Noise reduction application for Android smartphones

Acquiring microphone signals, designing a user interface and hardware connectivity

Authors:
Alexander de Moes  1354728
Sina Valipoor  1237284

June 29 2012

commissioned by Dr. Ir. Richard Heusdens
Preface

This thesis is the result of nine weeks of research regarding Android applications and more specifically regarding acquiring microphone signals, designing an graphical user interface and hardware connectivity. This thesis is also the conclusion to a three year bachelor’s degree of Electrical Engineering at Delft University of technology.

This thesis is especially useful to engineer who want to use signal processing, especially audio processing, on the Android platform since it shows what can and cannot be done on an Android smartphone, regarding audio processing.

We would like to thank our supervisors Richard Heusdens and Richard Hendriks for their help with our research and for providing the Samsung Galaxy S II smartphones for testing purposes. Without these phones, testing would have taken a lot more time. Additionally we would like to thank Ioan Lager for all the work he put into organizing the final bachelor course.

This project taught us how to work on one big project intensively with a group of people. This was a great learning experience for the both of us.

Alexander de Moes and Sina Valipoor

Delft June 28, 2012
Contents

PREFACE ....................................................................................................................................................... 1
ABSTRACT ..................................................................................................................................................... 4
1 INTRODUCTION ......................................................................................................................................... 6
2 SPECIFICATIONS FOR IMPLEMENTING THE BEAM FORMING ALGORITHM IN ANDROID ..................... 7
   2.1 THE BASIC VERSION ......................................................................................................................... 7
   2.2 THE BUSINESS VERSION ................................................................................................................ 8
   2.3 THE PRODUCT SPECIFICATIONS .................................................................................................. 8
3 DEVELOPING THE SMOOTH SPEECH APP FOR ANDROID ...................................................................... 9
   3.1 THE ARCHITECTURE OF THE ANDROID PLATFORM ......................................................................... 9
   3.2 USING JAVA TO CONSTRUCT THE APPLICATION ........................................................................... 10
      3.2.1 The Button event ....................................................................................................................... 10
      3.2.2 Implementation of Beam forming algorithm in Java ................................................................. 10
   3.3 ACQUIRING MICROPHONE AUDIO SIGNALS IN ANDROID ............................................................. 11
      3.3.1 The AudioRecord Class ............................................................................................................. 11
      3.3.2 The MediaRecorder Class .......................................................................................................... 11
   3.4 LIMITATIONS OF THE ANDROID PLATFORM ................................................................................ 12
      3.4.1 Android Kernel limitations ........................................................................................................ 12
      3.4.2 Mobile Device Hardware Limitations ........................................................................................ 13
      3.4.3 Limitations of the Android Media Library .................................................................................. 13
   3.5 BUILDING THE USER INTERFACE WITH XML .............................................................................. 14
   3.6 PERMISSIONS TO USE THE PHONE’S FUNCTIONS ......................................................................... 14
   3.7 READ AND PLAY FILES IN ANDROID ............................................................................................ 15
      3.7.1 The FileReader Class and The BufferedReader Class ............................................................ 16
      3.7.2 The AudioTrack Class ............................................................................................................... 16
   3.8 CONCLUSIONS ON USING JAVA IN ANDROID ............................................................................... 17
4 DESIGNING A GRAPHICAL USER INTERFACE ....................................................................................... 18
   4.1 WHY A USER INTERFACE IS NEEDED ............................................................................................. 18
   4.2 HOW USERS DETERMINE THEIR DIRECTION ............................................................................. 18
   4.3 USING THE SEEKBAR TO SELECT THE SOURCE .......................................................................... 19
   4.4 USING THE GYROSCOPE TO SELECT THE SOURCE ...................................................................... 20
   4.5 USING A GRAPHICAL ROTATING KNOB TO DETERMINE THE DIRECTION ................................... 21
   4.6 CONSIDERATIONS FOR THE FINAL USER INTERFACE ................................................................. 21
5 THE SMOOTH SPEECH APP WITH MULTIPLE SMARTPHONES ............................................................ 23
   5.1 THE SUM AND DELAY TECHNIQUE USING ARRAY OF SMARTPHONES ......................................... 24
      5.1.1 The Sum and Delay technique ................................................................................................... 25
      5.1.2 The Sum and delay technique for smartphones ........................................................................ 28
   5.2 INTERCONNECTION OF THE ANDROID SMARTPHONES FOR REAL TIME SIGNAL PROCESSING AND SMOOTH SPEECH APP ....... 31
      5.2.1 Network opportunities for Android Device and requirements for implementation of DAS beam forming for teleconference purposes .................................................................................. 32
      5.2.2 Bluetooth network for smartphones and Android .................................................................... 34
      5.2.3 Implementation of the Smooth Speech App in Bluetooth ......................................................... 37
   5.3 ALTERNATIVE SOLUTION FOR INTERCONNECTING SMARTPHONES FOR PERFORMING DAS BEAM FORMING ALGORITHM ................................................................. 39
      5.3.1 Wired Interconnection solutions ............................................................................................... 39
Abstract

Noise reduction techniques become more important in everyday functionalities of the smartphone, but most built-in techniques offer poor results. Google and Delft University of Technology are asked to investigate noise reduction for conferences with multiple smartphone. This is the main reason why the development of the smooth speech app with the ‘click and listen’ user interface has become a bachelor thesis project. In this paper the focus is on acquiring microphone data from a smartphone, the User Interface and the ability to interconnect multiple smartphones for teleconferencing. Answers to the question; how can we make an Android application that presents real-time microphone data to a beam forming algorithm and make it user friendly at the same time, are given in this thesis.

This paper proposes several methods for building an Android application that presents real-time microphone data to a beam forming algorithm and make it user friendly at the same time. Research was divided into four main parts. The first part is the programming part in which the Android platform is studied and several methods to access the microphone data are examined. The second part of this paper tries to find the best possible user interface for The Smooth Speech App. In the third part we explored methods to connect multiple phones for data transmission purposes and position detection. The fourth part will analyze the code programmed in Java for efficiency and time consumption purposes. Because of various limitations, another solution to acquire the microphone signals had to be found.

Because of the layering model in Android, the kernel cannot be accessed directly. Moreover, the only way to access the kernel is via the core libraries provided by the Android SDK. The Android Media library offers two classes for audio data acquisition, Media Recorder Class and Audio Recorder Class. While these classes offer a range of various opportunities for audio recording coming from the internal microphone of the system, they do not offer possibilities for multiple microphones recording at the same time. The possibility of recording multiple microphones at the same time is required for The Beam Forming algorithm.

For the graphical user interface three options were chosen. The first being a slider widget, the second using the gyroscope and the third using a graphical rotating knob. All three options are easy to use, but only the graphical rotating knob gave a proper graphical representation while the source is selected. Both the gyroscope and the graphical rotating knob are difficult to develop, while the seekbar was very easy to develop. Although a combination of a gyroscope and a graphical rotating knob offers the best solution, the seekbar was the chosen option, because it was easy to develop and our time was limited.

Interconnection of multiple smartphones for audio data transmission and manipulation is necessary for the business version of the Smooth Speech App. Also the position detection of each smartphone relative to the main smartphone is a must for the Business version of the App. Several options; Bluetooth, wifi, 3G based on UMTS are explored for the real time audio data transmission. Bluetooth turned out to be the best option for implementing the network and throughout the use of NEST application target localization can be performed [23].
The Smooth Speech App is developed for the Smartphones, coded in the Java programming language. The smartphone is a small pocket computer with limited energy and storage capability. Therefore the coding should be done in an efficient way [26]. The mathematical function needed for the Beam Forming Algorithm, such as the Fast Fourier Transform (FFT) was programmed in Java using the Michael Thomas Flanagan’s Java Scientific Library. [35] To improve efficiency and time consumption, an Android NDK, which reuses C and C++ native code can be used [33].

The two microphones of the Samsung Galaxy S II cannot be accessed at the same time, that is why we had to find another solution to acquire multiple microphone signals at the same time. A PC running Matlab offers a cost-free and accessible solution, while the ioio ad-on board is costly and has to be ordered from the United States. However, the ioio board offers a real-time solution that can be used in practice. Because of limited time, the chosen solution is a PC running Matlab, but the ioio ad-on board was a better overall solution.

In conclusion, to acquire the microphone signals from a smartphone, Google must implement the beamforming algorithm on kernel level and provide a smartphone that has more than one A/D converter. This thesis offers a solution to bypass the limitations of Android and the smartphone by using a PC with Matlab, but a better solution is adding a hardware add-on board such as ioio, because it offers a real-time solution.
1 Introduction

As noise reduction techniques become more important in everyday functionalities of the smartphone, the new generations of the smartphone come with built-in techniques to reduce noise. However, these built-in techniques offer poor result for every application used in the smartphone, and they usually do not offer a user interface. An example of such an application is Siri, which is a voice-controlled system build into iPhones. According to [2] it offers poor solutions for noise reduction and the support has been an issue for some owners of earlier iPhones. This is why Google is looking for methods to improve noise reduction techniques. Furthermore, since most people carry smartphones, Google and Delft University of Technology are investigating noise reduction for conferences with multiple smartphones. This is the main reason why the development of the smooth speech app with the ‘click and listen’ user interface has become a bachelor thesis project. The Smooth Speech App consists of two parts, the Beam forming Algorithm and the User Interface.

In this paper the focus is on acquiring microphone data from a smartphone, the User Interface and the ability to interconnect multiple smartphones for teleconferencing. Answers to the question; how can we make an Android application that presents real-time microphone data to a beam forming algorithm and make it user friendly at the same time, are given in this thesis. The smooth speech app with the click and listen interface is programmed as an Android SDK package in Eclipse and is intended for Android smartphones. The Application can serve as a noise reduction application for Android smartphones or for teleconferences with multiple connected smartphones.

In this thesis, the smooth speech app is introduced and the click and listen interface is further explained. In chapter two an introduction is made for the project implementation of Beamforming in Android and the product specifications for this project. In chapter three we will look at the architecture and capabilities of the Android platform. Additionally, in chapter four, the graphical user interface is developed. In the fifth chapter interconnection between multiple smartphones is explained, needed for conferencing purposes. In chapter six programming improvements are described and chapter seven describes alternative solutions to acquire microphone signals. Finally in chapter eight the conclusions are drawn and recommendations are given.
2 Specifications for implementing the Beam forming algorithm in Android

The implementation of the beam forming algorithm in Android is researched and developed by five Electrical Engineering Bachelor students at the Delft University of Technology (TU Delft). The research for implementing the beam forming algorithm in Android is commissioned by Google and a supervisor from TU Delft (R. Heusdens). The implementation of the beam forming algorithm is divided into two parts: the algorithm part and the application part. The algorithm part deals with finding the best algorithm available for performing the beam forming on mobile devices. [7] The application part deals with the implementation of the beam forming in Android mobile devices. This implementation is performed and examined on Samsung Galaxy S2 running on Android version 2.3.3. For performing the beam forming algorithm the mobile devices hardware and hardware drivers are pushed to the limited. The research is meant to find these limits for mobile devices and mobile operating system Android.

The beam forming algorithm makes it possible to select the desired sound direction, filtering the unwanted sound coming from other directions. This feature offers opportunities for using the beam forming algorithm in mobile devices. This paper focuses on research for implementing the beam forming for noise reduction in voice calls and record applications, and for using the beam forming for teleconference purposes. The Smooth Speech Application is developed to test the beam forming algorithm for these two purposes in Samsung Galaxy S2 device. There are two versions developed: the Basic and the Business version which be explained in section 2.1 and 2.2 below. The Smooth Speech application is developed and designed for Samsung Galaxy S2 which is a mobile device with limited energy and memory capacity. After developing an application for mobile devices an improvement of the writing software is performed. The Android-SDK uses Java programming language and java is based on object oriented programming. The analysis and improvement of object oriented programming is done in chapter seven.

2.1 The Basic Version

The Basic version is meant for testing the beamforming algorithm with internal MIC’s of the Samsung Galaxy S2. For the possibility to perform the beamforming algorithm, more than one audio source is needed. [7] This means that the smartphone should not be limited to only one internal MIC and the mobile operating system (being Android in this case) should offer opportunities to record with multiple internal MIC’s at the same time. This concept is researched in chapter three. Using the beamforming algorithm needs interaction with the smartphone user, to enter the desired direction of the sound for example. Three opportunities to build a user interface for the Smooth Speech application are described in chapter four. In chapter eight the conclusion and recommendations are giving based on knowledge gained in the research done in chapter three and four.
2.2 The Business Version

The Business version is built to investigate the use of beam forming for the teleconference purposes. The desire of Google is to build an application in Android which allows the users to perform teleconferences, independent of the location and by using only the smartphones. In the teleconference mode each smartphone acts like a single MIC. For the teleconference purposes, the smartphones need to be linked through an ad-hoc network, which enables real time signal transmission and offers position detection. In chapter five a research is done in setting up a network for mobile devices based on opportunities in an Android device. In chapter eight the recommendations are giving for implementing the network.

2.3 The Product specifications

A list of product specification is made up to come up with requirements needed to be achieved by the end of this project. This project is a research and develop project which is divided between two groups. An extensive list with the terms of requirements with regards to final product can be found in Appendix E. To meet with these requirements, we have come up with the following research topics:

- Record with multiple MIC’s at the same time. (For having data to test the beam forming algorithm)
- Investigate the opportunities to record multiple internal microphones in Samsung Galaxy S2.

- Investigate the hardware limitations of an Android smartphone. Samsung Galaxy S2

- Investigate the Graphical User Interface (GUI) Design in Android and design best suited interface for the beam forming Application.

- Investigate the opportunities for setting up an Ad-Hoc network between the smartphones.

- Investigate the improvement of Android application built. The improvement in programming, memory and energy usage in the device.
3 Developing the smooth speech app for Android

The Android runtime environment offers a core set of operating system libraries that can be accessed via Java and XML. While the Java programming language is used to process the activities and algorithms, the XML markup language is used to program the user interface of our application. In this chapter the architecture of the Android platform and the basic implementation of the Smooth Speech App will be explained. Section 3.1 will explain the architecture of the Android platform. Section 3.2 will give more details about how Java is used in our application. How to acquire microphone signals will be explained in section 3.3 and 3.4 will discuss the limitations of the Android platform. In section 3.5, the XML code will be explained, while section 3.6 describes the addition of permissions for use of the mobile devices hardware. Reading files from the external or internal storage of the Android device is discussed in section 3.7.

3.1 The architecture of the Android platform

The Android platform consists of five different layers [1]: applications, application framework, libraries, Android runtime environment and the Linux kernel, as seen in figure 3.1. The lowest layer, the Linux 2.6 kernel acts as an abstraction layer between the hardware and the rest of the software stack. The Android runtime includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language, including the Dalvik virtual machine[1]. The Dalvik virtual machine is needed to run the java code of the applications. Each of the layers can only communicate with the layer directly under or above the other, so when our application wants to access the microphone, it needs the proper libraries to communicate with the Linux kernel. Additionally, the application layer cannot access hardware information, such as internal clock signals. This is actually beneficial for programmers, since they do not have to worry about the hardware. Every application works the same on every piece of hardware running Android.

![Android Architecture Diagram](image)

*Figure 3.1 Android Architecture Diagram[1]*
In order to program the application, the Android software development kit or SDK that integrates into the Eclipse integrated development environment was used. The application will run on a Samsung Galaxy S II. The main reason this phone was chosen, is the fact that this phone has two built-in microphones, that are needed for the beamforming algorithm. The full specifications of the phone can be found in Appendix A.

3.2 Using Java to construct the application

The Android development kit uses an Eclipse [3] plug which enables it to program the functionalities of our application in Java. Still some of the java functions in Android are not present in basic Java syntax. Most of those functions are there to communicate with the GUI. Those functions will be explained in this subchapter.

3.2.1 The Button event

The button1 variable in the java code must be linked to the button1 in the XML code. Code example 1 shows how the variable button of the type button is linked to button1 in the R.java file. The R.java keeps track of all the variables, strings and GUI elements in the XML file.

```java
button = (Button) findViewById(R.id.button1);
code example 2.1
```

In order for the program to react on click input from the user, the onClick activity must be implemented. The onClick activity is activated when the touch screen of the phone is clicked. If the location of the finger matches that of the button, the string of TextView1 is changed. Code example 2 is an example of an onClick activity.

```java
public void onClick(View v) {
    if(v.getId() == R.id.button1) {
        TextView1.setText("microphone 1 activated");
    }
}
code example 2.2
```

3.2.2 Implementation of Beam forming algorithm in Java

The beam forming algorithm is implemented in java and performs mathematical operations on the samples coming from the Audio record class.
3.3 Acquiring microphone Audio signals in Android

The Android multimedia framework includes support for capturing and encoding the common audio formats. For recording the microphone signals from the Android device the Android Media library is used. The Android media library package consists of 16 interfaces and 27 classes. [1] For recording audio the AudioRecord or the MediaRecorder class can be used. There are Differences between the AudioRecord class and MediaRecorder class. The AudioRecord class delivers an output which is sampled version of the audio that is recorded. These samples can directly be used to perform mathematical operations on it like the beam forming algorithm. The MediaRecorder class can be used to record audio and video. The output files are in .3gp format which can be played directly by the MediaPlayer class or the internal media player of the Android device. [1] The MediaPlayer class can be used to control playback of audio/video files and streams. Section 3.3.1 will discuss the AudioRecord class and section 3.3.2 will discuss the MediaRecorder class.

3.3.1 The AudioRecord Class

The AudioRecord Class delivers a sampled version of the Audio that is recorded and manages the audio resources for Java applications to record audio from the audio input hardware of the platform. After the recording is finished, the samples have to be pulled back in the data Array from the AudioRecord object. [1] The advantage that the AudioRecord Class offers is that while running the App the Samples are saved in temporary variable and can be accessed and manipulated thought the App running time. This mean that unlike the MediaRecorder Class there is no need to save the samples to a location on the external memory of the smartphone before manipulating them. This can save Memory space. Audio Class is a Class in Java and for being able to use it, a constructor must be developed containing the desired specifications. The Desired parameters are:

- Audio Source
- Sample rate (in HZ)
- Channel configuration
- Audio format
- Buffer size

In Appendix C.1 more information can be found according to constructing an AudioRecord Object.

3.3.2 The MediaRecorder Class

The MediaRecorder Class captures Audio and Video, and enables saving them to the smartphones internal or external memory. The format in which the audio and video is captured is .3gp and by use of the MediaPlayer Class these file can directly be played on the smartphone. [1] However the MediaRecorder Class is not suited for signal processing. Because of the format in which it delivers the output file an extra signal processing is needed to bring the signal to the desired level for processing. Another disadvantage that comes with MediaRecorder Class is that it requires saving space for the captured Audio data in .3gp format. [1] While capturing Audio data coming from the internal microphone of the Android device, this data is directly saved to a file on the internal or external storage location of the smartphone. The following parameter is needed for the constructor to use the MediaRecorder Class. [1]
3.4 Limitations of the Android platform

During the research numerous limitations have been encountered and in this paragraph all these limitations will be discussed. These limitations disable the direct implementation of the Smooth Speech App on the Samsung Galaxy S2 device. The Section 3.4.1 will discuss the limitations of the Linux kernel and in 3.4.2 the limitation due to smartphones hardware are discussed and section 3.4.3 will discuss the limitations of the Android media library.

3.4.1 Android Kernel limitations

The layering model in Android is developed so that app developers do not have to worry about the underlying hardware. The core libraries give the Android developer access to the hardware, via the kernel, but the developer cannot access hardware that has no core library. That is why app developers are limited by the functionality of the core libraries. An application cannot read microphone data directly, that is why the MediaRecorder or AudioRecord classes of the Android.Media library was needed to access microphone data. Additionally, it is very inefficient when data has to travel through all the different layers and back, when everything could be processed in one layer. See figure 2.3.

![Figure 3.3 smooth speech app versus kernel integration](image-url)
3.4.2 Mobile Device Hardware Limitations

The kernel on consumer phones is locked, so that sensitive data, like the phones IMEI (International Mobile Equipment Identity) number or carrier information cannot be changed unless there is permission of the smartphone manufacturers. So if the beamforming algorithm was to be added to the kernel, Google would have to integrate it themselves. Another option is hacking the software of the phone. This is better known as rooting. Since this option is not legal, it was not taken into consideration.

Another problem is that smartphone manufacturers are not forced to add every function of the core libraries into their smartphone. The Android libraries allow applications to select different microphone sources as discussed in the previous section, such as main microphone and camera microphone. However the smartphone manufacturers design their smartphones on the different concepts, adding or subtracting extra hardware. This means that each smartphone has a different hardware layering and hardware components. Although most smartphones contain most common hardware, the choice for adding an extra A/D converter is up to the manufactures.

The First device on which the Smooth Speech App is installed, is the Samsung Galaxy S2. The Samsung Galaxy S2 seems to be the fastest smartphone on the market [4], so the focus of Samsung lies in speeding up the device rather than adding more hardware for Digital Signal Processing (DSP) like an extra Analog to Digital Converters (ADC). This makes the Samsung Galaxy S2 an unsuited device for implementation of the Smooth Speech App.

3.4.3 Limitations of the Android Media Library

The Android Media Library manages various interfaces in audio and video. However for the Smooth Speech App and the Beam Forming Algorithm, at least two or more microphones have to record audio at the same time. Every time a recording event happens by using MediaRecorder or the AudioRecord class, the audio equipment hardware that is set for the audio source in the constructor will be activated and is able to capture audio while the other audio sources are blocked and will not record till the first audio source is finished recording. This means that the Android platform architect is settled in such a way that device is able to use or manage one hardware at a time if the mobile device hardware (like the Samsung Galaxy S2) contains only one ADC. While the smartphone (Samsung Galaxy S2) offers multiple hardware for audio data capture, it only contains one ADC. This is why the Android software allows the use of only one at the time. This is one of the Limitations in using Android libraries to capture audio for the Samsung Galaxy S2 smartphone.

Furthermore; The Android Operating System (AOS) is shockingly inefficient in dealing with real-time audio signals. The Android OS adds about seven second delays to the signal, and the hardware adds another six seconds delay [5]. These gives a default overall delay of 13 seconds meaning that each audio signal recorded will experience 13 seconds delay before it can be processed or play. As this is not always the case for example when an App is built only to record and play, the delay will be about 500ms. However when signal processing is involved, as it is the case with beam forming algorithm the delay becomes that large. As the signal processing also brings delays around 1
to 2 seconds, the final overall delay will only grow to be more and the concept of real-time signal processing will fail for the Smooth Speech App.

A wide research on the web shows that Android lacks the good support for building audio apps. Various Android App developers complain about the missing functionalities in the standard libraries used by Android and the delays added by the Android OS.[2] Also the lack of support for simultaneously recording of multiple audio sources at the same time limits the developers in the Apps they can build for signal processing. As long as Google does not come up with a solution the Android OS is limited for building audio processing Apps and fails for the development of the Smooth Speech app. In conclusion, neither MediaRecorder nor AudioRecord is suitable for the smooth speech app.

3.5 Building the user interface with XML

The extensible markup language, better known as XML, is a standard that is used to present structured data in plain text. Therefore it is a very useful tool to present a graphical user interface and that is why the Android graphical user interface is presented in XML.

The main.xml file contains the GUI data of the Android application. When an object like a button for example, is added, it needs an ID, a size and a string. Code example 3 shows how button1 is declared. The main.xml file can be found in in appendix F.2. The smooth speech app uses multiple objects including Button, ImageView, TextView and Seekbar. The graphical part of the user interface will be discussed in chapter four.

```xml
<Button
    Android:id="@+id/button"
    Android:layout_width="wrap_content"
    Android:layout_height="wrap_content"
    Android:text="@string/mic1" />
```

Code example 2.3

3.6 Permissions to use the phone's functions

The Android manifest file presents all the necessary information about the application to the Android system. The system must have all the information of the manifest file, before it can run a single piece of code of the application. [1] Some of the functions of the manifest are:

- Name the Java package for the application
- Declares the permissions of the application
- Generates a list of library used by the application
- states the minimum required API level
The Android device needs permissions to use the sensors on the phone. As stated above, these permissions are declared in the manifest file. Since multiple sensors of the phone were used, including the microphone, SD card and gyroscope, it is very important that the right permissions from the users to access these sensors are given to the application. That is why these permissions were added in the manifest file. Code example 4 shows what these declarations look like. The manifest file of the smooth speech app can be found in appendix F.3.

<uses-permission Android:name="Android.permission.RECORD_AUDIO"></uses-permission>

Code example 2.4

### 3.7 Read and play files in Android

Reading files in Android smartphone is performed because the samples are read out from a .txt file which is saved on the External-Storage of the smartphone. The samples are saved as a String and the .txt sample files are read out line for line. Figure 2.4 shows an example of the samples saved in test.txt file on the Samsung galaxy S2 smartphone.

![Figure 3.4 Example of sample.txt file on Samsung Galaxy S2](image)

After reading out the samples one by one they are converted to a double type. This is necessary because than the sample value can be manipulated mathematically by java. [6] After the conversion, the samples are read in an array. The sample arrays are manipulated by the Beam Forming algorithm and afterwards played by the smartphone. The Beam Forming Class reads in the samples and performs the Beam Forming algorithm on them. [7] After the performance of the Beam Forming algorithm, the Beam Forming algorithm class produces one array of samples. These sample values are converted to a type Short so that they can be played directly by the AudioTrack class in Android.Media package.
3.7.1 The FileReader Class and The BufferedReader Class

The Java.IO package manages the Data input and output from the files in Android. [1] The Java.IO contains the FileReader class which is used in Smooth Speech App for the File Read out of the samples. The FileReader class is a specialized reader that reads from a file in file system. [1]

After the construction of a File Reader the BufferedReader class is used to buffer the input data. The BufferedReader wraps an existing Reader and buffers the input. [1] The BufferedReader contains the readLine() method.[1] The readLine() method reads each line of the file one for one. The data on each line is than converted to a String type. Code example 5 below shows an example of using the BufferedReader Class and the readLine() method.

```java
FileReader fileReaderText1 = new FileReader(path1);
BufferedReader br1 = new BufferedReader(fileReaderText1);
input1 = br1.readLine();
```

By the next BufferedReader.readLine() command the next line in the file is buffered and read out. Java contains standard libraries for converting the String type to a Double type. These libraries can be imported and used in Android SDK.[1,6] The library used in the Smooth Speech App for converting String type to a Double is the Java.Lang.Double. The Double class converts the String to a Double type.

3.7.2 The AudioTrack Class

The AudioTrack Class manages and plays a single audio source for Java Application. [1] The Android.Media package contains the AudioTrack class. The AudioTrack instance can operate under two modes: static or streaming. Static mode is used when dealing with short sounds that fit in memory and need to be played fast. Streaming mode offers buffer options where the application writes a continuous stream of data. The Samples which are played are first pushed to the AudioTrack object using the method Write() and played afterwards. Code example 6 below shows an example of using the AudioTrack Class, Write() and play() method.

```java
public int bufferSize = AudioRecord.getMinBufferSize(8000, AudioFormat.CHANNEL_CONFIGURATION_MONO, AudioFormat.ENCODING_PCM_16BIT) * 256;
AudioTrack play = new AudioTrack(AudioManager.STREAM_VOICE_CALL, 8000, AudioFormat.CHANNEL_CONFIGURATION_MONO, AudioFormat.ENCODING_PCM_16BIT, bufferSize, AudioTrack.MODE_STATIC);
play.write(inputsample1, 0, inputsample1.length / 1);
play.play();
```

For constructing the AudioTrack Object, the same parameters are needed as for constructing an AudioRecord Object explained in section 2.3.1. More information about these parameters can be found in Appendix C.1.
3.8 Conclusions on using Java in Android

The layering model in Android blocks direct access to the kernel, but the core libraries provided by the Android SDK enables access to the kernel, though accessibility is limited to few basic functions. For example, the Android Media library offers two classes for audio data acquisition, the Media Recorder Class and the Audio Recorder Class. While these classes offer a range of various possibilities for audio recording coming from the internal microphone of the system, they do not offer possibilities for multiple microphone recording at the same time. The Android SDK offers opportunities to design a sophisticated and a professional layout in the Main.xml file [16] section 2.5 and 3. Before an Android App can be used, the Manifest file should be set up with the necessarily permissions and java package names [1]as seen in section 2.6.

Furthermore, as seen in section 2.4, the microphones cannot be accessed due to limitations of the android platform and Samsung phone; two microphones cannot be accessed at the same time. By using other options offered by the standard Android SDK libraries, we can work our way around the limitations of the Android libraries. The Read and Play file libraries in Android [1] section 2.7, enables the performance of the Beam Forming algorithm on the Android smartphone without using the microphones of the Samsung phone. This method could be used as an alternative technique to present samples to the beamforming algorithm. Additionally, the best way to implement the smooth speech app, including the beamforming algorithm, is by integrating it into the Android platform at kernel level.

In conclusion to this chapter, the two microphones of the Samsung phones cannot be accessed at the same time, because the phone only has one A/D converter and that is why another solution must be found to acquire microphone signals. The best option to implement the beamforming algorithm is by integrating it into the Android platform instead of an application.
4 Designing a graphical user interface

The ‘click and listen’ graphical user interface is a very important part of our application, because it is the bridge between the user and our technology. In this chapter the different choices made while developing the user interface of the smooth speech app will be explained. Section 4.1 will explain why a user interface is needed and section 4.2 will describe the way users determine their direction. Additionally, section 4.3, 4.4 and 4.5 will discuss three different options for the user to determine their direction. Finally in section 4.6

4.1 Why a user interface is needed

Noise reduction in today’s smartphones is completely invisible for the user. The reason for this, is that traditional noise reduction techniques use passive algorithms that require no input from the user. The beamforming algorithm however needs to know from which direction the input signal is coming, so it can cancel out noise from all other directions. This function is built into the user interface. To determine the direction of the source the following options have been chosen, that will be explained sections 4.3, 4.4 and 4.5:

- a seekbar widget
- the internal gyroscope
- a graphical rotating knob

4.2 How users determine their direction

Beamforming it is not a part of our research, but still basic knowledge was needed to know how it works in order to design the proper user interface. In short beamforming is a signal processing technique that uses an array of sensors to determine the direction of the source, so with beamforming the direction of each source can be determined. When the direction our desired source is known, all other sources can be filtered out. The difficulty lies in finding the direction of the source. The red circle in figure 4.1 shows the location of the source, who is the user in this case. Audio coming from this direction will be enhanced and audio coming from all other directions is filtered out. However, when the position of either the phone or the source is changed, the direction of the source in the application has to be changed aswell.
The graphical user interface will have to show the user which direction is chosen as its source at any time. Therefore our user interface must include a way to show this to the user. Figure 3.2 is a simplified version of figure 4.1 and will be used to show the user which direction is selected as the source. The green line shows the direction of the source.

4.3 Using the seekbar to select the source

The slider widget, better known as the seekbar, is a basic Android widget already available in the Android SDK. That is why it is very easy to implement and the slider itself requires no coding. The user simply slides the bar to the desired angle and the selected angle is used as the source. When the slider is swiped to the left side, zero degrees is the direction of the source and when the slider is swiped all the way to the right, 359 degrees is the direction of the source. Because the slider widget
is a basic Android widget, most users are familiar with it and therefore it is easy to use. However, a slider does not give a good graphical representation of the source.

![Slider at 180 degrees](image)

**Figure 3.3 the slider or seekbar at 180 degrees**

### 4.4 Using the gyroscope to select the source

The Samsung galaxy S2 possesses an internal gyroscope that could be used to determine the direction of the source. The MicroElectroMechanical, or MEMs gyroscope inside the Samsung Galaxy S II is an AGD82103 sensor. This sensor operates by making use of a *proof mass* plate, which oscillates when a drive signal is applied to capacitor plates [8:71]. When the phone is rotated, the proof mass is moved in the X, Y and Z direction and the change in capacitance is measured by a processor. This change in capacitance variation is used to detect angular acceleration in the X, Y and Z direction, shown in figure 4.4. This information can then be used to determine towards which direction the phone moves.

![Gyroscope on the Samsung Galaxy S II](image)

**Figure 4.5 gyroscope on the Samsung Galaxy S II can see the direction of the phone’s movement**
The information of gyroscope can be used to determine the direction of the source by the user and when the orientation of the phone changes, the phone still knows the direction of the source. However, when the position of the phone is changed along the X, Y and Z axes, the angle changes and complex algorithms must be constructed to keep track if the location of the source. This is beyond the scope of our research and is therefore left open for further study.

4.5 Using a graphical rotating knob to determine the direction

Another option to determine the direction of the source is with a graphical rotating knob. This knob is basically the same as figure 3.2, but now it does not only show the direction of the source to the user, but the user can also move the direction of the source with a finger. This is shown in figure 3.5. Figure 3.6 shows to rotating knob that was changed to approximately 230 degrees.

Programming this knob requires a lot of work since finger inputs need to be registered and the knob has to be changed at the same time. This is however a very convenient solution for the user as it gives him a graphical representation while selecting the source.

4.6 Considerations for the final user interface

In the previous sections 3 possible solutions to determine the direction of the source have been described and this section has taken all their advantages and disadvantages into consideration. An overview of all considered points can be found in table 3.1. First of all no option is usable when the position of either the user or the phone is changed, but this is a limitation of the beamforming algorithm rather than a limitation of our three options. All three options are easy to use, but only the graphical rotating knob gives a proper graphical representation while the source is selected. Both the gyroscope and the graphical rotating knob are difficult to develop, while the seekbar is very easy to develop since the seekbar itself does not require any coding. Only the gyroscope still works when the orientation is changed. Although a combination of a gyroscope and a graphical rotating knob offers the best solution, the seekbar was the chosen option, because it was easy to develop and our time is limited.
Table 4.1 points considered while choosing the best way to determine the direction of the source

<table>
<thead>
<tr>
<th></th>
<th>seekbar</th>
<th>gyroscope</th>
<th>Graphical rotating knob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to use</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Good graphical representation</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Easy to develop</td>
<td>✓</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Usable when the orientation is changed</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Usable when the position is changed</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

The final graphical user interface is a combination of figure 3.2 that shows the user which direction is chosen and a seekbar to select the location of the audio source. This final user interface is shown in figure 4.7 and is called the click and listen interface.

![Figure 4.7 the complete GUI or click and listen interface](image)
5 The Smooth Speech App with multiple smartphones

In this chapter investigation are made for using the Smooth Speech App with multiple smartphones. The purpose of using the Smooth Speech App with multiple smartphones comes from the desire to use the Beam Forming techniques for teleconference purposes. This would allow the user’s to have a teleconference using their smartphones, where the participants (at the same location) can still consult without disturbing the teleconference. The Beam Forming technique makes spatial filtering of the unwanted sound possible by creating a focused ‘beam-like’ sensitivity pattern for the microphone array. Using this ability for Teleconference purposes, only the wanted voice can be sent through; while filtering the unwanted sound and voices coming from other directions. [35] An array of microphones (at least two) is needed to be able to perform beam forming. Also there are different techniques available for performing the Beam Forming. [7]

The investigation till now where made only looking at one smartphone, where we have used the internal microphones of the smartphone to perform Beam Forming. Now we want an interconnection of multiple smartphones where each smartphone acts like a separate microphone. With these smartphones an array is formed where each smartphone has fixed position relative to each other and the table. The Sum and Delay [7] [35] algorithm is used to perform Beam Forming. In section 5.1 the concept of Sum and Delay technique using an Array of smartphones is described.

The smartphones need to be interconnected for Real Time Digital Signal Processing (DSP) purposes, so being able to perform the Beam Forming algorithm and build the array of smartphones. Section 5.2 will describe and research the network opportunities for Android smartphone to interconnect. The interconnection of multiple smartphones for real time signal data processing is a new concept. In section 5.2 we investigate the opportunities to build a network for real time signal processing with Android smartphones.

As an alternative investigations are made for designing an extra hardware called MIC array. The Beam Forming concept using the Sum and Delay technique can be performed using this array and the internal MCU. The array can be connected to android smartphone wireless for data transmission.
5.1 The Sum and Delay technique using array of smartphones

In this section the simplest Beam Forming architecture, the delay and Sum is described for array of smartphones. The Beam Forming technique allows the microphones to have a focused maximal sensitivity in one direction only; this is graphically shown below in figure 5.2.

Figure 5.2: Response of omnidirectional and focused beam pattern [35].

The Sum and Delay Beam forming technique allows virtually and electronically steering the beam pattern without physically moving the array. This is achieved by adding a delay to each of the array elements and summing the signals afterwards. In section 5.1.1 the concept of Sum and Delay beam forming is shortly explained for an array of microphones. In section 5.1.2 the same concept is introduced for an array of smartphones.
5.1.1 The Sum and Delay technique

The Sum and Delay beam forming technique uses an array of microphones where each microphone experience difference in time of arrival of the wave originated at the source. This difference in time of arrival is due to direction from which a wave is originated. After summing up the incoming signals from all the microphones, the maximum output is achieved when the signal originated from a source perpendicular to the microphone array. [35]

This is graphically shown in figure 5.3 below.

By adding an extra delay stage to each microphone in the array, the beam can be controlled. This allows the users to steer the beam to focus on the desired direction. This is graphically shown in figure 5.4 below.
The data from each microphone is sampled and the delays are introduced by buffering these samples. Figure 5.5 below illustrate this concept.

A plane wave approaching from an angle of 45 degrees on the microphone array, will reach microphone A before it reaches microphone B. We can steer the Beam to the wave direction by delaying the signal in microphone A.

- $\theta_1$ = Green angle
- $\theta_2$ = Red angle
- $d$ = distance that the wave needs to travel to reach Microphone B after reaching Microphone A.
- $\tau$ = delay
- $v$ = speed of sound

First the delay is calculated:

\[
d = 0.25 \times \cos(\theta_1) = 0.25 \times \sin(\theta_2) = 0.1768 \text{ meter} \quad (5.1)
\]

\[
\frac{d}{v} = \frac{0.1768}{343.3} = 0.5149 \times 10^{-3} \approx 0.5149 \text{ ms} \quad (5.2)
\]
If the incoming signal is sampled by $f_s = 48000 \text{ Hz}$,

$$t_s = \frac{1}{48000} = 20.83\mu s \quad (5.3)$$

$$delay_S = \frac{0.5149\text{ms}}{20.83\mu s} = 24.717 \text{ samples} \quad (5.4)$$

If the signal in microphone A is delayed by 24.717 samples and summed with signal coming from microphone B, a virtual beam is created that point directly to the incoming wave direction.

Once the direction of the desired wave is known a virtual beam can be constructed. According to formula 5.5 below for each microphone the amount of delays needed in samples can be calculated.

$$delay \text{ in samples} = \frac{f_s l \cos(\theta_s)}{v_s} \quad (5.5)$$

$f_s$ = sampling frequency

$\theta_s$ = angle of the incoming wave

$l$ = distance between the microphones

The integer part of the samples to be delayed is achieved by using a buffer that holds the samples. [35]. The fractionally part of the samples to be delayed is achieved by reconstructing the original waveform and resampling it. [35] An in-depth algorithm derivation and analyze of the Sum and Delay technique and analyses of two other beam forming techniques can be found in thesis of the algorithm group. [7]

For simplicity the following assumption is made, the target source is positioned in the far- and free-field. So we can neglect the difference in damping between the microphones and we assume the incoming waves to be planar.
5.1.2 The Sum and delay technique for smartphones

In this section we introduce the implementation of the beam forming by using the Sum and Delay technique with multiple smartphones. For simplicity only two smartphones are used to replace the two microphones in section 5.1.1. Figure 5.6 below shows the situation graphically.

![Graphical representation of the beam forming with two smartphones.](image)

Figure 5.6 Graphical representation of the beam forming with two smartphones.

The internal microphone of the smartphone is used for acquiring the signals and the internal Analog to Digital converter is used to sample the signals. From the smartphones we get the sampled version of the signals. An example of this signals are shown in figure 5.9 below. This signal assumed to have no noise, idealistically only the signal from the desired direction is displayed.
Figure 5.7 Sampled signal coming from the smartphones.

The signal from smartphone 2 is the same as the signal from smartphone 1 only shifted in time. The plane wave front approaching from an angle \(\theta_1^*\) of -45 degrees on the smartphone array, will reach smartphone 1 before it reaches smartphone 2. By using formula 5.5 we can calculate the delay in samples needed. The angle of the incoming wave \(\theta_1\) in formula 5.5 is related to the wave front angle \(\theta_1^*\) as follow:

\[
\theta_1 = 90 - |\theta_1^*| \tag{5.6}
\]

The \(l\) and \(f_s\) are the distance between the smartphones in [m] and the desired samplings frequency in [Hz], respectively. In this case we consider one dimensional smartphone array, where the positions of the smartphones difference in one direction (y) only, this is shown in figure 5.6 above.

Furthermore the distance between the smartphones \(l\) has an effect on beam pattern and creation of extra unwanted lobes, this effect is called grating lobes [35]. Grating lobes are unwanted lobes and cause the array to pick up signals from other directions other than the desired direction. Grating occurs when the extra distance a signal wave front must travel between the smartphones is a multiple of the signal’s wavelength. Furthermore if the distance between the smartphones is greater than the half of the incoming wave; the spatial aliasing [7] will occur.

\[
l < \frac{\lambda}{2} = \frac{c}{2f_s} \tag{5.7}
\]

Formula 5.7 sets boundary for the distance between the smartphones \(l\) if we know the desired frequency of the signal for which we want to use beam forming.
The samplings frequency \( (f_s) \), plays an important role in reconstructing the original signal from the samples. According to Nyquist-Shannon sampling theorem, the samplings frequency should at least be twice as the highest frequency available in the signal or the bandwidth.

\[
f_s \geq 2f_{\text{MAX}}
\]

(5.8)

This is needed for reconstructing the original signal and in order to be able to delay the fractionally part of the samples as explained in section 5.1.1. The Smooth Speech App samples the audio signals with 8 kHz. This what Android offers among 2 other rates which are higher and is in agreement with the Nyquist-Shannon rate because the voice frequency band is approximately up to 3.4 kHz. [7]

After delaying these samples from the signal on smartphone 1 we have aligned the signals from both smartphones. After summing the signal, the signal to noise ratio (SNR) of the signal coming from the desired direction will increase because this signal will be attenuated and a virtual beam is directed towards the desired direction. Once the sampled data is received from both smartphones, the delaying in samples according to the wave front angle is achieved by the Delay and Sum (DAS) algorithm developed by the Beam Forming algorithm group 1 [7]. This algorithm is programmed in Java in Android-SDK as a class and can be implemented directly in an Android device.

Next the interconnection of the smartphones for the samples data transmission is researched and discussed.
Interconnection between android smartphones can be achieved wired and wireless. Wireless interconnection of smartphones and portable computers, is becoming more and more important and almost all the possible wired connection is replaced by wireless. Wireless offers freedom but at the same time it is more challenging to transmit data through a wireless connection as it easily can get corrupted or lost. This depends highly on the type network used. [12] Smartphones and portable computers have several opportunities to set up a wireless network for data transmission. Android offers libraries for use of Bluetooth module for real time data transmission. Bluetooth is wireless open technology for data transmission between smartphones in short ranges. The Bluetooth allows creation of highly secure personal area networks. Here the focus will be on setting up a wireless network using Bluetooth module.

Portable computer like smartphones have a limited storage capability for energy, so the energy consumption should also be considered when dealing with data transmission. Furthermore for the DAS beam forming algorithm it is important that the delay in the network is as small as possible when transmitting the data. The DAS algorithm uses the delay in signal received in the smartphone which is further from the original sound source than the smartphone, who receives the signal first, to virtually steer the beam to the desired direction. This is explained in section 5.1. This needs and requires the smartphones to start recording at the same time instant, sample with the same sample frequency and in the real time applications like teleconferences the samples need to be sent in real time with lowest possible delay to the main smartphone to perform the DAS algorithm on the samples. After the performance of the algorithm which also introduces delays, [7] the beam formed captured signal should be sent to the receiver on the other side. This is a process that introduces delays in every step and is shown graphically in figure 5.8 below.
First in 5.2.1 the opportunities to build a network with Android devices are researched and the requirements from the network for teleconference purposes using DAS beam forming with android smartphones are discussed. In section 5.2.2 the Bluetooth module for smartphones is introduced and further discussed for implementation.

5.2.1 Network opportunities for Android Device and requirements for implementation of DAS beam forming for teleconference purposes

The Android smartphones are equipped with a radio transmitter and receiver for wireless data transmission. Most of the Android devices have three different options for supporting the wireless connections [11].

The 3G antenna is used for the normal call function performed by the smartphone. It is based on the UMTS network principles. The smartphones are using Cellular networks for standard voice call transmission which is based on Backhaul transmission. Backhaul is usually taking to mean the link between a base station and its associated network controller [12]. In the late 1990s the first implementation of data on the GSM phones based on circuit-switched approach was achieved. The General Packet Radio Service (GPRS) was introduced; GPRS added packet switching, based on the Internet Protocol (IP), to GSM. The UMTS was the follower of GPRS and has both a circuit-switched element (for voice) and a packet-switched element (for data).

A standard for wireless communication link is the Bluetooth module. This module is used for data transfer between the smartphones. The file transfer profile depends on several underlying profiles and protocols. The Android API does support a Bluetooth mode for data transmission.

The WIFI wireless modem build in the smartphone enables them to connect to the internet. The WiFi is for wireless data networking products that work according to the international standard IEEE.802.11 protocol. The common WiFi standard in Android device is based on IEEE 802.11g standard [11]. This standard transmits on the 2.4 GHz band and can transmit up to 54 Mbit/s [11] sections 4.4.2. The Android API does not at this point support ad-hoc mode for WIFI to work. [11]

For the teleconference purposes investigation made for implementing network using Bluetooth to interconnect the smartphones for data transmission. The UMTS can be used to link the main smartphone with the other side of the teleconference.

The requirements of the network for implementing the DAS algorithm come from the purpose of the use of the DAS algorithm and the Android smartphones capability. The DAS algorithm will be used to offer beam forming capability which enables spatial filtering of the sound, this in turn can be used for teleconference purposes. For simplicity investigation are made for performing beam forming with DAS algorithm and 2 Android smartphones.
Smartphones regardless of the operating system, run on processors with clock speeds ranging from 100-624 MHZ. Each smartphone has its own internal clock signal and two identical smartphones will never have synchronized clock signal. This is also the case with Android smartphones. For the purpose of beam forming both the smartphones need to start recording at the same time. This is necessary because as explained in 5.1 the samples need to be taken at the same identical time instant for both the smartphones. This can be achieved once the two smartphones are interconnected by the network, a triggering event on the application can start recording for both smartphones.

Once both Android smartphones start recording, the incoming data is already sampled with sample frequency specified in the function implementation as explained in section 3. So the sample frequency can be specified in the function implementation of Smooth Speech App to be the same for both smartphones. Immediately after having enough samples the slave smartphones needs to send data through the network to the master smartphone with delay as lowest as possible.

Also position detection is needed to be able to perform DAS algorithm for simplicity we assume both smartphones to have the identical x and z coordinates and only differ in y coordinates. This is shown in figure 5.6 and explained in section 5.1.2. There are yet (to this time) no techniques that can accurately offer position detection of one smartphone relative to other smartphone in one axis coordinates. This is needed for the DAS beam forming algorithm. There are techniques developed like signal strength based localization [36] which offers accuracy up to 2 to 3m, using WLAN network for mobile devices. The Time Difference of Arrival (TDoA) offers target localization accurate up to 1 to 3 meters; however it needs a synchronized connection between smartphones with accuracy in range of single-digit ns in order to achieve this accuracy [36]. Even if this synchronized connection is setup, the accuracy of 1 meter is not acceptable for DAS algorithm.

The clock synchronized connection between the two smartphones is needed for sending the data from one smartphone to other and for a possibly position detection. However a clock synchronized connection with accuracy of ns is impossible to achieve with commercial-of-the-shelf (COTS) WLAN devices like smartphones [36], this would require hardware change and implementation of highly stable oscillators. In Applications where TDoA is used for localization usually the Base Stations are connected through a wired backbone to ease the synchronization between the Base Stations.

The establishment of a clock synchronized wireless network for android smartphones is not possible with the smartphone hardware and the available android libraries and is a topic for a new research. Studies should that till now whenever an Android application is built to communicate with other android smartphones a custom made libraries are build which are added to the kernel and Bluetooth protocol is used for the network.[11] The Bluetooth specifications are adopted and the design is based on them. Aldo Bluetooth does not offer a clock synchronized connection but it does offer a real time speech signal transmission and data transmission.

The position detection is left for further research and for simplicity we assume a holder build for the two smartphones with the position of each smartphone fixed. The difference in y coordinates are known and implemented directly in algorithm. The holder is also adjustable because we then can freely choose the distance (by keeping the limitations as explained in section 5.1.2 back in the mind) between the smartphones. Therefore a function can be built in GUI design of Smooth Speech App.
where the distance is passed by the users to the algorithm. The holder is graphically shown below in figure 5.9.

![Figure 5.9: the custom made Android smartphone holder for teleconference purposes.](image)

The custom made smartphone holder may suggest a wired interconnection but the focus still lies in wireless interconnection of the smartphones. In the next section we research the Bluetooth module and try to implement the network by using Bluetooth. In Section 5.3.1 a research is made based on wired interconnection of two smartphones using adjustable smartphone holder.

### 5.2.2 Bluetooth network for smartphones and Android

Bluetooth is a master-slave, packet-based protocol. It allow 8 devices (Smartphones) to interconnect with one being master and seven slaves. All the 8 devices follow the clock signal of the master which is the event trigger for transmitting data. This Event trigger clock based on the master clock ticks at a 312.5 µs intervals. Two intervals make a slot of 625 µs time slot for data transmission. Retransmissions are performed if necessary to insure data integrity. To achieve maximum data rate, the number of retransmissions should be reduced, this is possible when a electromagnetic interference in the range of 2.4GHZ exists. Bluetooth was primarily designed for low power usage; short range wireless communication protocol based on low-cost transceiver microchips in each device. Bluetooth supports the synchronous services such as voice traffic through the use of physical link type the Synchronous Connection Oriented (SCO). The SCO link is asymmetric point to point link between the master and slave. [13] The SCO link is an circuit-switched connection between the master and the slave and allows the master to support three SCO link to the same slave and different slave. Bluetooth is accessible through the standard Android Libraries. Applications such as Voice over
Bluetooth are developed for real time voice transmission over the Bluetooth network. [13] This is done using the SCO link type of connection. Bluetooth offers a low complexity, cost and power consumption ability to build a Personal Area Networks (PANs) for smartphones. Theoretically Bluetooth offers a maximum rate of 1 Mbps for data transmission.

The implementations of time-critical and real time applications like Smooth Speech App for Beam forming purposes through wireless networks should ideally be performed by connection oriented links with predefined bandwidth efficiency. Bluetooth defines a synchronous-connection oriented mechanisms for servicing time critical application (for sampled audio transmission) at a constant rate equal to 64 kbps. This limited bandwidth performance is the main drawback for Bluetooth technology, relevant to transmission of quality sampled digital audio. [37] At this moment all the time-bounded applications which are using Bluetooth for real time audio data transmission and requiring a constant throughput must be established through SCO link. The effective bandwidth is then only 64 kbps for Bluetooth technology available on this time. Other drawbacks are; the establishment of SCO link dramatically decreases the transfer capabilities of any co-existing ACL link. In noisy environments, packet retransmissions are applied to ensure data integrity; this will reduce the effective bandwidth. The above explained drawbacks will limit the use of Bluetooth for high quality digital audio transmissions because the reproduction will need and is performed in practical application at bitrates in the range of Mbps.[37] Considerations are necessary for realizing high quality audio transmission. The original audio data must be compressed prior to the transmission and the application using the Bluetooth module for data transmission should adopt the Bluetooth specifications.

The Android platform offers support for the Bluetooth network stack allowing setup of PANs. The application framework provides access to the Bluetooth functionalities through the Android Bluetooth API’s. [38] These APIs enable point-to-point connection between android smartphones for data transmissions. Using the Bluetooth API, Android application can perform; Scan for other Bluetooth devices, Establish RFCOMM channels, Connect to other devices through services discovery, Transfer data to and from other devices and Manage multiple connections. Using the RFCOMM layer applications can be writing for Android that transfer data. The application is writing in Java in Android SDK. [38] Its a part of the Smooth Speech App program. In Android the role of master and slave for the devices is not defined explicitly when setting up a Bluetooth SCO link by RFCOMM and depends on the application. The roles can be changed during the application and is invisible for the users. The designer should determine in his application whether the device will act as a client, server or both and the services (profiles) that is supports.

Next the process of setting up an SCO connection for audio data transmission between two smartphones in Android is described shortly, for more information the reader is referred to Android developers guide available online. [38] Four major tasks are necessary to communicate using Bluetooth in Android:
- Setting up Bluetooth
- Finding available devices in the local area
- Connecting devices
- Transferring data between devices

**Setting up Bluetooth**

Setting up Bluetooth involves, finding the Bluetooth adapter available on the smartphone and interacting with it for data transmissions, Giving the right permissions to use the Bluetooth adapter in manifest File as explained in chapter 2 and enabling Bluetooth by asking the Mac-address of the device which can be used as identification of the device when connected.

**Finding available devices**

This procedure involves, scanning the local area using the Bluetooth adapter for other Bluetooth devices, which are enabled to be discovered. Once the other devices respond to the discovery request by sharing the device name, class and MAC address. Using this information the device can choose to initiate a connection with the device. When a connection is established the devices are said to be paired meaning that the device name, class and MAC address of the device is saved in the program and can be read using the Bluetooth APIs. Using this MAC address a connection can be initiated with it any time without performing discovery. Once the devices are paired they can be connected and share data by RFCOMM channel.

**Connecting devices**

In order to create a connection between two devices, both server-side and client-side mechanisms should be used. One device being server and open a server socket and other device being client to initiate connection. The client and server considered connected to each other when they each have a connected BluetoothSocket The client opens a RFCOMM channel to the server and will receive the BluetoothSocket on the same RFCOMM channel. The server will accept the incoming connection and will receive the BluetoothSocket. At this point each device can obtain input and output streams and data transfer can begin.

**Transferring data between devices**

Once a connection is established between a client and a server a bi-direction data transfer can take place through the BluetoothSocket. The InputStream and OutputStream can be used to handle the transmissions through the socket by using getInputStream() and getOutputStream() methods. To read and write data to the streams the read(byte[]) and write(byte[]) can be used.
5.2.3 Implementation of the Smooth Speech App in Bluetooth

Implementation of Bluetooth for the business version of the Smooth Speech App is challenging. Here we will discuss setting up a PAN between two smartphones only for transmission of sampled audio signals from one to other. The data is then used to perform the Beam forming using the DAS algorithm. This is graphically shown below in figure 5.10.

![Diagram showing point-to-point single audio data transmission for Android Application through Bluetooth SCO link](image)

Figure 5.10: Point-to-point single audio data transmission for Android Application through Bluetooth SCO link

By using the AudioRecord class as explained in section 3.2, the audio coming from the microphones can be sampled with 8 kHz in an 8-bit format. The samples are saved in RAM memory and can be sent directly to the Bluetooth module for transmission. The sampled audio is the only data transmitted. The bitrate can be calculated using formula 5.9.

\[
\text{Bit rate} = f_s \times (\text{bit depth}) \times (\text{number of channels})
\]

By using the AudioRecord class as explained in section 3.2, the audio coming from the microphones can be sampled with 8 kHz in an 8-bit format. The samples are saved in RAM memory and can be sent directly to the Bluetooth module for transmission. The sampled audio is the only data transmitted. The bit rate can be calculated using formula 5.9.

\[
\text{Bit rate} = f_s \times (\text{sampling rate}) \times (\text{bit depth}) \times (\text{number of channels})
\]  

5.9

Bit rate for producing the samples would equal 64 kbps for each smartphone using one channel microphone, the specified sampling frequency and bit format. These samples are then transmitted using the setup point-to-point SCO link by Bluetooth. As explained before the SCO link is limited to 64 kbps and the smartphones are producing with 64 kbps.
The smartphone that will perform the DAS algorithm will be the receiver and the server in our BluetoothSocket RFCOMM transmission channel. The second smartphone which sends the sampled data will be the client. The choice is arbitrary because the both client and server can send and transfer data. The choice has to do with the fact how the desired connection is setup; once the main smartphone (being the server) is set enabled for discovery, the other smartphone can discover the server and ask permission to setup a connection as explained above in section 5.2.2. Once a connection is setup the transmission can begin.

The data is created by bit rate of 64kbps at the client side. After the transient startup response of the network initialization and setting up which will take an amount of time, in the dynamic response phase of the network the maximum bitrate is used. This leaves no rooms for mistake and error in the transmission which triggers the retransmission mechanism and introduces delays in data transmission. When we are in the dynamic response of this system, data is created at 64 kbps and send through the network by 64kbps. Due to small distance (<1 meter) between the smartphones in the array setup the chance for an error in transmission is very small.

At the server side (main smartphone) the incoming data is buffered and saved in memory, idealistically a buffer of 1 second is needed. The delay due to sampling is negligible but the delay due to DAS algorithm programmed in java SDK should take in consider, using Android NDK can speed up the process as explained in section 6. Once the samples are received and saved in memory, they are aligned with the samples talking from the internal microphone of the main smartphone. This is achieved by a recording trigger event; once the server and client are connected as explained in section 5.2.2 and are able to send data in real time, first a trigger signal is send to start recording with both microphones at the same time instant. After performing the Das algorithm the samples can be send to the destination using standard UMTS Network which is based on Integrated Services Digital Network (ISDN) principles.

The information provided here is based on theoretical research done for further investigation of implementing beam forming in Android using multiple smartphones. This information should provide enough knowledge about the use of Bluetooth and ability to use wireless networks to transfer digital sampled audio in real time.
5.3 Alternative solution for interconnecting smartphones for performing DAS beam forming algorithm

In this section alternative solution for interconnecting smartphones for performing DAS algorithm are purposed. In section 5.2 we discussed Bluetooth module for interconnecting multiple smartphones and real time data transmission. In this section we will look at the alternative solution for interconnection of smartphones using wired and wireless connections. This is done as alternative solution for performing DAS algorithm using multiple Android smartphone. The focus will be only on interconnecting the smartphones for real time bi-direction transport of audio samples. For simplicity purposes the interconnection of two Android smartphones will be researched. The purpose of this section is to represent a research result, due to shortage of time and resources no real implementation for testing purposes are performed. However these results can be used for further research and implementation purposes.

5.3.1 Wired Interconnection solutions

A wired interconnection of two smartphones is physically not possible without intervention of a computer system. This computer system makes a synchronized connection between the computer and the smartphone, according to the internal clock rate of the computer. So to be able to interconnect two Android smartphones, a small computer has to be used. As an alternative solution a custom made Micro Controller Unit (MCU) can be used. A MCU is a small computer on a single integrated circuit containing a processor core, Memory and programmable input/output peripherals. Its main functionality is performing tasks for embedded systems.

A popular open-source microcontroller, descendant of the open-source Wiring platform is Arduino Uno. The Arduino Uno is a microcontroller board based on the ATmega328. The Arduino uno is graphically shown below in figure 5.11.

**Arduino Uno**

![Figure 5.11: the Arduino Uno](image-url)
Arduino has the following specifications:

- ATmega328 Microcontroller
- 5V Operating Voltage
- 32 KB (ATmega328) Flash Memory
- 2KB (ATmega328) SRAM
- 1KB (ATmega328) EEPROM
- 16MHz Clock Speed
- USB connection

For more information about the Arduino Uno the reader is referred to the official site. [39]

The Arduino Uno can be used as the interfering Microcontroller unit when interconnecting the Android smartphones. Using Android USB host API and Arduino USB Connection, an interconnection between Arduino and Android is possible for data transmission. [40] Samsung galaxy S2 supports the Android USB Host App which makes the smartphone act like a USB Host. Using the firmware of the Arduino a connection through the USB can be made with the Android phone for interchanging data. [40]

Interconnecting two Android phones requires two Arduino Uno MCU boards for each Smartphone to connect with. Afterwards the both Arduino’s will be connected with each other for mutual data transmission. The Samsung galaxy s2 offers a strong processor (Dual Core), which is much faster than the ATmega328 Microcontroller equipped in Arduino Uno. The DAS algorithm should be performed on the main smartphone and 2 Arduino’s are used for interconnection and data transfer only. The 2 Arduino’s can be connected wired by use of the standard input/output pins. The setup of the situation is shown below in figure 5.12.

As can be seen in figure 5.12, the adjustable Smartphone holder is used for smartphone array. This holder was introduced in section 5.2.1 and had the benefit that the distance between the smartphones is adjustable, making it more suitable for performing the DAS Beam forming. The interconnection can be seen between the Arduino’s mutually, between the Arduino and Smartphones. Consequently the two smartphones are wired connected.

This type of connection looks wired and outdated, it introduces larger amount of delay for the final application. The signals now have to travel through 2 MCU’s and 4 wires before they will reach the main smartphone, this will introduce Delays. Also at the same time the chance for a corrupted data or loosing data is smaller because of the wired connection.

Arduino now days using also Bluetooth modules to interconnect with smartphones, as mentioned earlier wireless connections are becoming more favorable over wired.
5.12: Setup of the Smartphone array wired interconnected using Arduino's
5.4 The Suited Network for Implementing the Smooth Speech App

The Android Smartphones offers three network connecting opportunities which were discussed in section 4.1.2. However 2 of this wireless connection types cannot be accessed by the Android API directly. These were the 3G and the WIFI module.

The 3G is a GSM phone based network which is used for real time call functions, using the infrastructure provided by the phone operators. This network functionality cannot be accessed by the Android libraries.

Students and Researchers have developed libraries for Ad-hoc network on Android implementation using WIFI IEEE 802.11g. [11] Although these layers are not used for real time audio data transmission but they offer an Ad-Hoc protocol layer for Android. This is a beginning concept in evolution.

Bluetooth is accessible through the standard Android Libraries. Applications such as Voice over Bluetooth are developed for real time voice transmission over the Bluetooth network. [13] This is done using the SCO link type of connection. These already developed modules for the Bluetooth can be used to implement the Smooth Speech App network.

The clock signal synchronization is established in network when Real Time Voice Traffic is available on the network. By using this information the table 4.1 is constructed below.

Table 5.1 The Android Network Connecting Opportunities and their properties

<table>
<thead>
<tr>
<th></th>
<th>3G</th>
<th>WIFI</th>
<th>Bluetooth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API Accessible</strong></td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Under Construction/Evolution</strong></td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Data Transmission</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Real Time Voice Traffic</strong></td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Small Smartphone Networks already designed</strong></td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Position detection</strong></td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>

The implementation of the Smooth Speech App network at this time can only be managed using the Bluetooth module and in particular the SCO link type of connection.
6 Android Programming Improvement and Analyses

The Smooth Speech App is developed in Android, meant for smartphones running on Android. Smartphones are small pocket computers with limited battery and storage capability. [14] The Beam Forming Algorithm performed by the Smooth Speech App is programmed in java language. The Beam Forming algorithm performs mathematical functions on the incoming audio data samples. The mathematical function such as the Fast Fourier Transformation (FFT) are performed and coded with java language. The Android SDK uses Java language to build the Android applications however; applications written in Java language are slower than applications written in native C/C++ language. [15] The Android NDK offers the Android Developers application to code in C/C++ and to improve the performance of the Android Application when used appropriately. [14]

In this chapter the Android programming improvements, code improvements will be investigated. The java programming language will be investigated and the object-oriented analysis is described, which allows analyses and improvements of code. [16] Also Android-NDK, which is effective in CPI-intensive operation such as signal processing [14], needed for the Smooth Speech App is investigated. The Android NDK provides stable headers for the C library, Math library, JNI interface, minimal set for C++ support and OpenGL ES library. [1] These stable headers provide benefits to certain classes and speed them up by reuse of existing code. [18] In the Smooth speech App code, the algorithm part has to perform Fast Fourier Transformation (FFT) on the audio data signals. The Fast Fourier Transformation operation is a CPI-intensive process. The Smooth Speech App uses the Michael Thomas Flanagan’s Java Scientific Library for implementation of FFT algorithm in java. [17] Investigation is made about the benefits of performing this process in Android NDK. The Smooth Speech App package containing all the classes are analysed using object-oriented analyse which is shortly explained in section 6.1. In section 6.2 the Android NDK is explained and the implementation of FFT using NDK is explained. In section 6.3 or written Java classes are further analysed.

6.1 Java and Object-oriented analysis

The Android SDK uses Java language to build the applications for Android devices. Java language is a general purpose and object-oriented language designed to have few implementation dependencies [6]. Java is designed on the basis of “Write once, run anywhere” (WORA) meaning it can run on various platforms without the need of recompiling [10]. Currently one of the most important and popular languages in use is java. Object-oriented programing is based on using objects to access data in the main program. The object is than the programming construct that combines data with set of methods for accessing and managing those data. An object-oriented program may contain more than one object or even multiple copies of the same object with different constructors [6]. Object-oriented analysis offers a way for programmers to analyze their program writing in object-oriented way [10]. As it seems to be a difficult task still the concept of Object-oriented analysis process is evolving. For being able to analyze on Object-oriented program the appropriate skills are needed. The Analyst must be able to manage the complexities involved in the Object-oriented analyze.
Understanding the process environment for which the program is writing and being able to perform abstractions is a must [10]. Object-oriented analyze is performed before the error occurs and is seen as a process to minimize the chances for a malfunction to occur.

There are 4 methods for performing an analyze on the system, functional decomposition, Dataflow method, information modeling and object-oriented analyzes [10]. Object-oriented analyze being the most advanced and desirable for programing in Java language.

The object-oriented analyze is based on five activities:

- To find all the classes and objects involved
- Define the main structure of the program
- Indicate the subjects
- Defining Attributes
- Defining Services

All of these activities have to be performed. The sequence in which these activities are performed is not important and left as a choice for the analyzer self. These activities lead the analyzer from high level of abstraction to lower level of abstraction. [10]

6.1.1 The Product List

Before any software program is writing a list of specifications are made based on exceptions form the software writing called the product specification. The software also has a responsibility towards the function it is implemented to do. The function could simply be Algorithm which has to be performed on the input and sent to the output or to perform a specific task whenever a variable value changes. The product list of the Smooth Speech App can be found in Appendix E.

6.2 The Android NDK

The Android NDK is a toolset that uses native libraries from C/C++ codes for performance of critical portions in the Android Applications. [18] It also enables the reuse of legacy code written in C/C++ language. [14] However in each case the consideration of using native code must be applied. Using native code brings advantages and disadvantages; it does not always increase application performance but always increase applications complexity. [14] Typically good candidates for using the NDK are self-contained, CPU intensive operations that do not allocate much memory, such as signal processing methods. [1]
6.2.1 Native code

Native code which is platform depended machine code (In this case Android) is an system of instruction set that can performed directly by the Central Processing Unit of the Smartphone. [19] There are many basic operation native codes written for another platform such as Linux to perform the basic operations and methods. The Android NDK enables reuse of the part of these codes for Android Application written. Most of these native codes are written in C/C++ so for reusing them the Android NDK enables a port to an existing body of C code written for another platform.

The following instruction set is supported by latest release of NDK (Revision 8 May 2012): [1]

- ARMv5TE
- ARMv7-A
- X86
- MIPS

In Appendix D more information can be found on these various instruction sets.

The target smartphone for building the Smooth Speech App is the Samsung Galaxy S2. The Samsung Galaxy S2 contains a 1.2 GHZ dual-core ARM Cortex-A( CPU) and an ARM Mali-400 (GPU). [4] This ARM Cortex-A supports the NDK.

6.2.2 Implementation of FFT using NDK

The standard Android API does not offer a class for performing hardware FFT on data. [1] However it does offer a class implemented in the native code which can be used by Java Native Interface (JNI) and Android NDK. The class is called the Visualizer and can be found in Android.Media.Audiofx package. [1] The class offers methods like getFft and doFft. The FFT is a CPI intensive operation as explained above. These operations are faster when they are implemented in C and C++ native code and accessed by java native interface. The Android’s Dalvik VM is optimized for memory footprint rather than processing speed.
7 Alternative solutions to acquire microphone signals

As said in section 2.4, it is not possible to acquire multiple microphone signals at the same time due to limitations of the Android platform and the Samsung Galaxy S II phone and that is why a different solution must be found to bypass these limitations. In this chapter the different bypass options will be discussed and the best will be selected. In section 7.1, methods to design a microphone array will be explained and in section 7.2 and 7.3 acquiring the microphone data with a pc running Matlab and external hardware will be explained respectively. Finally in section 7.4, the best solution will be chosen.

7.1 Building external microphones

To build a microphone array, multiple microphones are needed. To simulate telephone conditions, high quality telephone microphones are used. These microphones are then connected to a 3.5mm TRS jack, or simply 3.5mm jack [9]. The 3.5mm jack has three channels; left, right and ground. The positive wire of the left microphone is connected to the left channel and the positive wire of the right microphone is connected to the right channel. The negative wires of the left and right channel are both connected to the ground channel. The wiring of the microphones is shown on the bottom of figure 7.1. The 3.5mm jack in the top of figure 7.1 is the input jack, which connects to an AD converter or computer.
The Samsung phone has two microphones that are 125 mm apart and therefore the microphones in the array are also 125mm apart. Figure 7.2 shows the finished microphone array that has two microphones and a 3.5mm TRS connector.

![Microphone Array Image](image)

Figure 7.2 the microphone array

### 7.2 Acquiring microphone data with a PC running Matlab

Because of the limitations of both hardware and software discussed in section 2.4, another option to retrieve two audio signals simultaneously has to be found and one of those options is a PC running Matlab. The microphone array discussed in section 7.1 can be plugged into the microphone input of the PC and Matlab can access them with the code found in appendix B. The samples collected by Matlab are then saved into two separate files and stored on the Samsung Galaxy S II. The application reads these files and performs the beamforming algorithm on the raw samples and plays the recorded audio sample after the beamforming was applied.

The advantages of this method include the fact that it is accessible, since Matlab runs on all used computers. Furthermore, this method adds no additional cost and implementation is simple. The main disadvantage is the fact that the sample files compiled by Matlab have to be transferred from the PC to the phone manually. Additionally, since this process is not real-time, it cannot be used in practice. The code used to acquire the microphone signals can be found in appendix B.
7.3 Acquiring microphone data with external hardware

Another option to retrieve two audio signals simultaneously is to add external hardware to the USB of the Samsung galaxy S II and the ioio add-on board is an example of such an external hardware board. The ioio ad-on board is a ready-made solution that can be programmed directly from the Android SDK [10]. The microphone array that was constructed in section 6.1 can also be used with the ioio board, when connected to two A/D converters that convert analog data into digital samples. These samples are then send to the beamforming algorithm and this can be done real-time.

The main advantages are the fact that noise reduction can be done in real-time and for this reason this solution can be used in practice. The main disadvantages include the high price, which is fifty dollars and the fact that it had to be ordered in the United States and it required a lot of additional soldering and programming.

7.4 The best alternative to acquire the microphone signals

Two possible alternatives to acquire data have been proposed. The points that were taken into consideration can be found in table 7.1. A PC running Matlab offers a cost-free and accessible solution, while the ioio ad-on board is costly and has to be ordered from the United States. However, the ioio board offers a real-time solution that can be used in practice. Because of limited time, the chosen solution is a PC running Matlab, but the ioio ad-on board is still a better overall solution.

| Table 7.1 points considered while choosing the best way to acquire microphone data |
|----------------------------------------|----------------------------------------|
| **Accessibility** | **Cost** | **Real-time** | **Can be used in practice** |
| a PC running Matlab | the ioio ad-on board |
| Accessibility | ✓ | X |
| Cost | ✓ | X |
| Real-time | X | ✓ |
| Can be used in practice | X | ✓ |
8 Conclusions and recommendations

This chapter will give conclusions based on all research done in all the previous chapters. Section 8.1 sums all the conclusions and will draw a main conclusion. Recommendations based on these conclusions will be given in section 8.2

8.1 Conclusions

This thesis answers the question; how to make an Android application that presents real-time microphone data to a beam forming algorithm and make it user friendly at the same time? The following paragraphs are summaries of the conclusions made in all the previous chapters.

As described in chapter 3.1, the kernel cannot be accessed directly, because of the layering model in Android. Moreover, the only way to access the kernel is via the core libraries provided by the Android SDK. The Android Media library offers two classes for audio data acquisition, Media Recorder Class and Audio Recorder Class. While these classes offer a range of various opportunities for audio recording coming from the internal microphone of the system, they do not offer possibilities for multiple microphones recording at the same time. The possibility of recording multiple microphones at the same time is desired for the Beam Forming algorithm. The smooth speech app cannot be implemented into the Samsung Galaxy S2.

The two microphones of the Samsung Galaxy S II cannot be accessed at the same time that is why we had to find another solution to acquire multiple microphone signals at the same time. A PC running Matlab offers a cost-free and accessible solution, while the ioio ad-on board is costly and has to be ordered from the United States. However, the ioio board offers a real-time solution that can be used in practice. Because of limited time, the chosen solution is a PC running Matlab, but the ioio ad-on board was a better overall solution.

For the graphical user interface three options were chosen. The first being a slider widget, the second using the gyroscope and the third using a graphical rotating knob. All three options are easy to use, but only the graphical rotating knob gave a proper graphical representation while the source is selected. Both the gyroscope and the graphical rotating knob are difficult to develop, while the seekbar is very easy to develop. Although a combination of a gyroscope and a graphical rotating knob offers the best solution, the seekbar was the chosen option, because it is easy to develop and our time was limited.

Interconnection of multiple smartphones for audio data transmission and manipulation is necessary to connect multiple smartphones. Also the position detection of each smartphone relative to the main smartphone is a must for the Business version of the App. Several options; Bluetooth, wifi, GMS and UMTS were explored for the real time audio data transmission. Bluetooth turned out to be the best option for implementing the network and throughout the use of NEST application target localization can be performed [23].
The Smooth Speech App was developed for Smartphones, coded in java language. The smartphone is a small pocket computer with limited energy and storage capability. Therefore the coding should be done in an efficient way [9]. The mathematical function needed for the Beam Forming Algorithm, such as the Fast Fourier Transform (FFT) is programmed in Java using the Michael Thomas Flanagan’s Java Scientific Library. [35] To improve efficiency and time consumption, an Android NDK, which reuses C and C++ native code can be used [33].

In conclusion, to acquire the microphone signals, Google must implement the beamforming algorithm on kernel level and provide a smartphone that has more the one A/D converter. This thesis offers a solution to bypass the limitations of Android and the smartphone by using a PC with Matlab, but a better solution is adding a hardware ad-on board such as ioio, because it offers a real-time solution. Furthermore for interconnection of the smartphones for use in conferencing purposes, the Bluetooth module can be used.

8.2 Recommendations

These recommendations are made based on our conclusion and experience gained during the project. For further research on audio processing, such as beamforming on Android smartphones it is recommended to use the ioio hardware ad-on board with two separate A/D converters and two identical microphones, since this solution offers a real-time solution. Furthermore, we recommend the implementation of the Beam Forming Algorithm directly in the Android API as a standard library. This Android standard library should be used with smartphones containing the appropriate DSP hardware. The GUI should be a combination of a gyroscope and a graphical rotating knob.

The network needed for teleconferencing purposes should be implemented in Bluetooth. The Bluetooth is at this time the only available library on the Android API for accessing networks with android device. Novel techniques are being developed to use Bluetooth modules for real time voice transmission and Target localization, making Bluetooth an even better option.
References


[27] Peter Brinkmann and Peter Kirn, “Embedding Pure Data with libpd.” uni-weimar.de., 2011


<table>
<thead>
<tr>
<th>Network</th>
<th>HSPA+ 21Mbps/ HSUPA 5.76Mbps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EDGE/ GPRS Class 12</td>
</tr>
<tr>
<td></td>
<td>Quad band GSM 850/900/1800/1900</td>
</tr>
<tr>
<td></td>
<td>Quad band UMTS 850/900/1900/2100</td>
</tr>
<tr>
<td>AP</td>
<td>Dual Core Application Processor</td>
</tr>
<tr>
<td>Dimensions</td>
<td>125.3X66.1X8.49mm</td>
</tr>
<tr>
<td>Display</td>
<td>4.3&quot; WVGA SUPER AMOLED Plus *</td>
</tr>
<tr>
<td>Memory</td>
<td>16GB/32GB</td>
</tr>
<tr>
<td></td>
<td>MicroSD (up to 32GB)</td>
</tr>
<tr>
<td>Camera</td>
<td>8MP AF with LED Flash + 2MP Front</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Wi-Fi a/b/g/n</td>
</tr>
<tr>
<td></td>
<td>BT v3.0+HS</td>
</tr>
<tr>
<td></td>
<td>USB v2.0</td>
</tr>
<tr>
<td>Connectors</td>
<td>MicroUSB, 3.5mm Ear Jack</td>
</tr>
<tr>
<td>* May not be applicable in some regions.</td>
<td></td>
</tr>
<tr>
<td>* Display: &quot;4.27&quot; in actual measurement</td>
<td></td>
</tr>
<tr>
<td>Battery</td>
<td>1650mAh</td>
</tr>
<tr>
<td>OS</td>
<td>Android Platform 2.3</td>
</tr>
<tr>
<td>Message</td>
<td>SMS/MMS (OMA v1.2)</td>
</tr>
<tr>
<td></td>
<td>Email (POP3/IMAP/SMTP, SSL/TLS)</td>
</tr>
<tr>
<td></td>
<td>Exchange ActiveSync Email</td>
</tr>
<tr>
<td>Audio</td>
<td>MP3, OGG, AAC, AAC+, eAAC+, AMR-NB</td>
</tr>
<tr>
<td></td>
<td>AMR-WB, WMA, WAV, MID, AC3, IMY, FLAC, XMF</td>
</tr>
<tr>
<td>Video</td>
<td>MPEG4, H.264, H.263, WMV, DivX, VC-1</td>
</tr>
<tr>
<td></td>
<td>Recording &amp; Playback 1080@30fps</td>
</tr>
<tr>
<td>Image</td>
<td>JPEG, PNG, GIF, WBMP, BMP, AGIF</td>
</tr>
<tr>
<td>GPS</td>
<td>A-GPS</td>
</tr>
<tr>
<td>Convergence</td>
<td>Easy Set-up (WPS PIN/PBC, Wi-Fi Direct)</td>
</tr>
<tr>
<td></td>
<td>AllShare (DLNA1.5)</td>
</tr>
<tr>
<td></td>
<td>Samsung Kies 2.0, Samsung Kies air</td>
</tr>
<tr>
<td>4 Hubs</td>
<td>Social Hub / Readers Hub /</td>
</tr>
<tr>
<td></td>
<td>Music Hub / Game Hub</td>
</tr>
<tr>
<td>Others</td>
<td>TouchWiz</td>
</tr>
<tr>
<td></td>
<td>Google Mobile Services</td>
</tr>
</tbody>
</table>

Table 1: Samsung Galaxy S II specifications [35]
Appendix B: Matlab code

function recorded = audio(N, Fs, ch)

recorded = wavrecord(N, Fs, ch);
t = (1:1:N);
plot(t,recorded(:,1));
hold on
plot(t,recorded(:,2), '-.r');

save('H:\Desktop\test.txt', 'recorded', '-ascii')
end
Appendix C: AudioRecord and MediaRecorder class

Appendix C.1 the AudioRecord Class

To construct a new AudioRecord object the following parameters have to be added to the constructor:

- Audio Source
- Sample rate (in HZ)
- Channel configuration
- Audio format
- Buffer size

The Audio Source
The Audio Source is defined by the MediaRecorder.AudioSource final class. The following option can be chosen for the Audio Source:

- CAMCORDER
  (Microphone audio source with same orientation as camera)
- DEFAULT
  (Default Microphone audio source)
- MIC
  (Microphone audio source)
- VOICE_CALL
  (Voice call uplink + downlink audio source)
- VOICE_COMMUNICATION
  (Microphone audio source tuned for voice communication such as VoIP)
- VOICE_DOWNLINK
  (Voice call downlink (Rx) audio source)
- VOICE_RECOGNITION
  (Microphone audio source tuned for voice recognition)
- VOICE_UPLINK
  (Voice call uplink (Tx) audio source)

The Sample Rate
The Sample rate configures the audio data sample rate in Hz. This is the rate at which the samples are taking from the audio. The Sample rates are defined for human speech signals. For most of the human beings, almost all of the energy is contained in the 5Hz – 4 kHz bandwidth. The sample rate is then chosen to be 8kHz which is also the sample rate used by nearly all the telephony systems.

According to Android [1], 44100 Hz is currently the only rate that is guaranteed to work on all devices, but other rates such as 22050, 16000, and 11025 may work on some devices. However after
programming and try outs on the Samsung Galaxy S2, we found that 10000 Hz was the maximum sample rate that would guarantee qualitative samples. Above the 10000 Hz sample rate the device would generate an error and the Audio record failed.

**Channel Configuration**

The channel configuration defines the recording channel configurations. The choice is between mono and Stereo configuration. Mono is based on one channel recording and Stereo is based on two channel recording.

**Audio Format**

The Audio format defines the digital format in which the analog audio data is coded. The coding is done by mean of Pulse-code modulation (PCM). PCM is a method used to digitally represent sampled analog signals. PCM is the standard used in digital system for audio signal representation. In the Android Media Library the choice is between 8 bit and 16 bit for each sample.

**The BuffersSize**

The BufferSize is needed for the successful creation of an AudioRecord object. The Audiorecord class defines a method which calculate the minimum Buffer Size needed called getMinBufferSize() [1]. By adding the desired sample rate, channel configuration and audio format to the constructor the method calculates the minimum buffer size needed. The actual Buffer size used depends on the expected frequency at which the AudioRecord instance will be polled for new data and should be chosen to be higher than the minimum buffer size calculated by the method getMinBufferSize().

For the beam forming App, we construct 2 objects from the Audiorecord class. The first object containing the main smartphone microphone as the audio source, samples at 8 kHz, mono, 16-bit PCM encoded and a buffer size large enough to capture 1 s of incoming signal as samples. The second object is similar to the first object except now the audio source is the second microphone of smartphone. The second microphone is usually the camera recorder microphone of the smartphone as it is the case with Samsung galaxy s2.

The recorded samples for the 2 mic’s separated are read in a array and pass to the class which performs the beam forming algorithm.
Appendix C.2 the MediaRecorder Class

To construct a new MediaRecorder object the following parameters have to be added to the constructor:

- AudioSource
- OutputFormat
- AudioEncoder
- OutputFile

The Audio Source
In the MediaRecorder Class the Audio Source is also defined by MediaRecorder.AudioSource final Class. As it was the case with the AudioRecord Class the same options can be chosen for the Audio output Hardware:

- CAMCORDER
  (Microphone audio source with same orientation as camera)
- DEFAULT
  (Default Microphone audio source)
- MIC
  (Microphone audio source)
- VOICE_CALL
  (Voice call uplink + downlink audio source)
- VOICE_COMMUNICATION
  (Microphone audio source tuned for voice communication such as VoIP)

- VOICE_DOWNLINK
  (Voice call downlink (Rx) audio source)
- VOICE_RECOGNITION
  (Microphone audio source tuned for voice recognition)
- VOICE_UPLINK
  (Voice call uplink (Tx) audio source)

The Output Format
The output format defines the format in which the output file is produced during recording. The choice is between:

- AMR NB
  The AMR stands for Adaptive Multi-Rate audio codec. The AMR is adopted as the standard speech coding used in GSM and UMTS. The file format is supported by wide range of media
player software’s on the computer and also by Android. The NB stands for narrowband and in general it is optimized for POTS wireline quality of 300-3400 Hz.

- **AMR WB**
The AMR WB is the same as AMR NB only it provides improved speech quality due to a wider speech bandwidth of 50-7000 Hz. It is codified as G.722.2, which is formally known as wideband coding of speech around 16 Kbits/s.

- **DEFAULT**
The option DEFAULT is used when the Android device is limited in capability to save audio or video in one of the other formats explained here. When DEFAULT is chosen the audio or video is captured and saved in a format desirable by the device.

- **MPEG4**
The MPEG 4 stands for compression of audio or visual digital data. It was designed by Moving Picture Experts Group (MPEG) and set as a standard for group of audio and video coding. The MPEG 4 codec is used in streaming media over the web, CD distribution, voice and broadcast television applications.

- **Raw AMR**
The Raw AMR is the raw version of AMR encoding explained above with very small narrow band.

- **THREE GPP**
The THREE GPP stands for .3gp format. 3GP format is a multimedia container format for audio and video on 3G smart phones. It is defined by Third Generation Partnership Project (3GPP) for 3G UMTS multimedia devices.

---

**The Audio Encoder**
The Audio Encoder defines the format in which the captured audio is encoded. The choices are between:

- **AAC**
The Advanced Audio Coding (AAC) is a standardized encoding scheme for digital audio. As the follower of MP3 it achieves an better sound quality at similar bit rate.

- **AMR_NB**
The AMR stands for Adaptive Multi-Rate audio codec. The AMR is adopted as the standard speech coding used in GSM and UMTS. The file format is supported by wide range of media player software’s on the computer and also by Android. The NB stands for narrowband and in general it is optimized for POTS wireline quality of 300-3400 Hz

- **AMR_WB**
The AMR WB is the same as AMR NB only it provides improved speech quality due to a wider speech bandwidth of 50-7000 Hz. It is codified as G.722.2, which is formally known as wideband coding of speech around 16 Kbits/s.
The option DEFAULT is used when the Android device is limited in capability to code audio or video in one of the other formats explained here. When DEFAULT is chosen the audio or video is coded in a format applicable and desirable by the device.

**The Output File**
The Output File object defines the path to which the output file is produced. The path should contain the desired directory on the Android device where the output file has to be saved. By calling the `getExternalStorageDirectory()` method, the available directory for storage on the Android devices is returned as shown in figure 2.2 below.

```java
adres = Environment.getExternalStorageDirectory().getAbsolutePath();
```

*Figure 2.2 The use of getExternalStorageDirectory() method.*

The Android Applications program

...implemented in java are very output path sensitive, if the devices cannot find the path or the file to write to, it will generate an error and the entire application stops working. When using the `getExternalStorageDirectory()` method, the device will search for the appropriate path to write to and no error will be generated.
Appendix D: instruction sets supported by Android NDK

In this appendix the instruction sets supported by Android NDK are explained. The information gathered comes from Android.com and Hosted by FrAndroid. [1]

Introduction: ==============

Every piece of native code generated with the Android NDK matches a given "Application Binary Interface" (ABI) that defines exactly how your application's machine code is expected to interact with the system at runtime.

A typical ABI describes things in *excruciating* details, and will typically include the following information:

- the CPU instruction set that the machine code should use
- the endianness of memory stores and loads at runtime
- the format of executable binaries (shared libraries, programs, etc...) and what type of content is allowed/supported in them.
- various conventions used to pass data between your code and the system (e.g. how registers and/or the stack are used when functions are called, alignment constraints, etc...)
- alignment and size constraints for enum types, structure fields and arrays.
- the list of function symbols available to your machine code at runtime, generally from a very specific selected set of libraries.

This document lists the exact ABIs supported by the Android NDK and the official Android platform releases.

I. Supported ABIs: ===============

Each supported ABI is identified by a unique name.

I.1. 'armeabi'

------

This is the name of an ABI for ARM-based CPUs that support *at* *least* the ARMv5TE instruction set. Please refer to following documentation for more details:

- ARM Architecture Reference manual (a.k.a. ARMARM)
- Procedure Call Standard for the ARM Architecture (a.k.a. AAPCS)
- ELF for the ARM Architecture (a.k.a. ARMELF)
- ABI for the ARM Architecture (a.k.a. BSABI)
- Base Platform ABI for the ARM Architecture (a.k.a. BPABI)
- C Library ABI for the ARM Architecture (a.k.a. CLIABI)
- C++ ABI for the ARM Architecture (a.k.a. CPPABI)
- Runtime ABI for the ARM Architecture (a.k.a. RTABI)

- ELF System V Application Binary Interface (DRAFT - 24 April 2001)


Note that the AAPCS standard defines 'EABI' as a moniker used to specify a _family_ of similar but distinct ABIs. Android follows the little-endian ARM GNU/Linux ABI as documented in the following document:


With the exception that wchar_t is only one byte. This should not matter in practice since wchar_t is simply *not* really supported by the Android platform anyway.

This ABI does *not* support hardware-assisted floating point computations. Instead, all FP operations are performed through software helper functions that come from the compiler's libgcc.a static library.

Thumb (a.k.a. Thumb-1) instructions are supported. Note that the NDK will generate thumb code by default, unless you define LOCAL ARM_MODE in your Android.mk (see docs/ANDROID-MK.html for all details).

I.2. 'armeabi-v7a'
--------------

This is the name of another ARM-based CPU ABI that *extends* 'armeabi' to include a few CPU instruction set extensions as described in the following document:

- ARM Architecture v7-a Reference Manual

The instruction extensions supported by this Android-specific ABI are:

- The Thumb-2 instruction set extension.
- The VFP hardware FPU instructions.

More specifically, VFPv3-D16 is being used, which corresponds to 16 dedicated 64-bit floating point registers provided by the CPU.

Other extensions described by the v7-a ARM like Advanced SIMD (a.k.a. NEON), VFPv3-D32 or ThumbEE are optional to this ABI, which means that developers should check *at* *runtime* whether the extensions are available and provide alternative code paths if this is not the case.

(Just like one typically does on x86 systems to check/use MMX/SSE2/etc... specialized instructions).

You can check docs/CPU-FEATURES.html to see how to perform these runtime
checks, and docs/CPU-ARM-NEON.html to learn about the NDK's support for building NEON-capable machine code too.

**IMPORTANT NOTE:** This ABI enforces that all double values are passed during function calls in 'core' register pairs, instead of dedicated FP ones. However, all internal computations can be performed with the FP registers and will be greatly sped up.

This little constraint, while resulting in a slight decrease of performance, ensures binary compatibility with all existing 'armeabi' binaries.

**IMPORTANT NOTE:** The 'armeabi-v7a' machine code will *not* run on ARMv5 or ARMv6 based devices.

### I.3. 'x86'

This is the name of an ABI for CPUs supporting the instruction set commonly named 'x86' or 'IA-32'. More specifically, this ABI corresponds to the following:

- instructions normally generated by GCC with the following compiler flags:

  ```
  -march=i686 -msse3 -mstackrealign -mfpmath=sse
  ```

  which targets Pentium Pro instruction set, according to the GCC documentation, plus the MMX, SSE, SSE2 and SSE3 instruction set extensions.

- using the standard Linux x86 32-bit calling convention (e.g. section 6, "Register Usage" of the "Calling conventions..." document below), not the SVR4 one.

The ABI does *not* include any other optional IA-32 instruction set extension, including, but not limited to:

- the MOVBE instruction
- the SSSE3 "supplemental SSE3" extension
- any variant of "SSE4"

You can still use these, as long as you use runtime feature probing to enable them, and provide fallbacks for devices that do not support them.

Please refer to the following documents for more details:

- http://gcc.gnu.org/onlinedocs/gcc/i386-and-x86_002d64-Options.html
- Calling conventions for different C++ compilers and operating systems http://www.agner.org/optimize/calling_conventions.pdf
- Intel IA-32 Intel Architecture Software Developer's Manual volume 2: Instruction Set Reference
- Amendment to System V Application Binary Interface
II. Generating code for a specific ABI:
========================================

By default, the NDK will generate machine code for the 'armeabi' ABI. You can however add the following line to your Application.mk to generate ARMv7-a compatible machine code instead:

\[
\text{APP\_ABI := armeabi-v7a}
\]

It is also possible to build machine code for *two* distinct ABIs by using:

\[
\text{APP\_ABI := armeabi armeabi-v7a}
\]

This will instruct the NDK to build two versions of your machine code: one for each ABI listed on this line. Both libraries will be copied to your application project path and will be ultimately packaged into your .apk.

Such a package is called a "fat binary" in Android speak since it contains machine code for more than one CPU architecture. At installation time, the package manager will only unpack the most appropriate machine code for the target device. See below for details.

III. ABI Management on the Android platform:
=============================================

This section provides specific details about how the Android platform manages native code in application packages.

III.1. Native code in Application Packages:
-------------------------------------------

It is expected that shared libraries generated with the NDK are stored in the final application package (.apk) at locations of the form:

\[
\text{lib/\&lt;abi&gt;/lib\&lt;name&gt;.so}
\]

Where \&lt;abi&gt; is one of the ABI names listed in section II above, and \&lt;name&gt; is a name that can be used when loading the shared library from the VM as in:

\[
\text{System.loadLibrary("\&lt;name&gt;");}
\]

Since .apk files are just zip files, you can trivially list their content with a command like:

\[
\text{unzip -l \&lt;apk&gt;};
\]

to verify that the native shared libraries you want are indeed at the proper location. You can also place native shared libraries at other
locations within the .apk, but they will be ignored by the system, or more precisely by the steps described below; you will need to extract/install them manually in your application.

In the case of a "fat" binary, up to three distinct libraries can be placed in the .apk, for example at:

- lib/aromeabi/libfoo.so
- lib/aromeabi-v7a/libfoo.so
- lib/x86/libfoo.so

### III.2. Android Platform ABI support:

The Android system knows at runtime which ABI(s) it supports. More precisely, up to two build-specific system properties are used to indicate:

- the 'primary' ABI for the device, corresponding to the machine code used in the system image itself.
- an optional 'secondary' ABI, corresponding to another ABI that is also supported by the system image.

For example, a typical ARMv5TE-based device would only define the primary ABI as 'armeabi' and not define a secondary one.

On the other hand, a typical ARMv7-based device would define the primary ABI to 'armeabi-v7a' and the secondary one to 'armeabi' since it can run application native binaries generated for both of them.

A typical x86-based device only defines a primary abi named 'x86'.

### III.3. Automatic extraction of native code at install time:

When installing an application, the package manager service will scan the .apk and look for any shared library of the form:

```
lib/\$primary-abi\$/lib\$name\$.so
```

If one is found, then it is copied under $APPDIR/lib/$name/.so, where $APPDIR corresponds to the application's specific data directory.

If none is found, and a secondary ABI is defined, the service will then scan for shared libraries of the form:

```
lib/\$secondary-abi\$/lib\$name\$.so
```

If anything is found, then it is copied under $APPDIR/lib/$name/.so

This mechanism ensures that the best machine code for the target device is automatically extracted from the package at installation.
Appendix E: Terms of Reference

Program Requirements and Specifications

The Smooth Speech application development project is split up in two parts, the Beam Forming algorithm and the Implementation on the Android smartphone. The requirements discussed here are aimed at the implementation of the application. The requirements for the Beam forming algorithm can be found in Thesis “Two Sensor Array Beamforming Algorithm”. [7]

1 Requirements with Regards to the Intended Use of the Product

[1.1] The Android device running the application should be able to perform The Beam Forming Algorithm.

[1.2] The Android device running the application should be able to record with multiple MIC presented at the device.

[1.3] The application should be able to record with multiple audio acquisition hardware at the same time.

[1.4] By use of standard Android libraries, the recorded data should be sampled at 4KHz.

[1.5] The Android device should be able to establish a clock synchronized network with other smartphones for real time voice data manipulation

[1.6] The application should contain a user friendly interface to interact with the user

2 Requirements with Regards to the Design of the System

[2.1] The user interface should graphically display the purpose of the Beam Forming Algorithm

[2.2] The Android Application should be written using the Android SDK in Eclipse. This is based on Java Programming Environment.

[2.3] The Android Application written in Java Language should have limited execution time
3 Requirements with Regards to the Production Process

[3.1] The Android Application should be developed for Android version 2.3.3

[3.2] The Android Application should run on The Samsung Galaxy S2 Smartphone.

[3.3] Android NDK should be considered to implement the Mathematical functions in the Beam Forming algorithm.

[3.4] A MIC Array should replace the shortcomings of Samsung Galaxy S2 device.
Appendix F: source code

Appendix F.1 Java Code

```java
package layout.Test;

import android.app.Activity;
import android.os.Bundle;
import android.os.Environment;
import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.lang.String;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.SeekBar;
import android.widget.SeekBar.OnSeekBarChangeListener;
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.MediaRecorder;
import flanagan.complex.*;

public class Layout2Activity extends Activity implements OnClickListener, OnSeekBarChangeListener {
    /** Called when the activity is first created. */
    Button button1;
    Button button2;
    Button button3;
    Button button4;
    Button button5;
    ImageView Image1;
    TextView TextView1;
    TextView TextView2;
    private SeekBar graden;
    String StringGraden;
    MediaRecorder Mic1;
    MediaRecorder Mic2;
    double[] MicrophoneData1;
    int SampleLength = 10;
    //MicrophoneData1 = new int[SampleLength];
    double[] MicrophoneData2;
    //MicrophoneData2 = new int[SampleLength];
    String FILENAME1 = "/test.txt";
    String path1 = Environment.getExternalStorageDirectory().getAbsolutePath() + FILENAME1;
    String FILENAME2 = "/testII.txt";
}
```
String path2 = Environment.getExternalStorageDirectory().getAbsolutePath() + FILENAME2;
String FILENAME3 = "/beamformed.txt";
String path3 = Environment.getExternalStorageDirectory().getAbsolutePath() + FILENAME3;
final File samples1 = new File("path1");
final File samples2 = new File("path2");
final File inputbeamformed = new File("path3");
int fs = 40000;
int durationSignal = 1;
public double value1;
public double value2;
public double value3;
public double[] inputSamples1 = new double[fs * durationSignal];
public double[] inputSamples2 = new double[fs * durationSignal];
public double[] inputbeamformed = new double[fs * durationSignal];
public short[] phoneBeamformed = new short[fs * durationSignal];
public short[] inputsample1 = new short[fs * durationSignal];
public short[] inputbeamformedshort = new short[fs * durationSignal];
FileReader fileReaderText1;
FileReader fileReaderText2;
FileReader fileReaderbeamformed;
BufferedReader br1;
BufferedReader br2;
BufferedReader br3;
final int Buffer = AudioRecord.getMinBufferSize(8000, AudioFormat.CHANNEL_CONFIGURATION_MONO, AudioFormat.ENCODING_PCM_16BIT) * 250;
AudioTrack play;
MVDR mvdr;
DelayAndSum das;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    //speakers aan zetten
    AudioManager manage = (AudioManager) this.getSystemService(Context.AUDIO_SERVICE);
    manage.setSpeakerphoneOn(true);
    //GUI
    button1 = (Button) findViewById(R.id.button1);
    Image1 = (ImageView) findViewById(R.id.imageView1);
    button2 = (Button) findViewById(R.id.button2);
    button3 = (Button) findViewById(R.id.button3);
    button4 = (Button) findViewById(R.id.button4);
    button5 = (Button) findViewById(R.id.button5);
    graden = (SeekBar) findViewById(R.idSeekBar1);
    graden.setMax(359);
    graden.setProgress(0);
    graden.setOnSeekBarChangeListener(this);
    TextView1 = (TextView) findViewById(R.id.TextView1);
    TextView2 = (TextView) findViewById(R.id.TextView2);
    button1.setOnClickListener((android.view.View.OnClickListener) this);
    button2.setOnClickListener((android.view.View.OnClickListener) this);
    button3.setOnClickListener((android.view.View.OnClickListener) this);
    button4.setOnClickListener((android.view.View.OnClickListener) this);
button5.setOnClickListener((android.view.View.OnClickListener) this);
TextView2.setText(MaakEenString(graden.getProgress()) + " Degrees");
}

public void onClick(View v) {
    if (v.getId() == R.id.button1) {
        try {
            fileReaderText1 = new FileReader(path1);
            fileReaderText2 = new FileReader(path2);
            fileReaderbeamformed = new FileReader(path3);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            System.out.println("kan de file niet lezen !");
        }
        System.out.println("kan de file wel lezen!");
        br1 = new BufferedReader(fileReaderText1);
        br2 = new BufferedReader(fileReaderText2);
        br3 = new BufferedReader(fileReaderbeamformed);
        String input1 = "";
        String input2 = "";
        String input3 = "";
        int i = 0;
    }

    while (input1 != "){
        try {
            inputSamples1[i] = Double.parseDouble(input1);
            inputSamples2[i] = Double.parseDouble(input2);
            inputbeaformed[i] = Double.parseDouble(input3);
            input1 = br1.readLine();
            input2 = br2.readLine();
            input3 = br3.readLine();
        } catch (IOException e) {
            e.printStackTrace();
            break;
        } catch (NullPointerException e) {
            e.printStackTrace();
            break;
        } catch (NumberFormatException e) {
            e.printStackTrace();
            break;
        }
        i++;
    }
}
} 
System.out.println("De lengte van de Samples zijn
geworden:"+inputSamples1.length);

// bij doorgeven van de hoek heb ik in plaats 
vangrades.getProgress(), 0 gezet enkele werkend graden in algorithm
//

mvdr = new MVDR(inputSamples1,inputSamples2,0);
//
System.out.println("heb net een mvdr gemaakt

/////////////////////////////////////////////////////////////////////////////////////");
//
phoneBeamFormed = mvdr.getEstimatedSignal();

// hier word de DAS algorithm performed

das = new DelayAndSum(inputSamples1, inputSamples2, 0);
System.out.println("heb das gemaakt!");
phoneBeamFormed = das.getEstimatedSignal();

//System.out.println("heb de playsample binnen\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\n
for ( int j = 0; j< inputSamples1.length; j++)
{
    inputsample1[j] = ( short)(1000000 *inputSamples1[j]);
    inputbeamformedshort[j] = ( short) (1000000 *inputbeaformed[j] );
    phoneBeamformed[j] = (short) (100000 * phoneBeamFormed[j]);
}
System.out.println("de lengte van de te spelen samples zijn
geworden"+inputsample1.length);

}

if(v.getId() == R.id.button2) {

    // Hier worden de opgenomen samples afgespeeld en vertoond

    play = new
    AudioTrack(AudioManager.STREAM_VOICE_CALL,8000,AudioFormat.CHANNEL_CONFIGURATION_MONO,AudioFormat.ENCODING_PCM_16BIT,Buffer,
        AudioTrack.MODE_STATIC);
    play.write(inputsample1, 0, inputSamples1.length);
    play.setMaxVolume();
    play.play();
    for (int g=0; g<inputSamples1.length; g++)
    {
        //TextView1.setText("The following Samples are recorded:"+inputsample1[g]);
    }
}

if(v.getId() == R.id.button3) {
// Dit is de stop button
play.stop();
play.flush();
play.release();

if(v.getId() == R.id.button4) {
    //TextView1.setText("This is how it sounds when MVDR Beamforming is performed by the Phone! ");
    play = new AudioTrack(AudioManager.STREAM_VOICE_CALL,8000,AudioFormat.CHANNEL_CONFIGURATION_MONO,AudioFormat.ENCODING_PCM_16BIT,Buffer,
        AudioTrack.MODE_STATIC);
    play.write(phoneBeamformed, 0, inputbeamformedshort.length);
    play.setStereoVolume (play.getMaxVolume(),
    play.getMaxVolume());
    play.play();
}

if(v.getId() == R.id.button5) {
    //TextView1.setText("This is how it sounds when MVDR Beamforming is performed by Matlab! ");
    play = new AudioTrack(AudioManager.STREAM_VOICE_CALL,8000,AudioFormat.CHANNEL_CONFIGURATION_MONO,AudioFormat.ENCODING_PCM_16BIT,Buffer,
        AudioTrack.MODE_STATIC);
    play.write(inputbeamformedshort, 0, inputbeamformedshort.length);
    play.setStereoVolume (play.getMaxVolume(),
    play.getMaxVolume());
    play.play();
}

private CharSequence MaakEenString(int progress) {
    return Integer.toString(progress);
}

public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
    TextView2.setText(MaakEenString(gradation.getProgress()) + " Degrees");
    if ((gradation.getProgress() >= 0) && (gradation.getProgress()<= 20 )) {
        Image1.setImageResource(R.drawable.draaiknop0);
    } else if ((gradation.getProgress() >= 21) && (gradation.getProgress()<= 40 )){
        Image1.setImageResource(R.drawable.draaiknop1);
    } else if ((gradation.getProgress() >= 41) && (gradation.getProgress()<= 60 )){

Image1.setImageResource(R.drawable.draaiknop2);
} else if ((graden.getProgress() >= 61) && (graden.getProgress()<= 80
}){
  Image1.setImageResource(R.drawable.draaiknop3);
} else if ((graden.getProgress() >= 81) && (graden.getProgress()<= 100
}){
  Image1.setImageResource(R.drawable.draaiknop4);
} else if ((graden.getProgress() >= 101) && (graden.getProgress()<= 120
}){
  Image1.setImageResource(R.drawable.draaiknop5);
} else if ((graden.getProgress() >= 121) && (graden.getProgress()<= 140
}){
  Image1.setImageResource(R.drawable.draaiknop6);
} else if ((graden.getProgress() >= 141) && (graden.getProgress()<= 160
}){
  Image1.setImageResource(R.drawable.draaiknop7);
} else if ((graden.getProgress() >= 161) && (graden.getProgress()<= 180
}){
  Image1.setImageResource(R.drawable.draaiknop8);
} else if ((graden.getProgress() >= 181) && (graden.getProgress()<= 200
}){
  Image1.setImageResource(R.drawable.draaiknop9);
} else if ((graden.getProgress() >= 201) && (graden.getProgress()<= 220
}){
  Image1.setImageResource(R.drawable.draaiknop10);
} else if ((graden.getProgress() >= 221) && (graden.getProgress()<= 240
}){
  Image1.setImageResource(R.drawable.draaiknop11);
} else if ((graden.getProgress() >= 241) && (graden.getProgress()<= 260
}){
  Image1.setImageResource(R.drawable.draaiknop12);
} else if ((graden.getProgress() >= 261) && (graden.getProgress()<= 280
}){
  Image1.setImageResource(R.drawable.draaiknop13);
} else if ((graden.getProgress() >= 281) && (graden.getProgress()<= 300
}){
  Image1.setImageResource(R.drawable.draaiknop14);
} else if ((graden.getProgress() >= 301) && (graden.getProgress()<= 320
}){
  Image1.setImageResource(R.drawable.draaiknop15);
} else if ((graden.getProgress() >= 321) && (graden.getProgress()<= 340
}){
  Image1.setImageResource(R.drawable.draaiknop16);
}
else if ((graden.getProgress() >= 341) && (graden.getProgress()<= 349)) {
    Image1.setImageResource(R.drawable.draaiknop17);
} else if ((graden.getProgress() >= 350) && (graden.getProgress()<= 359)) {
    Image1.setImageResource(R.drawable.draaiknop0);
}

public void onStartTrackingTouch(SeekBar seekBar) {
    //function not used
}

public void onStopTrackingTouch(SeekBar seekBar) {
    //function not used
}
Appendix F.2 the XML Code

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/TextView1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>

    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/draaiknop0"/>

    <Button
        android:id="@+id/read"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Read"/>

    <Button
        android:id="@+id/show"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Show"/>

    <TextView
        android:id="@+id/TextView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"/>

    <SeekBar
        android:id="@+id/seekBar1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <Button
        android:id="@+id/play"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Play"/>

</LinearLayout>
```
Appendix F.3 the Manifest file

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="Read.File.Play"
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="10" />
    <uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission android:name="android.permission.INSTALL_LOCATION_PROVIDER" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
    <uses-permission android:name="android.permission.REORDER_TASKS" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name">

        <activity
            android:name=".ReadFileandPlayActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```