]. Available: http://www.google.com/patents/US6941246

[10] C. H. Knapp and G. Carter, “Generalized correlation method for estimation of time delay.” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-24, pp. 320–327, 1976.

[11] S. Wehr, I. Kozintsev, R. Lienhart, and W. Kellermann, “Synchronization of acoustic sensors for distributed ad-hoc audio networks and its use for blind source separation,” in *Multimedia Software Engineering, 2004. Proceedings. IEEE Sixth International Symposium on*, Dec 2004, pp. 18–25.

[12] D. Cherkassky and S. Gannot, “Blind synchronization in wireless sensor networks with application to speech enhancement,” in *Acoustic Signal Enhancement (IWAENC), 2014 14th International Workshop on*. IEEE, 2014, pp. 183–187.

[13] J. Goslinski, M. Nowicki, and P. Skrzypczynski, “Performance comparison of EKF-based algorithms for orientation estimation on Android platform,” *Sensors Journal, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[14] N. van Wijngaarden and E. Wouters, "Acoustic Enhancement via Beamforming Using Smartphones," Bachelor's Thesis, Delft University of Technology, June 2015.

[15] R. Brinkman and T. de Rooij, "On Determining Smartphone Microphone Directivity with Application to Beamforming," Bachelor's Thesis, Delft University of Technology, June 2015.

[16] S. Lasassmeh and J. Conrad, "Time synchronization in wireless sensor networks: A survey," in *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, March 2010, pp. 242–245.

[17] M. Pawig, G. Enzner, and P. Vary, "Adaptive sampling rate correction for acoustic echo control in voice-over-ip," *Signal Processing, IEEE Transactions on*, vol. 58, no. 1, pp. 189–199, Jan 2010.

[18] A. Ballew, A. Kuzmanovic, and C. C. Lee, "Fusion of live audio recordings for blind noise reduction," in *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, July 2011, pp. 1–7.

[19] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing*. Pearson Education, 2007.

[20] W. Chu, "Impulse-response and reverberation-decay measurements made by using a periodic pseudo-random sequence," *Applied Acoustics*, vol. 29, no. 3, pp. 193 – 205, 1990.

[21] F. MacWilliams and N. Sloane, "Pseudo-random sequences and arrays," *Proceedings of the IEEE*, vol. 64, no. 12, pp. 1715–1729, Dec 1976.

[22] B. Girod, R. Rabenstein, and A. Stenger, *Signals and Systems*. Wiley, 2001.

[23] G.-B. Stan, J.-J. Embrechts, and D. Archambeau, "Comparison of different impulse response measurement techniques," *AES: Journal of the Audio Engineering Society*, vol. 50, no. 4, pp. 249–262, 2002.

[24] E. W. Hansen, *Fourier Transforms: Principles and Applications*. Wiley, 2014.

[25] International Data Corporation. Smartphone OS market share 2015, 2014, 2013, and 2012. [Online]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp

[26] C. Maia, L. Nogueira, and L. M. Pinho, "Evaluating Android OS for embedded real-time systems," *Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*.

[27] Google. Getting Started – Android Developers. [Online]. Available: https://developer.android.com/training/index.html

[28] ——. Android NDK – Android Developers. [Online]. Available: https://developer.android.com/tools/sdk/ndk/index.html

[29] F. Liu, *Android Native Development Kit Cookbook*. Packt Publishing, 2013.

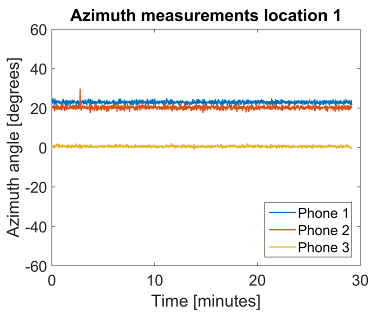
[30] A. Varshavsky, M. Chen, E. de Lara, J. Froehlich, D. Haehnel, J. Hightower, A. Lamarca, F. Potter, T. Sohn, K. Tang, and I. Smith, "Are GSM phones the solution for localization?" in *Mobile Computing Systems and Applications, 2006. WMCSA '06. Proceedings. 7th IEEE Workshop on*, Aug 2006, pp. 34–42.

[31] S. Brahler. (2010, Oct.) Analysis of the Android architecture. [Online]. Available: https://os.itec.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf

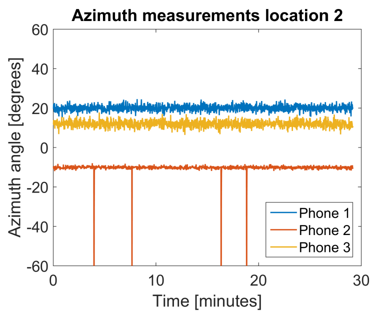
Appendix A

Full simulation and experiment results

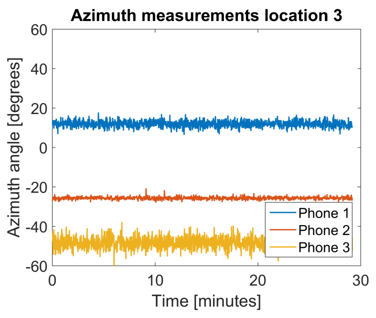
A.I Orientation measurements



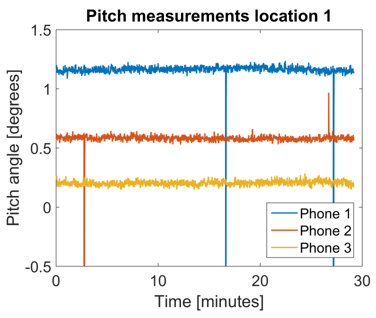
(a) Azimuth angles for location 1.



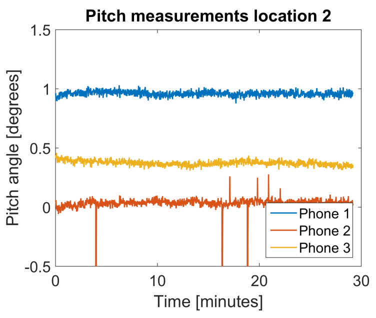
(b) Azimuth angles for location 2.



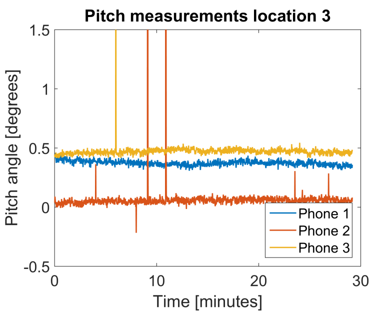
(c) Azimuth angles for location 3.



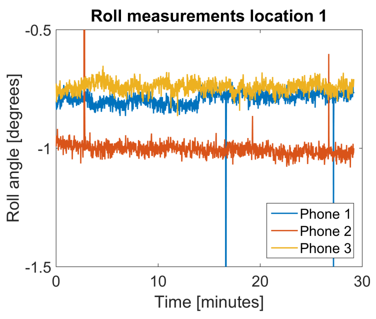
(d) Pitch angles for location 1.



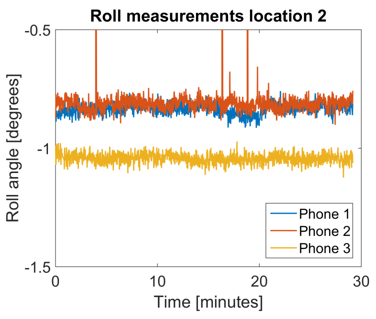
(e) Pitch angles for location 2.



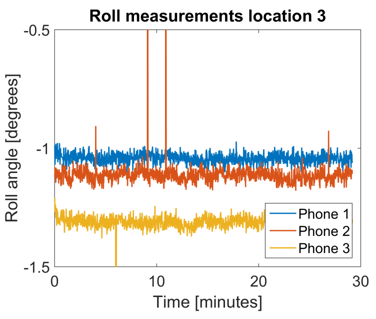
(f) Pitch angles for location 3.



(g) Roll angles for location 1.



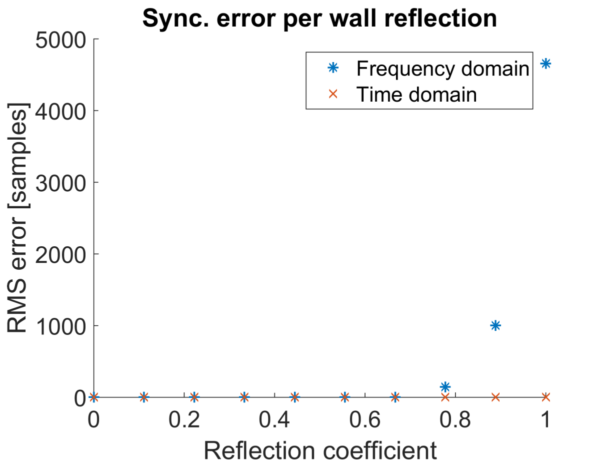
(h) Roll angles for location 2.



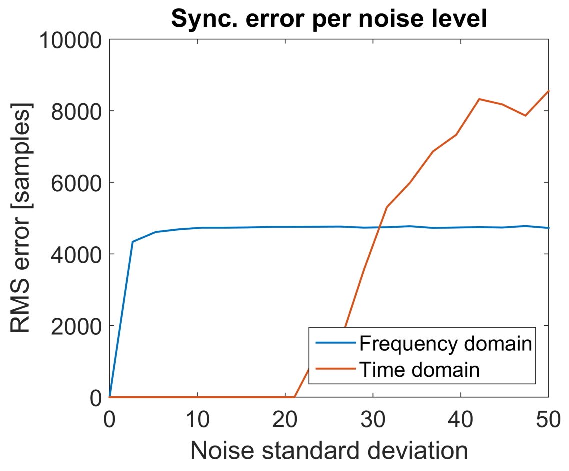
(i) Roll angles for location 3.

Fig. 12. Complete measurement results for all orientation measurements performed.

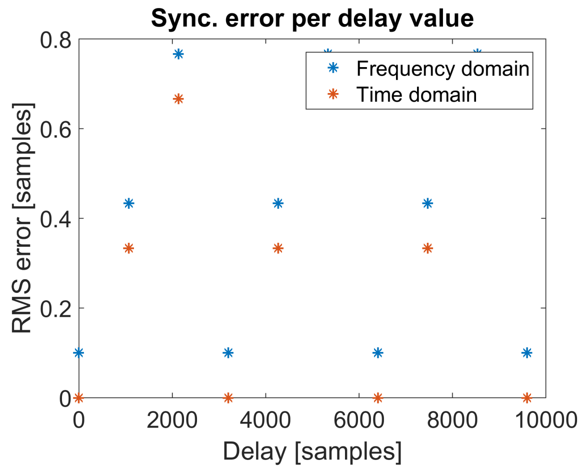
A.II Time offset simulation



(a) Delay estimation error as a function of wall reflection.



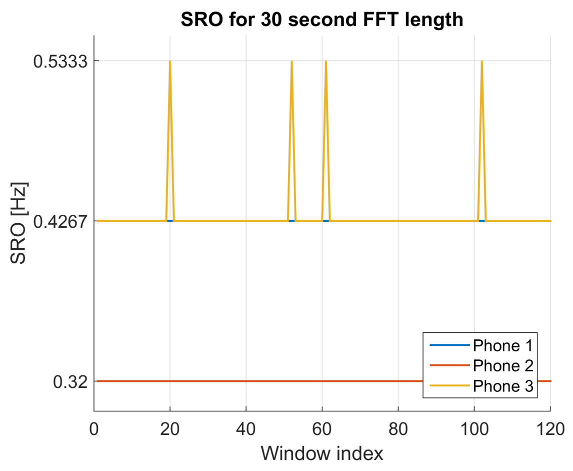
(b) Delay estimation error as a function of noise power



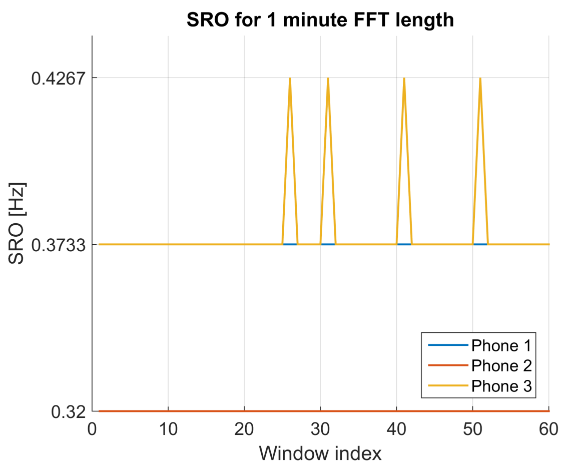
(c) Delay estimation error as a function of inserted delay

Fig. 13. Simulated results for TD and FD synchronization algorithm. The time-domain algorithm performs better under nearly all circumstances.

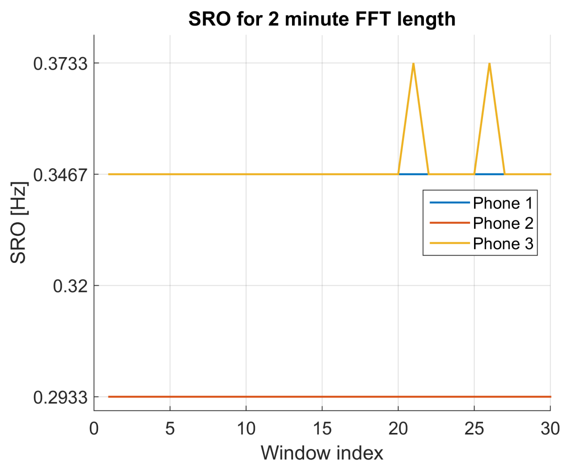
A.III Sampling rate offset measurements



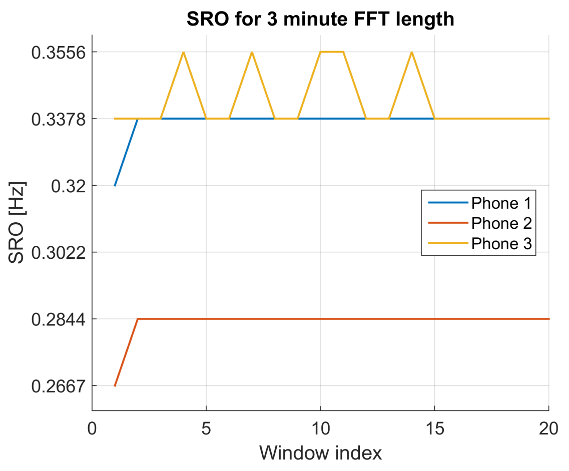
(a) 30 second FFT window, $\Delta f = 3.3$ mHz.



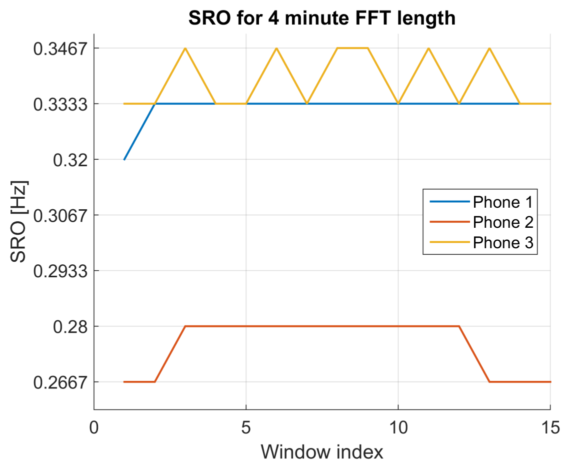
(b) One minute FFT window, $\Delta f = 6.6$ mHz



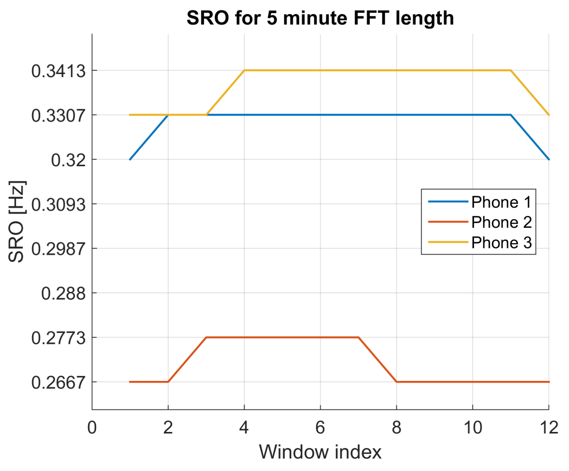
(c) 2 minute FFT window, $\Delta f = 13.2$ mHz.



(d) 3 minute FFT window, $\Delta f = 19.8$ mHz.



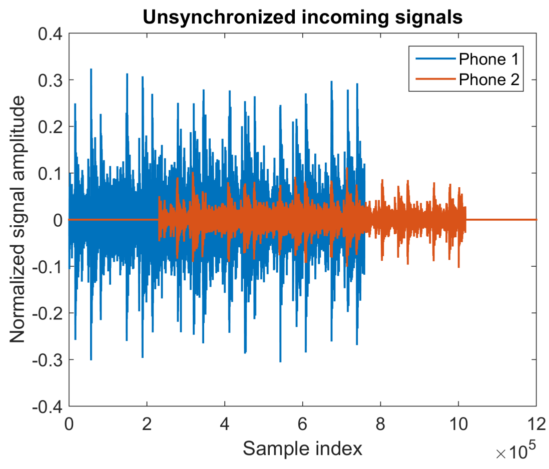
(e) 4 minute FFT window, $\Delta f = 26.4$ mHz.



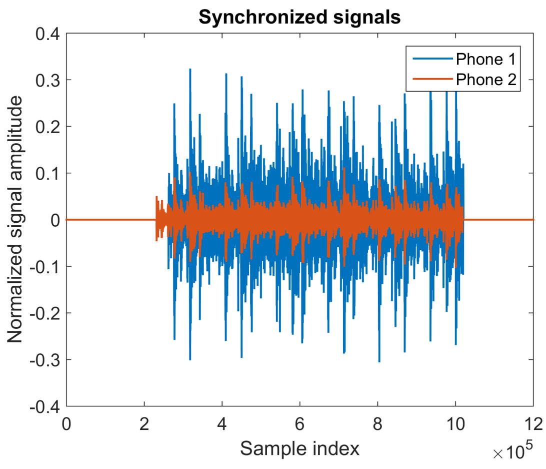
(f) 5 minute FFT window, $\Delta f = 33$ mHz.

Fig. 14. Sampling rate offset measurements for different FFT window lengths. A larger FFT window gives a higher frequency resolution Δf but lower time resolution.

A.IV Real-life synchronization results



(a) Unsynchronized smartphone audio.



(b) Synchronized smartphone audio.

Fig. 15. Before (15a) and after synchronization (15b) of received audio in a real-life scenario. This is analogous to Figs. 3a-3b. The recordings correspond to the “Music (low MLS volume)” measurement in table 11b.

Appendix B

Global coordinate system used by smartphone

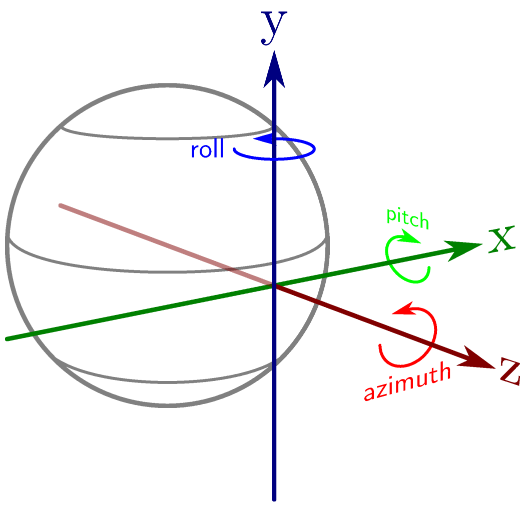


Fig. 16. The global coordinate system used for the smartphones in this work.