Master Project

Real-Time Enhancement of E-Larynx Speech using an Android Application

Daniel Emilio Pellicer García

Signal Processing and Speech Communication Laboratory
Faculty of Electrical and Information Engineering
Graz University of Technology, Austria

Advisors:
Martin Hagmüller, Graz University of Technology, Austria
Anna Katharina Fuchs, Graz University of Technology, Austria

Graz, September 2012

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am …………………………… ………………………………………………..
(Unterschrift)

(Englische Fassung für den Fall, dass die Diplomarbeit in Englisch verfasst wird –
es ist nur eine Sprachversion zu verwenden, die andere daher löschen):

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…………………………… ………………………………………………..
date (signature)

Abstract

People who have lost their larynx have three possibilities of talking. These include using Esophageal Speech, Artificial Larynx or Tracheoesophageal Puncture. Of those three the most used is the artificial larynx. This method uses an external device called electrolarynx. The electrolarynx produces a buzz at a specific frequency which the person uses for articulating words. The problem with the electrolarynx is that it produces a flat frequency tone and it generates strong background noise.

This master project aims to improve the characteristics of the sound produced by an electrolarynx speaker adjusting the frequency contour of the electrolarynx depending on the prosodic information of the voice. For this purpose an android application will be created.

This android application would take the sound that comes out of the electrolarynx, apply some processing and then it will generate pulses using the synthesized frequency. These pulses will do the electrolarynx vibrate in the desired frequency producing a more natural fundamental frequency contour.

The main advantages of using an android application are the increase of portability of the algorithm and the fact that you don’t need additional hardware for the signal processing (It’s assumed that everybody has a mobile phone)

The resulting program has resulted to work properly. It works well used in conjunction with an electro-larynx and the sound is subjectively better. The results in the f_0 contour shows that even if the correlation with an original healthy speech was not perfect, the contour generated its similar to the contour of the f_0 of a human speech

Contents

List of figures

Notation

API	Application Programming Interface
DDMS	Dalvik Debug Monitor Server
DREL	Direct Radiated Electronic Larynx (noise)
DSP	Digital Signal Processor
EL	Electronic Larynx
f_0	Fundamental Frequency
F1	First Formant
F2	Second Formant
FFT	Fast Fourier Transform
IDE	Interface Development Environment
JNI	Java Native Interface
LPC	Linear Predictive Code
NDK	Native Development Kit
OS	Operative System
SDK	Software Development Kit
T_0	Fundamental Period

1 Introduction

In this chapter we are going to expose the idea of enhancement of e-larynx speech and its roots. First we will talk about the motivation of the project. Then we will speak about speech prosody and perceived speech. To finish we will talk about the generation of a pitch contour. All this chapter serves as an introduction of what is done later in the android application and why. The main idea of the whole chapter comes from the chapter 6 of Martin Hagmüller Doctoral Thesis. [Hag09]

1.1 Motivation

People who have lost their larynx because of cancer normally choose to use an electro larynx to speak. The electro larynx produces a buzz at a specific flat frequency that the person uses to articulate words. The main problems of the electro larynx approach are that it produces a totally flat frequency contour and introduces in the system a strong background noise.

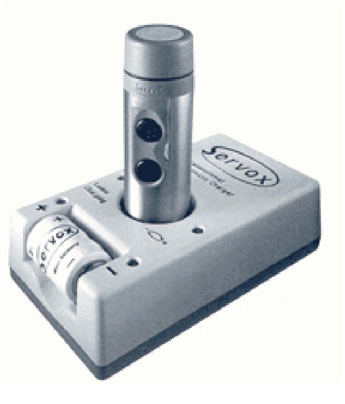


Figure 1.1 Image of an electrolarynx in its charging device. It uses a battery that usually lasts between 1-3 days.

In the figure 1.1 we can see the aspect of an electro larynx. The head of the electrolarynx would be put in the throat of the speaker and the buttons are used to activate the buzzing.

This master project aims to improve the characteristics of the sound produced by an electrolarynx. For this purpose the developing of an application in a mobile device is studied.

1.2 Introduction to Speech Prosody

Prosody in speech describes the relationships of amplitude, duration and fundamental frequency in speech. It helps the communication by introducing hints of how the sentences are build and it helps to improve the exact meaning of the information as well as including implicit data in the communication. The most important features of prosody are intonation, rhythm, rate, accentuation and timbre.

The fundamental feature in this project is the fundamental frequency f_0. It's also the better studied because it's easier to measure than other features. The f_0 movement characterizes for example the declination of a sentence. It's already been studied that flat f_0 contours lower the intelligibility of the speech [Hag09]. Even if the viability of other possibilities of improving the sound of e-larynx speech could be contemplated the easier and more noticeable way to follow would be to synthesize a fundamental frequency contour for improving the prosodic quality of e-larynx speech.

1.3 Perceived Pitch and Speech Spectrum

When using whispered sounds and electro larynx sounds there is some prosodic information hidden in the speech spectrum. In previous studies it has been demonstrated that, when no fundamental frequency is heard, subjects chose the formant frequency F2 as the perceived pitch. Other studies have shown that formants F1 and F2 move depending on the intended pitch.

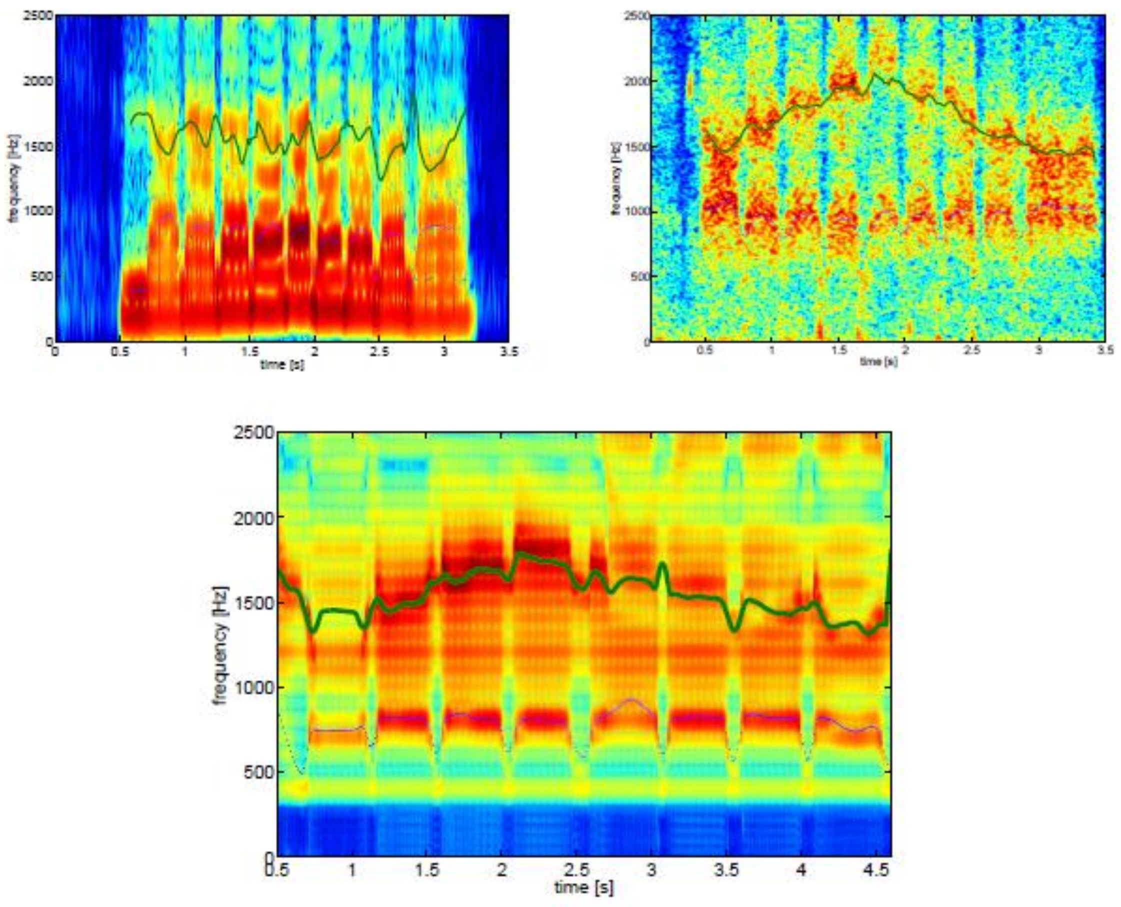


Figure 1.2: Spectrogram of a musical scale sung by a female person, with superimposed tracks of the formants F1 and F2. Top left: Laryngeal. Top right: Whispered. Bottom: Electro-Larynx (from [Hag09])

As it's shown in the figure 1.2 when the alaryngeal singing was analyzed, it showed that with some training the subjects were able to use formants one and two to create some kind of musicality.

1.4 Pitch Contour Generation

The intention is to calculate an artificial pitch contour using the analysis of the speech. Based on the influence of the formants on the perception of prosody, the formants are the chosen parameters to create a pitch contour.

The difference between normal speech production and EL speech is that the voicing source is produced outside the human body. Due to imperfect coupling only a part of the energy is transferred in the neck and interacts with the vocal tract. The other part of the energy goes directly to the listener through the air and this is what we call DREL.

Because the electrolarynx produces DREL sound and this is doing less distinct the formants, first we use a separation method to reduce the noise. This method is called spectral subtraction.

The spectral subtraction method is based on estimating the noise power spectrum and then subtracting this spectrum from the signal power spectrum. The noise is calculated during non-speech intervals. The Spectral subtraction makes the processing of the speech signal easier.

Even after spectral subtraction, formant tracking is not easy. There are still ongoing research questions. In the case we are treating is not a big problem because the formants are important only for speech units that the speaker emphasizes and to convey prosodic information.

For the format tracking a linear predictive code (LPC) is used. This algorithm tries to create a function that matches as good as possible the original input speech signal. Once this function is created the poles and zeroes are calculated. Using the information of the poles and zeroes the formants of the speech can be obtained.

For calculating the f_0 components we use the formants and we calculate f_0 samples block by block. Then these f_0 components are used to create a train of pulses that it's used to generate the output for the excitation of the electro larynx.

The artificial pitch it's only used in case of voiced sounds, so during pauses or unvoiced sounds a default f_0 frequency would be produced. The EL activity is detected just using an energy detector. For unvoiced sounds the decision is made using the frequency components of the speech and their centroid. The threshold values have to be adjusted to be adapted to the concrete circumstances.

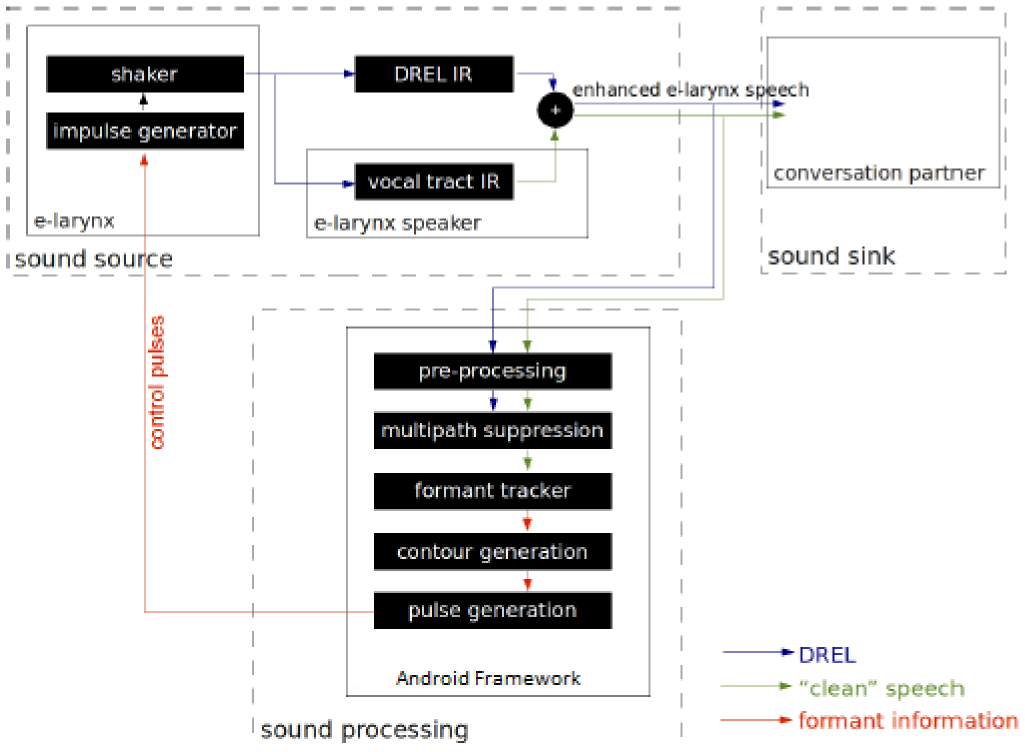


Figure 1.3 Main block diagram of the pitch contour generator. (Modified from [Noist11])

As seen in the figure 1.3, the speech signal is created in the sound source block with the appropriate f_0 driving the shaker. The f_0 calculation and the pulse generation it's done in the sound processing block. The signal with contour works as final output of the system as well as input for the new processing block in order to do the formant tracking and f_0 generation continuously.

These algorithms have already been tried and they work perfectly well for improving the subjective quality of the sound. Tests have been done and the results are that it doesn't improve the understanding of the speech but listeners prefer the sound of the enhanced version. [Hag09]

1.5 Conclusion

In this chapter we have studied the problems of an electronic larynx the principles of formants and speech and the possibility of implementing an algorithm that improves the sound of the e-larynx using a new f0 contour. The EL speech with a new f_0 superposed is preferred to a flat tone EL speech. More information about speech theory could be found in [SPOKEN]

The next step would be to make this algorithm available for everybody in a portable way and easily accessible. In this master project the possibility of implementing it in an android device is the main part of the study. In the next chapter we will talk about the Android system and the implementation of the algorithm in the Android framework.

2 Working with the Android Environment

In this chapter we will talk about the characteristics of developing an application in Android and about the special characteristics of our application, the problems that this generates and the solutions decided. Through the chapter we will talk about all the entities that form the program.

2.1 Android Introduction

Android OS is a new operative system for smartphones that has highly increased its popularity in the last years. Being it open source and easily manageable for everyone, the number of copies sold and number of applications for this OS is also increasing year by year. The fact that is a portable device that could be used for the signal processing part of the application, his grow in market share, and the possibility of using a wireless microphone and speaker for use in conjunction with the electronic larynx make this platform ideal for developing the application. For these reasons we chose to do our app for Android [Android]

Android has also some problems. The latency of these devices is in mean highly superior to the latency in its main competitor the iPhone. This could be a drawback because it affects directly to the performance of real-time applications. In our case the latency measured in Android device is low enough for the application to work correctly.

2.2 Eclipse and the Android SDK

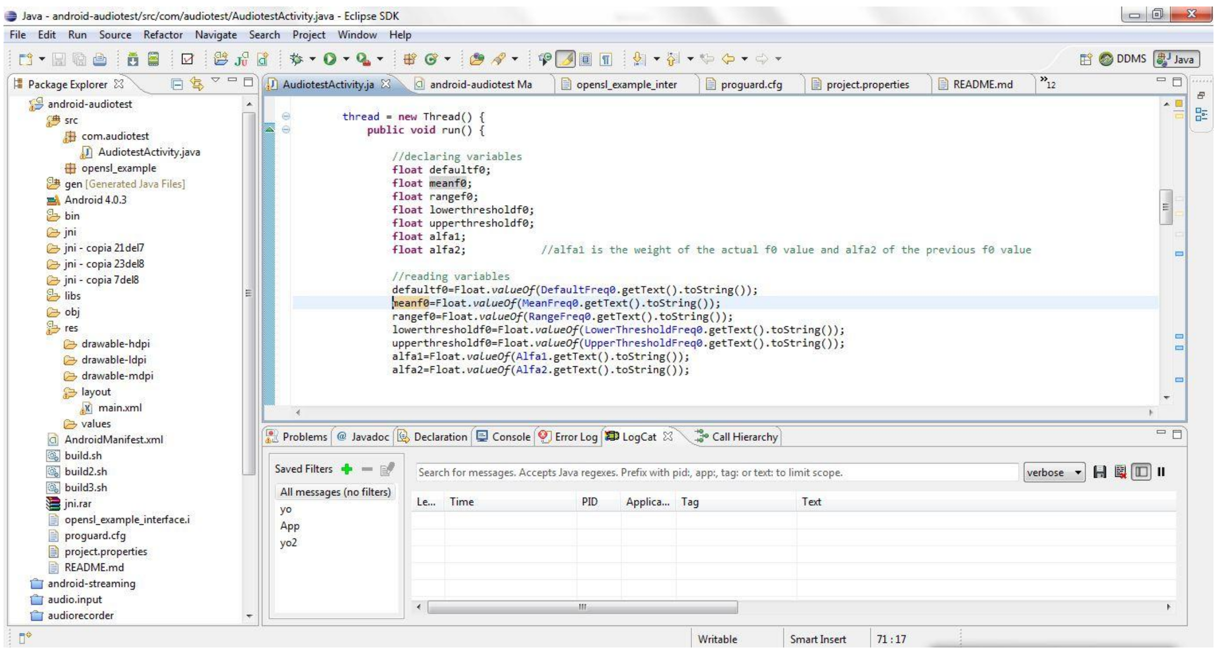
The android SDK is a software development kit that allows developers to make applications for android. The android SDK include sample projects, development tools, an emulator and the required libraries to create Android apps.

The applications are written using Java and run on Dalvik, a custom virtual machine designed for embedded use which runs on top of a Linux kernel.

The officially supported integrated development environment is Eclipse using the android development tools plugin. It's also possible to use any text editor to create Java and XML files and then use command line tools to create the application.

The SDK supports new and old versions of the android platform in case the developer would want to target his application to an older device. Development tools are downloadable contents so different platforms are available. [AndADT]

Eclipse IDE is free and open source software. As other IDEs, Eclipse is a software application that provides easier facilities to programmers for developing software. It has a source code editor, build automation tools and a debugger. It has a compiler and an interpreter (for c or java code). Eclipse is used to maximize the programmer productivity by providing components with similar user interfaces. [Eclipse]



```
thread = new Thread() {
    public void run() {

        //declaring variables
        float defaultf0;
        float meanf0;
        float rangef0;
        float lowerthresholdf0;
        float upperthresholdf0;
        float alfa1;
        float alfa2;            //alfa1 is the weight of the actual f0 value and alfa2 of the previous f0 value

        //reading variables
        defaultf0=Float.valueOf(DefaultFreq0.getText().toString());
        meanf0=Float.valueOf(MeanFreq0.getText().toString());
        rangef0=Float.valueOf(RangeFreq0.getText().toString());
        lowerthresholdf0=Float.valueOf(LowerThresholdFreq0.getText().toString());
        upperthresholdf0=Float.valueOf(UpperThresholdFreq0.getText().toString());
        alfa1=Float.valueOf(Alfa1.getText().toString());
        alfa2=Float.valueOf(Alfa2.getText().toString());
```

android-audiotest
src
com.audiotest
AudiotestActivity.java
opensl_example
gen [Generated Java Files]
Android 4.0.3
bin
jni
jni - copia 21del7
jni - copia 23del8
jni - copia 7del8
libs
obj
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
layout
main.xml
values
AndroidManifest.xml
build.sh
build2.sh
build3.sh
jni.rar
opensl_example_interface.i
proguard.cfg
project.properties
README.md
android-streaming
audio.input
audiorecorder

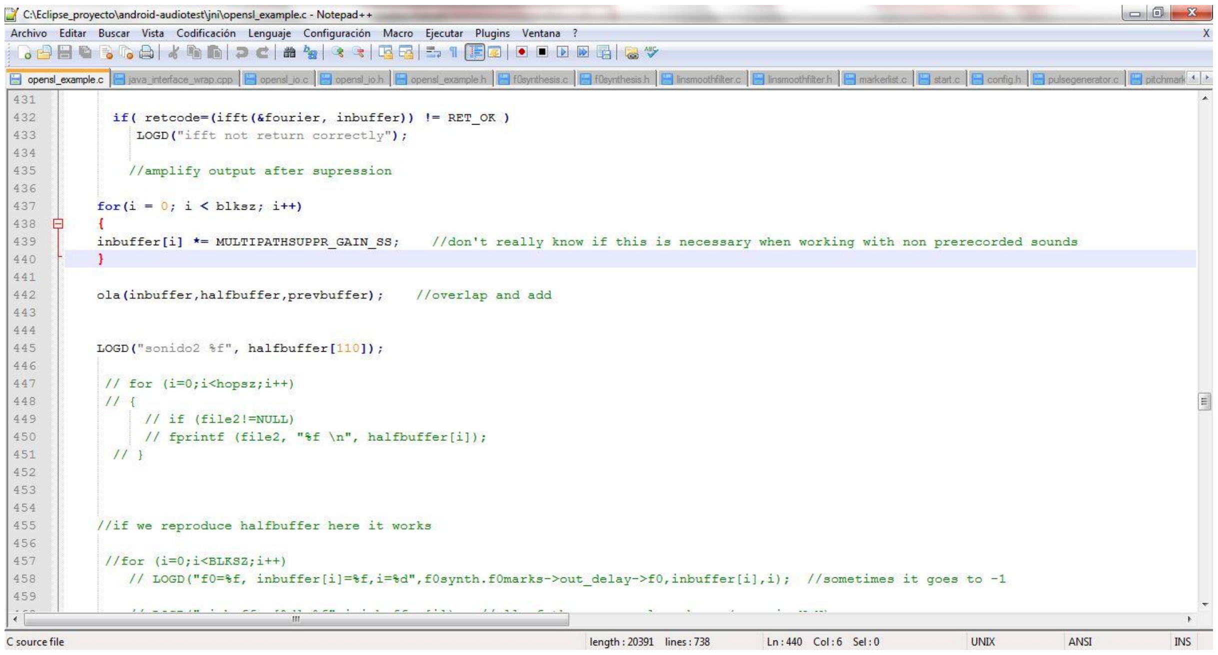
Saved Filters
All messages (no filters)
yo
App
yo2

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
verbose

Le...	Time	PID	Applica...	Tag	Text

Figure 2.1 Eclipse IDE. In the left part of the image there is the Package Explorer. We can observe there the different directories of the project. The more important folders are the src which contains the main java files, de jni which contains all the native program files, the res/layout that contains the graphical interface and the AndroidManifest.xml file that contains the user permissions. In the middle of the screen we can see the main window in which the programing is done. In the lower part there are different tabs. The most important ones are the LogCat tab and the Console Tab. In the console tab we can watch the flux of the program. In the LogCat tab we can see all the messages that the Emulator generates.

All the development could be done in eclipse (including the c code) but another text editor was preferred to program the C code to separate the two parts of the project in an easier way for the programmer.



```
        if( retcode=(ifft(&fourier, inbuffer)) != RET_OK )
           LOGD("ifft not return correctly");

          //amplify output after supression

      for(i = 0; i < blksz; i++)
      {
      inbuffer[i] *= MULTIPATHSUPPR_GAIN_SS;    //don't really know if this is necessary when working with non prerecorded sounds
      }

      ola(inbuffer,halfbuffer,prevbuffer);    //overlap and add


      LOGD("sonido2 %f", halfbuffer[110]);

       // for (i=0;i<hopsz;i++)
       // {
            // if (file2!=NULL)
            // fprintf (file2, "%f \n", halfbuffer[i]);
        // }



      //if we reproduce halfbuffer here it works

       //for (i=0;i<BLKSZ;i++)
          // LOGD("f0=%f, inbuffer[i]=%f,i=%d",f0synth.f0marks->out_delay->f0,inbuffer[i],i);  //sometimes it goes to -1
```

Figure 2.2 Notepad++.

A normal text editor with extended capabilities was used to program the C code. This program called Notepad++ (see fig2.2) allows the user to use some capabilities of normal IDEs but it's a free and open source program. [Notepad++]

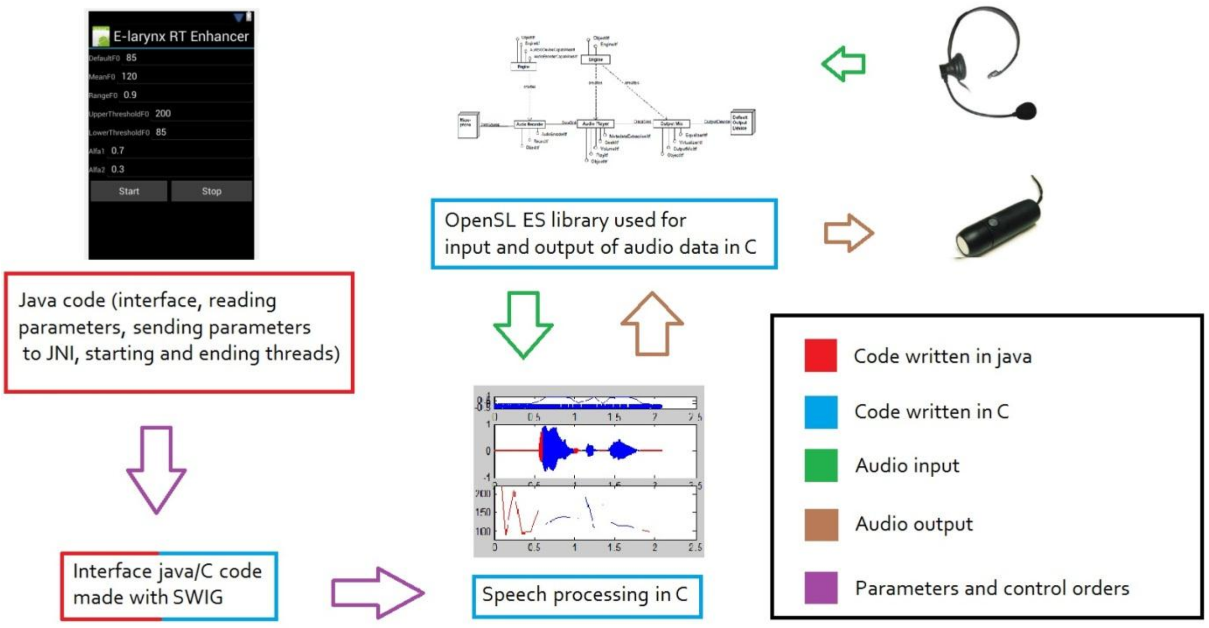
2.3 Signal processing applications in Android

At first it was thought to implement the program using only java, but we discovered using some testing with the emulator that using java functions for audio management was going to affect the performance of the program. With that in mind other possibilities were studied.

The Android SDK is comprehensive and capable, but there may be times when an application requires something more. As shown in [AndAct] chapters 13 and 19, other possibilities apart from java are implementable in Android. The main alternatives are using an application made completely using C or use a basic java code to wrap a more complex application that uses native methods of Android. When programming applications using pure C it doesn't produce applications that are easily executed on consumer hardware. This design approach requires an unlocked developer, or rooted, device and is arguably only applicable for developers who are building custom Android builds—it's not for the typical developer looking to deploy applications to consumer-based handsets. The "approved" manner of writing C code for the Android platform is with assistance from the Android Native Developer Kit, or simply the NDK.

The NDK will not benefit most apps but good candidates for the NDK are CPU-intensive operations such as signal processing, physics simulation, and so on. This program requires intensive CPU operations as everything is about signal processing so is ideal for using the NDK for it.

A general view of the program is shown in fig 2.3. Each part of the algorithm will be explained in the next subchapters.


E-larynx RT Enhancer
Start
Stop
OpenSL ES library used for input and output of audio data in C
Java code (interface, reading parameters, sending parameters to JNI, starting and ending threads)
Interface java/C code made with SWIG
Speech processing in C
🟥 Code written in java
🟦 Code written in C
🟩 Audio input
🟫 Audio output
🟪 Parameters and control orders

DefaultF0 85
MeanF0 120
RangeF0 0.9
UpperThresholdF0 200
LowerThresholdF0 85
Alfa1 0.7
Alfa2 0.3

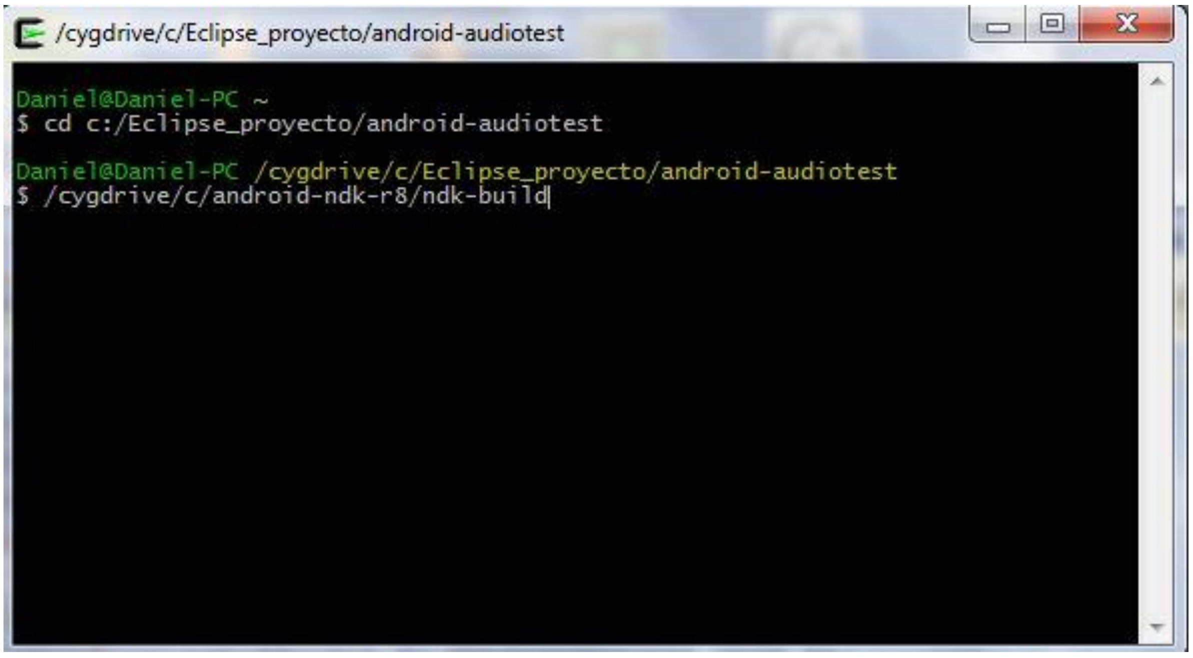
Figure 2.3 Software Block Diagram

2.4 The Native Development Kit (NDK)

The Android NDK is a set of tools that allows Android application developers to embed native machine code compiled from C and/or C++ source files into their application packages. The android NDK allows an android application source code to call methods implemented in native code through the JNI. This could be helpful because it would be possible to reuse existing code libraries written in these languages and possibly increase the performance. [AndNDK]

To use these methods you have to declare them in java and then add a native shared library that contains the implementation of these methods.

The android NDK is a complement to the normal android development kit to allow the programmer to generate native ARM binaries (ARM is a family of microprocessors). A Linux environment is required (The program called Cygwin is an option to do it in windows).



```
Daniel@Daniel-PC ~
$ cd c:/Eclipse_proyecto/android-audiotest

Daniel@Daniel-PC /cygdrive/c/Eclipse_proyecto/android-audiotest
$ /cygdrive/c/android-ndk-r8/ndk-build
```

Figure 2.4 Image of Cygwin. This is a linux-like environment for windows. It allows windows user to compile the c files using the NDK.

Even if it’s a good way to create applications, the NDK should be used in conjunction with the java programming language, to handle Android system events appropriately. However it’s possible to write a sophisticated application in native code with a small application part written in java used to start/stop it appropriately. This was the solution implemented in our code. The C Code was in its biggest part already programed and it was only necessary to adapt it to the android system so the best way to do it was doing little changes in every part of the C code and using java to wrap the application.

For using the NDK a good understanding of the JNI is recommended because many operations in the environment require specific actions from the programmers and sometimes they are not common knowledge.

To include the native methods in the java project you need to use an interface between Java and C. This interface consist on declaring the library in the java code and declaring the native methods used in java and in C as well. The way to declare the methods in the JNI is not straight forward as it needs to use a specific notation. At first it was thought that the best way to do it was using SWIG. SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. [SWIG]

When the project advanced more we realized that actually the notation in the JNI interface is not that difficult and the use of SWIG was not really necessary.

To conclude and as shown in fig 2.3, to compile the files using NDK you need to be in the root folder of your project (Where your application.mk is) and execute the ndk-build program. It will compile the files and put them together in a .so file. This file is a packaged library for the main java program to use. After that the developer only needs to build the normal java project and the library will include the native methods as well.

The NDK provides a set of system headers for stable native APIs that includes the Open SL ES native audio library. This is the library that is going to manage all the input and output of audio in our system. Using this library is the standard approach to manage audio using NDK

2.5 The OpenSL ES Library

2.5.1 Overview

The OpenSL ES API has an object-oriented approach. It uses two concepts, objects and interfaces. An object is an abstraction of a number of resources that are assigned for a specific set of tasks. And object has a type and this type sets the tasks that an object can do. It works similar to a class in C++. An interface is an abstraction of a number of features that an object provides. An interface has a set of methods or functions and a type. An interface can be defined as a combination of type and object to which is linked.

One object can have many interfaces but an interface only works for exactly one object. The application controls the object using interfaces.

An object has three states: unrealized, realized and suspended. Depending on the resources allocated and if after realized it loses resources the object will be in a different state.

Normally for accessing the properties of an object getters and setters are used.

As it’s possible to infer from above, an object has no actual representation in the code. The library refers to an object via its SLOBjectItf interface. Each interface is a C structure where all the fields are method-pointers. This interface structure are always managed using reference and never by value.

Every object has the SLObjectItf interface and this interface is used for all the basic operations on the object and for getting new interfaces for the object.

Other important entities in the architecture are the engine object and the SLEngineItf Interface. These are the entry-point of the API. The applications start the OpenSL ES session creating an engine object. After creating the engine object is possible to obtain the SLEngineItf of this object. With this interface is possible to create all the other object types in the API.

It’s important to know that the exact amount of resources available on an Open SL ES implementation may vary. As a consequence, when using the library an app should be always prepared to handle failure in object realization or dynamic interface additions.

2.5.2 Important Objects in the Architecture

The different kinds of objects that exist in the architecture are: Engine object, media objects, metadata extractor objects, audio output mix objects.

Engine objects have been already mentioned so we will proceed with the others.

Media objects implement usually players and recorders. They operate on audio data. A media object is defined by the operation it performs, the inputs it draws data from and the outputs it sends data to. With data sources and sink structures it's possible to indicate the origin and destiny of the data used by the media object. A data source at the same time is defined by the data locator and the data format.

Metadata Extractor object reads the metadata without allocating resources for playback. It only needs a source and not a sink

Audio output mix object allows routing the audio to different audio outputs.

2.5.3 Objects and Interfaces used

An audio player is used for sampled audio playback. It's possible to use the library using file-based and in-memory data sources, as well as buffer queues. The API supports data encoded in many formats.

An audio recorder is used for capturing audio data. It's not always possible to record in every device.

Pan control, advanced 3D effects, reverb etc. The different effects are introduced using interface exposed on an object. Important kinds of effects are: bass boost, equalization, virtualization and reverberation.

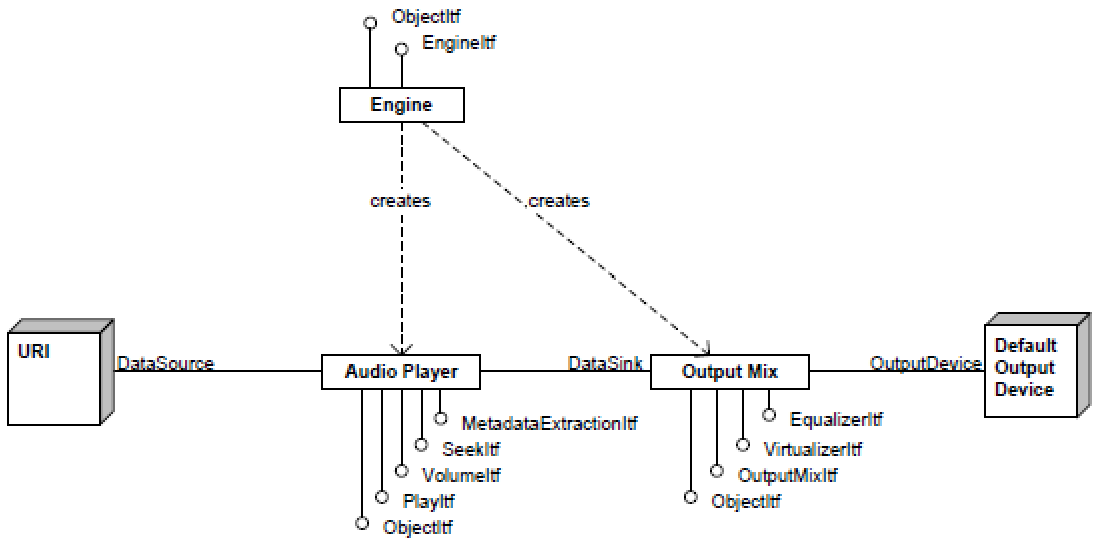


Figure 2.5 Audio player architecture using the OpenSL ES library. (from [OpenSLES])

In the fig 2.4 an example of audio player architecture could be contemplated. Here an audio player is created using the interface of the engine object. When you create the audio player you associate it with an Output Mix. The Output Mix is also created with the engine object. The data source is also set during the audio player creation and the output mix is by default associated to the default output device.

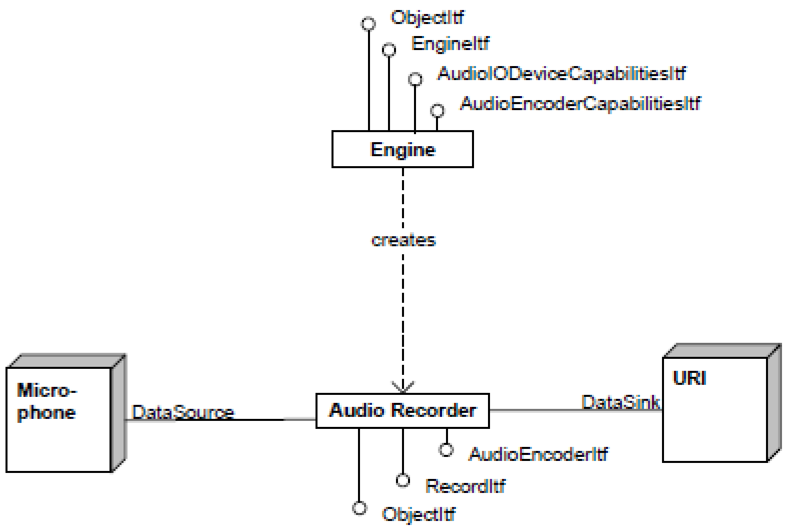


Figure 2.6 Audio recorder architecture using the OpenSL ES library. (from [OpenSLES])

In the fig 2.5 an example of audio recorder architecture is shown. The audio recorder is created using the engine object. When creating it we associate it to a data source (in this case a microphone) and the data sink could be a URI pointing to an audio file or it could be recorded in a buffer for later processing.

In our project these two architectures are connected in the middle by the main processing of the audio.

If more information is needed, the manual of OpenSL ES library could be consulted [OpenSLES]

2.6 Porting the C Code to the NDK

If there is already a code that could be used in the NDK it can be easily ported following a few steps.

First it's necessary to include all the files in the application.mk file of the android project. However it's also possible to include the main one and use the directive #include to add all the other files. The JNI accepts C or C++ files. In this case all the files were written previously for a DSP device so they all were in C. [Noist11]

Because all the files were written for the DSP there were a few functions that had to be added and others had to be modified. The functions that had to be added were the ones that were specifically designed for the DSP and the functions that had to be modified were the functions that required new parameters.

Also in the original problem, the overlap and add function was implemented in an interruption function and here it’s in the same line of flow as the main program.

When the code in C is ready and is time to compile a Linux environment is necessary. A possibility in Windows would be to use Cygwin. In this case the developer has to download all the components of Cygwin necessary to compile a file in C and then call the NDK application through the Cygwin shell. Because it’s compiling using a Linux environment, all the #include entries have to be done using ‘/’ instead of ‘\’. This is necessary to change if you are using files that were compiled with windows.

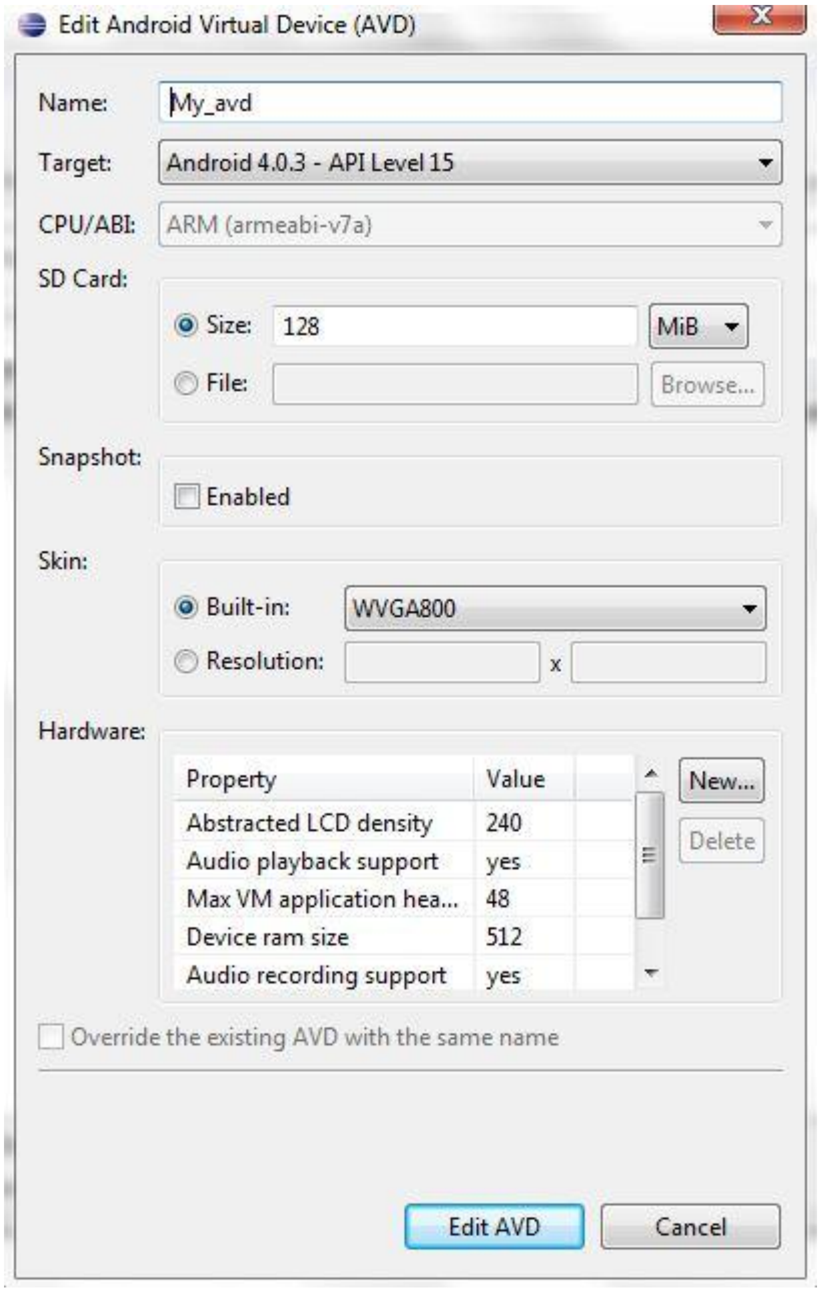
For debugging the code there are also two ways. The first one is to use the debugger that is included in the NDK and use it in conjunction with Cygwin. For people who don’t have a high understanding of Linux maybe is not a trivial work. The second way would be to use the LogCat of Android and Eclipse. Only including a new line in the c code you could use a function to write text and variables from the emulator to the eclipse IDE. In this way you could know the value of the variables in specific moments of time and then correct the problems.

If everything is done correctly the NDK will package together the functions in java and the native functions in C into a .apk file. This file then can be used to install the program in a smartphone.

2.7 The Android Emulator

The android emulator included in the SDK of android is not perfect. The performance of it is worse than the performance in a mobile phone and it only accepts a concrete set of parameters. In this case the emulator was going to be used to check the recording and the playback in real time, the latency and to check if the program was working properly.

The tools that are going to be necessary for the emulator have to be decided before creating it. In this case we have to add Audio playback support and audio record support in the hardware properties of the Emulator in the moment of its creation. In our case it was important to do some tests using files. We needed to open files and use them in the program and also to record the output samples in a file. If this is necessary the emulator has to include an SD card.



Edit Android Virtual Device (AVD)

Name:	My_avd
Target:	Android 4.0.3 - API Level 15
CPU/ABI:	ARM (armeabi-v7a)
SD Card:	Size: 128 MiB [✓] / File:
Snapshot:	Enabled [ ]
Skin:	Built-in: WVGA800 [✓] / Resolution: x

Hardware:

Property	Value
Abstracted LCD density	240
Audio playback support	yes
Max VM application hea...	48
Device ram size	512
Audio recording support	yes

Figure 2.7 Creating a emulator in eclipse. Using this tool the developer can select the android version, capacity for a SD card, and additional hardware properties. The most important ones for this project are the audio playback support and the audio recording support

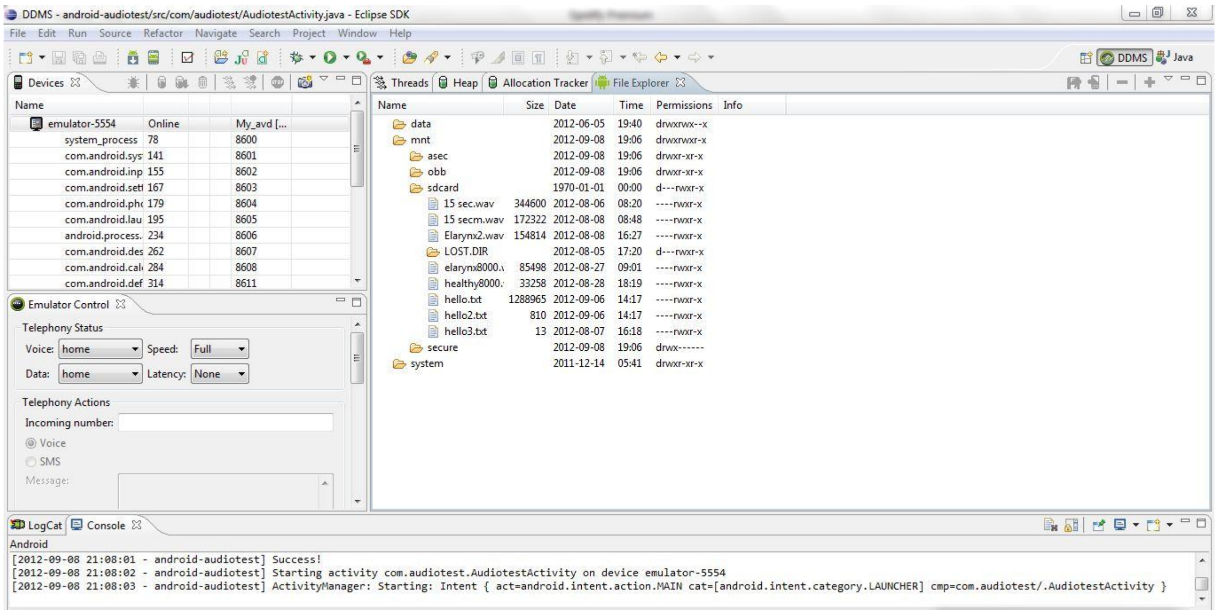
After the creation of the emulator, the developer can try the applications directly in the computer without using a mobile phone. This allows him to try the application in different versions of android etc.

The developer needs to have in mind that some of the configuration options are not possible in the emulator. For example in our case the only sampling rate that works in the emulator is 8000 Hz.

When testing the program we realized that the latency was much higher than the latency in a mobile phone. Subjectively could be said that at least 3 times more latency.

It was necessary then to check the written files to look for missing samples. None of the samples were lost so for this reason some of the tests were done better recording the output in a file and then comparing results in Matlab.

To push and pull files from the android environment and then use them in Matlab, the DDMS window of Eclipse is used. (See fig 2.7)


DDMS - android-audiotest/src/com/audiotest/AudiotestActivity.java - Eclipse SDK
File Edit Run Source Refactor Navigate Search Project Window Help
DDMS
Java
Devices
Emulator Control
Threads
Heap
Allocation Tracker
File Explorer
LogCat
Console

Name				
emulator-5554	Online			My_avd [...
system_process	78			8600
com.android.sys	141			8601
com.android.inp	155			8602
com.android.sett	167			8603
com.android.pho	179			8604
com.android.lau	195			8605
android.process.	234			8606
com.android.des	262			8607
com.android.cal	284			8608
com.android.def	314			8611

Telephony Status
Voice: home Speed: Full
Data: home Latency: None

Telephony Actions
Incoming number:
(● Voice) SMS
Message:

Name	Size	Date	Time	Permissions	Info
data		2012-06-05	19:40	drwxrwx--x	
mnt		2012-09-08	19:06	drwxrwxr-x	
asec		2012-09-08	19:06	drwxr-xr-x	
obb		2012-09-08	19:06	drwxr-xr-x	
sdcard		1970-01-01	00:00	d---rwxr-x	
15 sec.wav	344600	2012-08-06	08:20	----rwxr-x	
15 secm.wav	172322	2012-08-08	08:48	----rwxr-x	
Elarynx2.wav	154814	2012-08-08	16:27	----rwxr-x	
LOST.DIR		2012-08-05	17:20	d---rwxr-x	
elarynx8000.\	85498	2012-08-27	09:01	----rwxr-x	
healthy8000.:	33258	2012-08-28	18:19	----rwxr-x	
hello.txt	1288965	2012-09-06	14:17	----rwxr-x	
hello2.txt	810	2012-09-06	14:17	----rwxr-x	
hello3.txt	13	2012-08-07	16:18	----rwxr-x	
secure		2012-09-08	19:06	drwx------	
system		2011-12-14	05:41	drwxr-xr-x	

```
Android
[2012-09-08 21:08:01 - android-audiotest] Success!
[2012-09-08 21:08:02 - android-audiotest] Starting activity com.audiotest.AudiotestActivity on device emulator-5554
[2012-09-08 21:08:03 - android-audiotest] ActivityManager: Starting: Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER] cmp=com.audiotest/.AudiotestActivity }
```

Figure 2.8 DDMS window in Eclipse. It's used for pushing and pulling files from the device. In our project it was used to introduce audio files to be read by the program and do some testings, and also to extract files that were created by the program as an output.

2.8 The Java and XML Code

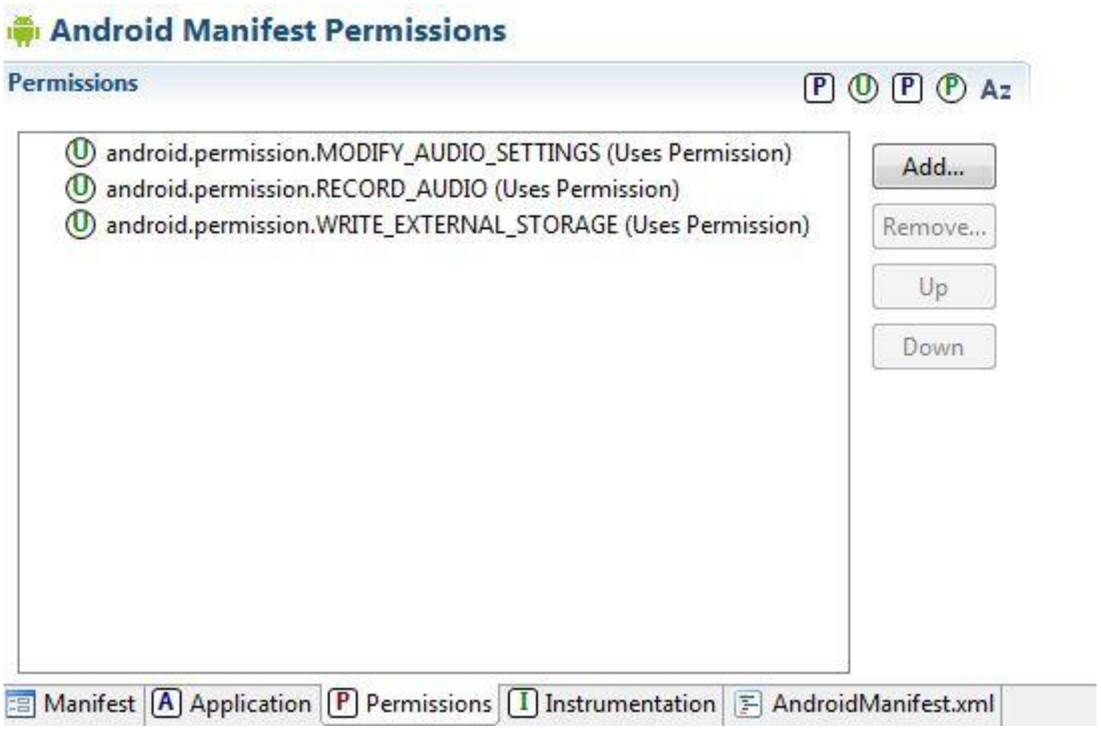
Once the program is done, it needs an interface to communicate with the user and it needs to add the correct user permissions.

2.8.1 User permissions

Each application is separated by Linux from each other and from the system. Additional finer-grained security features are provided using these user permissions. The user permissions is a mechanism that enforces restrictions on the specific operations that a particular process can perform.

A central design point of the Android security architecture is that no application, by default, has permission to perform any operations that would impact in other applications, the operating system, or the user. Because Android sandboxes applications from each other, this applications have to declare the permissions they require and the android system then asks if the user consents the installation of the concrete application with those concrete permissions. Android doesn't have a mechanism to grant permissions dynamically.

The permissions that are necessary in our program are *MODIFY_AUDIO_SETTINGS*, *RECORD_AUDIO* and *WRITE_EXTERNAL_STORAGE*. (showed in fig 2.8)


Android Manifest Permissions

Permissions

android.permission.MODIFY_AUDIO_SETTINGS (Uses Permission)
android.permission.RECORD_AUDIO (Uses Permission)
android.permission.WRITE_EXTERNAL_STORAGE (Uses Permission)

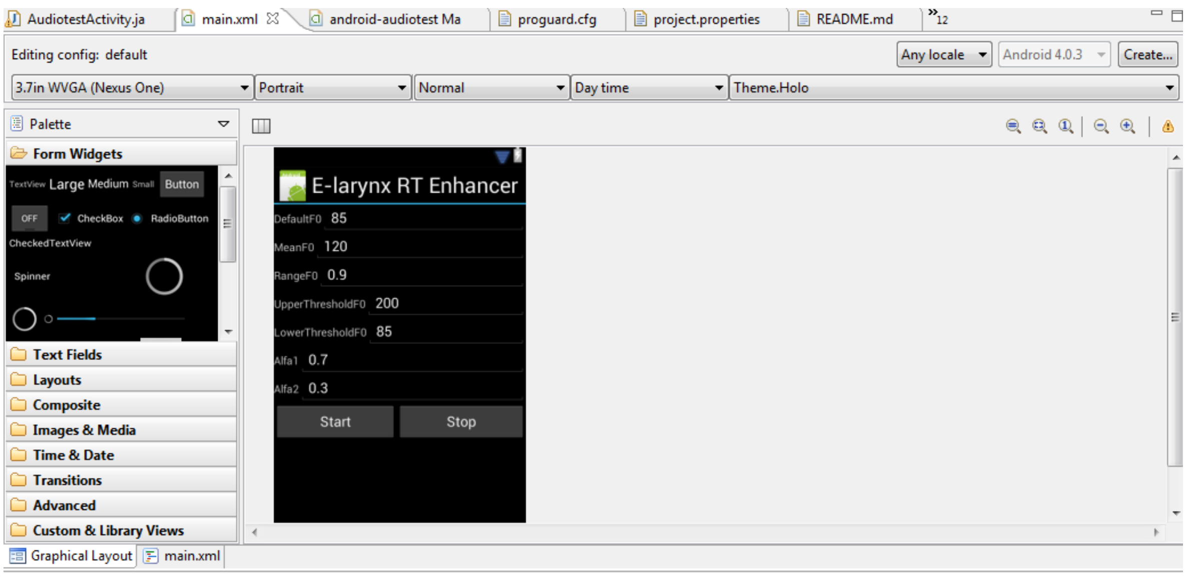
Figure 2.9 Android permissions. Opening the AndroidManifest.xml file you can chose between different tabs. When chosing the androidmanifest.xml tab you can introduce the permissions manually or chose the permissions tab and add them using the interface.

Modify audio settings allow the program to change the normal settings of android to adapt it to its requirements. Record audio allows the program to take audio samples from the microphone of the device (or other audio source) Write external storage allows the program to write data into files of the sd card. These permissions are written in the android_manifest.xml file and will be asked to the user in the installation time

2.8.2 Interface, Input Parameters and Java Code

The interface to communicate with the users is done normally in java or in xml. In this case it’s been made in xml. It’s possible to create a layout graphically with eclipse and then adjust each part of it in detail.

In this case in the interface we are using edit text boxes, buttons and text views. (Interface shown in fig 2.9) The edit text boxes are for writing the input parameters (they have some default values written when starting the program), the buttons are used for starting and stopping the program and the text views show the user the meaning of each value.



Editing config: default

3.7in WVGA (Nexus One) | Portrait | Normal | Day time | Theme.Holo

Any locale | Android 4.0.3

E-larynx RT Enhancer

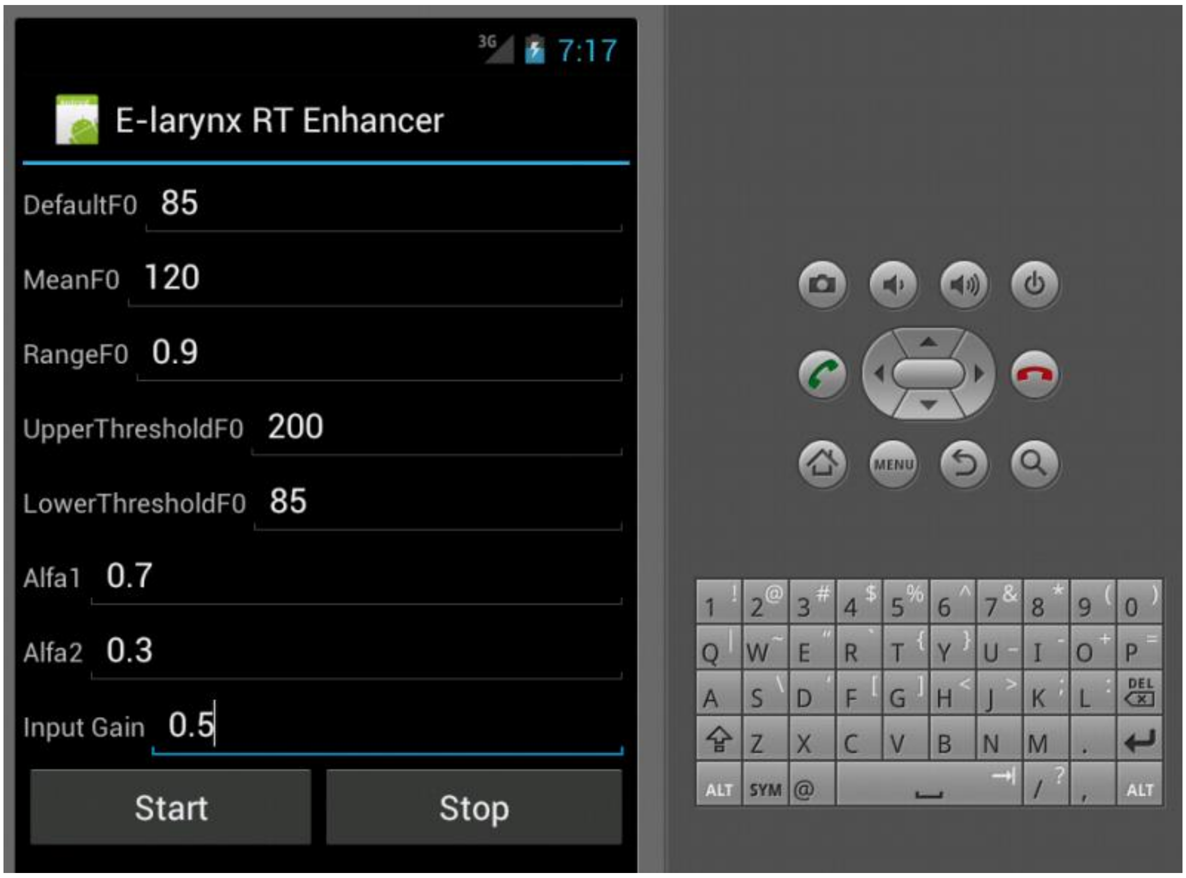
DefaultF0 85
MeanF0 120
RangeF0 0.9
UpperThresholdF0 200
LowerThresholdF0 85
Alfa1 0.7
Alfa2 0.3

Figure 2.10 Main.xml graphic interface window. For creating the graphic interface there are two possibilities. You could do it all in the main.xml tab of the main.xml file or create it first in the graphical layout interface and then do the fine tuning in the xml view.

Once the xml is created it needs to be loaded in the main java code.

The main java code in our case is very simple. It’s used basically to wrap the native code, show the interface to the user and send the parameters to the Native methods.

For this purpose it first loads the main.xml file which contains the graphical interface, then it reads the variables and it prepares the main thread that is going to execute the native method. After that it idles. When the user pushes start the main thread starts to run and the java code don't do anything else until the user press stop. In this case this will stop the thread and free the memory. When pushing start again, the new parameters will be loaded and the main thread will be created again. (fig 2.10)



E-larynx RT Enhancer

DefaultF0 85

MeanF0 120

RangeF0 0.9

UpperThresholdF0 200

LowerThresholdF0 85

Alfa1 0.7

Alfa2 0.3

Input Gain 0.5

Figure 2.11 Android Emulator. This is the interface of the program as it shows when running it in the emulator. The emulator allows the user to do all the functions of a normal phone but has limits in performance and resources. Here we see the different configurable parameters of the program and the buttons for starting and stopping the algoritm

2.9 Conclusion

In this chapter we have explored the possibility of implementing a f_0 contour generator application in Android. We have explored the problems and solutions and we have explained the different parts that were used to create the application.

In the next chapter the tests on a real device and different plot comparisons would be showed to understand the capabilities of the application.

3 Testing

In the last chapter we talked about the characteristics of the different parts of the Android framework that were necessary to implement the f_0 contour generator application.

In this chapter we will show the differences between working with the emulator and a real device and we will analyze the performance of the application and the results obtained with it.

3.1 Test in Real Device

When the application is finished or nearly finished is time to test it in the device. The first thing we realized in the tests was that the input gain was too high and it was getting the noise of himself and therefore changing the frequency randomly. This was solved using headphones or adjusting the input gain.

The next thing was when using an electro larynx. In the spectral subtraction every periodic noise would be lowered and “musical noise” would be added. When using the electro larynx at first we realized that it was also changing the frequency randomly and this was due to the fact that the musical noise was too big in amplitude. The solution was to separate progressively the electro larynx from the mobile phone or also adjusting the input gain.

3.2 Results and Comparisons

When first testing the program we realized that the smoothing that was already done was not smoothing the signal enough. For this reason we decided to do some smoothing also after the generation of each f_0 in each block. This additional smoothing only was to do a weighted average between the actual value and the previous value. The weight of each value can be decided by the user interface. By default 0.7 and 0.3 is now chosen and it seems it works correctly.

Also we decided to be able to modify other parameters that affect the f_0 generation to test them and try to adjust them as better as possible.

The values that are involved in the f_0 generation are the F2 formant, the mean of the f_0, the range amplitude from one extreme to the other, the upper and lower thresholds for the f_0 and the default f_0.

When measuring the F2 formant (see plot in fig 3.1) we realized that actually it follows quite accurately the F2 formant of the signal (measured with other more precise methods) so the problem for the f_0 was not there.

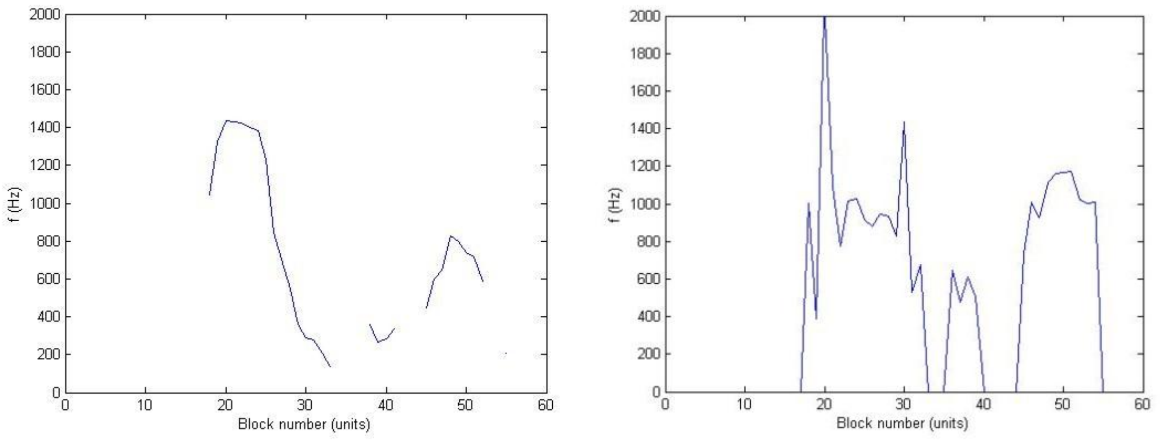


Figure 3.1 F2 shape of a segment of healthy speech. In the left we see the result obtained using a formant tracking function in matlab and in the right the formant obtained in the android program. Even if the correlation is not perfect, there is a big similarity between the two plots.

It was necessary to increase the f_0 mean and the f_0 range as how it was originally to make a nicer contour. The results are all objective because even if the f_0 is correlated with the original one, it's not possible with this method to create a totally reproduction of the original f_0. Only a f_0 contour that resembles the one of a human being. (showed in fig 3.2)

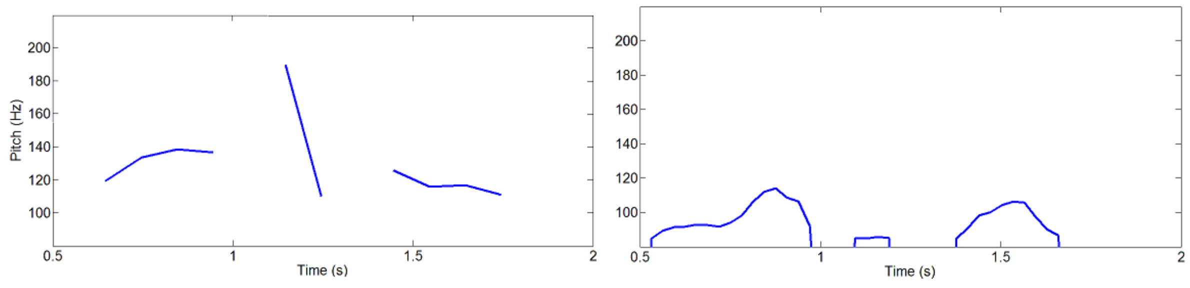


Figure 3.2 f_0 contour. In the left the f_0 contour calculated using a f_0 pitch tracker algorithm. In the right the f_0 generated using the formants of the original speech sample.

3.3 Conclusion and Future Developments

Right now the output of the program is just a succession of deltas separated by a distance T_0 that is calculated in the f_0 generation function. This is used for the excitation of the e-larynx and then the f_0 will change in real time.

A possible future development would be to be able to choose between different pulses in the output. In this way maybe it’s possible to improve the sound of the e-larynx.

As we have seen the algorithm creates an f_0 contour that simulates a contour of a human being. However the algorithm is not perfect and better possibilities have been already explored. For the purpose of this project it’s enough but in the future it could be improved using other techniques as for example the GMM approach [Reyn08]

4 Conclusions

There are a lot of people in the world who use E-larynx and that could benefit of this program. In this master project we have done a little intro in prosodic information in the voice and e-larynx problems. Then we have explained that other programs have been already implemented that works but we wanted to make something portable and more easily accessible for the people. The advantage of using an android device is that we assume that everybody have a mobile phone nowadays so the people who want to use this device doesn't have to carry with him another additional device.

After that we have commented the characteristics of an android project and the parts and necessities of this project. We have discussed the possibility of making a pure java application and we have seen that it was better to work with the Java native interface. Even if using the JNI is more difficult than just programming in java it is recommended for applications with high processing needs as digital signal processing applications. It fits perfectly in our work. Using the JNI it was possible to adapt a C code that was already working just changing some parts of the code.

After having adapted the code we have discussed the testing done and the adjusting of the parameters. We have also commented the results. Even if the results are not adapted totally to the original f0 the result is similar to a human f0 contour and the latency is ok for the application.

Using this application as a start it could be possible to improve it and launch it to the android market for the people to be able to use it in the future and take advantage of its portability and commodity.

A Appendix: Code Explanation

A.1 Software Block Diagram

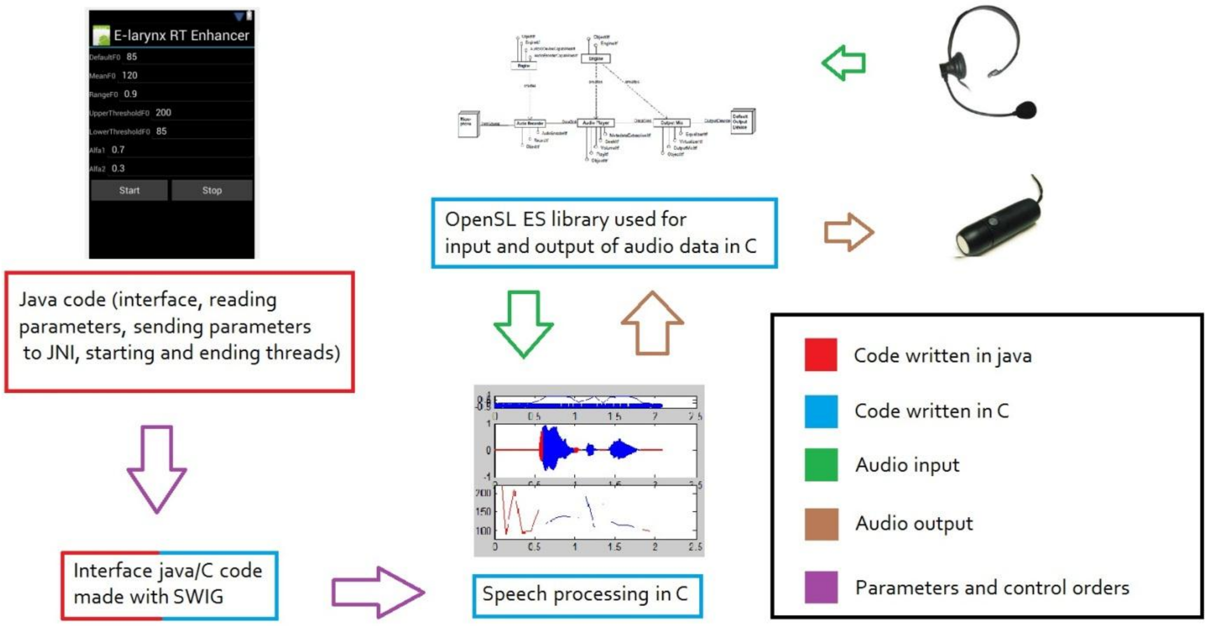

E-larynx RT Enhancer
DefaultF0 85
MeanF0 120
RangeF0 0.9
UpperThresholdF0 200
LowerThresholdF0 85
Alfa1 0.7
Alfa2 0.3
Start
Stop
OpenSL ES library used for input and output of audio data in C
Java code (interface, reading parameters, sending parameters to JNI, starting and ending threads)
Interface java/C code made with SWIG
Speech processing in C
🟥 Code written in java
🟦 Code written in C
🟩 Audio input
🟫 Audio output
🟪 Parameters and control orders

Figure A-0.1 Overall diagram of operation. The audio input is taken from the microphone and sended to the speech processing part then, once processed, the output audio is sended to the library and then to the e-larynx. The java code is used for showing the inteface and reading the parameters. Once the parameters are read and the start button is pushed, the java code send the parameters to the native method and gives the order to start

A.2 The Java and XML Code

A.2.1 AudiotestActivity.java

This function loads the interface from the main.xml file. After pressing the button start the java code read the parameters that are written in the edit text boxes and start the native method thread using these parameters. When the button stop is pushed the native method thread is stopped and all the memory is liberated.

After that new parameters could be loaded and a new thread could be started.

A.2.2 AndroidManifest.xml

In this file the user permissions are written. In our case they are: *MODIFY_AUDIO_SETTINGS*, *RECORD_AUDIO* and *WRITE_EXTERNAL_STORAGE*.

A.2.3 Main.xml

In this file the user interface is created. It consists of 1 edit text box and 1 text view strings for each configurable parameter and two buttons for starting and stopping the native method thread.

A.3 The Interface between Java and C. Swig and JNI Files

A.3.1 Java_interface_wrap.cpp

This file defines the native methods using the spetial nomenclature of the JNI inside C. It's generated automatically by SWIG

A.3.1 Opensl_example.java and opensl_exampleJNI.java

This two files are generated automatically by SWIG also. They declare the native methods in java, and opensl_exampleJNI also loads the library created with the NDK and all the c files.

A.4 The Speech Processing in C

Opensl_example.c

This is the thread that starts when pushing the start button in the user interface. First it loads all the necesary files using the directive #include. Then it defines a few parameters using #define, like the blocksize the hopsize the sampling rate and the returning values of the functions.

After that it creates all the structures and the variables necessary for the operation of the program. Once the thread starts it initializes all the functions that the program is going to use. When the initialization is finished the main loop start. This loop will continue working until the thread is stopped using the user interface.

Inside this loop, the program first reads the buffers of the input audio values using the OpenSL ES library. Then it performs a window and a voice detector. After that it does the FFT of the actual block of samples and it does the spectralsubtraction. When this is done, the overlapp and add is performed and then it starts with the analysis of frequencies.

First it measure the formant F2 in the fxdetection function and calculates the voiced and unvoiced regions of the speech. Later it uses the information of the F2 and the input parameters to calculate a f_0 contour in the function f0synthesis.

The last step is to use the pitchmarking for signaling the correct places to set deltas (having in mind the correct separation that indicates the f_0) and then it performs the pulsegenerator (in this case it just put a 1 in each sample indicated before)

If no frequency was read in the current block, a default frequency is built for the output. This frequency is chosen from the input parameters.

Once an output buffer is created, the OpenSL ES library is used again to send this to the output of the device. After that the loop starts again

A.5 The use of the OpenSL ES library.

A.5.1 OpenSL_IO.c

This file contains different methods:

A.5.1.1 openSLCreateEngine

This method creates, realizes and gets the interface of the Engine object. This is necessary for starting the session of OpenSL ES and creating further objects using the engine interface.

A.5.1.2 openSLPlayOpen

It’s the function that creates all that is necessary to output audio. It realizes the output mix object. Then it configures the audio player, and realizes and gets its interface. After that it gets the buffer queue interface and register the callbacks on the buffer queue. In its last step it sets the player to play. (If there is no buffer in the player there will be no sound until the player it’s fed)

The play interface its used to control the playback state of an object and get callbacks from the player.

The buffer queue interface is used for streaming audio data. It provides a method for queueing buffers in a player and also provides a callback function that is called when a buffer in the queue is completed. It’s possible to query the state of the buffer queue to know playback status.

A.5.1.3 openSLRecOpen

It’s the function that creates all that is necessary to record the audio. First it configures the source. Then it creates, realizes and gets the interface of the recorder object. Then it needs to get the buffer queue interface and as last step it registers the callbacks of the buffer queue.

The record interface is used for controlling the recording state of an object. In this case it puts the recorder object to record or stops it from doing it.

A.5.1.4 openSLDestroyEngine

This function destroys all objects and interfaces that have been used.

A.5.1.5 android_OpenAudioDevice

This function calls the engine, record and play functions to create, realize and get interfaces of all the objects that are going to be needed. For that purpose, it calls the functions openSLCreateEngine, openSLPlayOpen and openSLRecOpen. After that it allocates the memory for the input and output buffers

A.5.1.6 android_CloseAudioDevice

First it calls openSLDestroyEngine and then it destroys the threadlocks and frees the memory

A.5.1.7 android_AudioIn

It gets a buffer of audio samples from the device input. It uses the method enqueue of the buffer queue interface to do it.

A.5.1.8 android_AudioOut

It puts a buffer of audio samples to the output of the device. It uses the method enqueue of the buffer queue interface to do it.

A.5.1.9 Threadlocks functions

They ensure synchrony between callbacks and processing code.

createThreadLock: It creates a threadlock

waitThreadLock : It synchronizes two threads

notifyThreadLock : It sends a signal to another thread when an event occurs

destroyThreadLock: It destroys a threadlock

(Look OpenSL ES document for more info) add that to the references.

A.6 The modules of the main algorithm

This modules are obtained from Noisternig Master Thesis and they have been slightly modified for working in the Android framework [Noist11]

A.6.1 Centroid

It is used to calculate the centroid of the frequency of the samples in order to calculate if the sounds are voiced or unvoiced. It's used in the vuv function. It worked properly

A.6.2 Comp

Used for compressing elements if the optional label of compression is active.

A.6.3 Config.h

This file is used to include all the configuration names and labels and for easily changing configurating options of the code. Here there are only included the configuration options that are useful to our implementation of the code. The others were eliminated

A.6.4 F0detection

This piece of code detects the fundamental frequency of the voice. It is not totally needed because the fundamental frequency of a electrolarynx sound is going to have always the same f_0 so a default f_0 value could be used for the rest of the code.

A.6.5 F0synthesis

It takes the fxdetection output and calculates an f_0 contour based on the F2 formant and the input parameters. It performs a smoothing using a median filter a linear smoothing and a jump eliminator. Because this seemed not to be enough after calculating the f_0 sample of each block another smoothing is done using a weighted average filter with two alfa parameters that are also chosen by the user.

A.6.6 FFT

This file has the algorithm of the FFT. It takes the initialization parameters and the configuration options and sends them to the FFT library kissfft.

A.6.7 FFT library, kissfft

It takes the samples and perform a normal FFT. Because a library is used this file is suposed to work properly and is not revised.

A.6.8 FIR

Not implemented in this version of the program. It could be included in a future version to adapt the input of the microphone.

A.6.9 Fxdetection

This file uses a linear predictive code to calculate the formants of the input signal. It calculates up to 6 formant but we only need the formant number two for the creation of the f_0 contour.

A.6.10 Helpers

Functions that are used in a few different files are here grouped to have easier access to them. Just including the helpers header in each of those functions instead of the whole helping method makes the work easier.

A.6.11 Iir

Not implemented in this version of the program. In future versions the add of an Iir filter for the input would be interesting to avoid some problems that could appear with low frequencies. This file was creating a lot of problems in the android framework so it was not used for this implementation.

A.6.12 Linsmoothfilter

It’s used in the creation of the f_0 for smoothing the values from one block to the next one

A.6.13 LPC

This function is called inside the fxdetection.c file. It performs a linear predictive code analysis to get a function simmilar to the input, then get the zeroes and poles analyze them and using this information get the formants of the input.

A.6.14 Markerlist

It’s used to add new entries to a list of patchmarks and to manage this entries.

A.6.15 Medianfilter

It’s used inside the f0synthesis to smooth the array of f_0 samples calculated. Consists on using the median to filter the results.

A.6.16 Normalize

It’s an interface for channel-wise sample-by-sample processing

A.6.17 Padding

It includes zeroes at the end of a buffer.

A.6.18 Percentile

It calculated the percentile values for the spectral subtraction pins.

A.6.19 Pitchmarking

It indicates the index of the buffer where there should be signal. It calculates the correct indexes based on the f0contour. It calculates the distance between two values and then put a one in these indexes. The pulse generator would be in charge later to transform this into an excitation signal for the e-larynx

A.6.20 Polyroots

It's used inside the LPC file to calculate the roots of the function created.

A.6.21 Pulsegenerator

It takes the information of the pitchmarking function and transform it into an excitation signal for the e-larynx. Right now it's only possible to use deltas as output, but it would be interesting to add other kind of pulses and check if this improves the sound of the e-larynx

A.6.22 Spectralsubtraction

It estimates the noise power spectrum and then subtracts this spectrum from the signal power spectrum. The noise is calculated during non-speech intervals. The Spectral subtraction makes the processing of the speech signal easier.

A.6.23 Vad

It calculates if in the actual block there is voice. It's done with a basic power threshold. This threshold could be adjustated in the config.h file.

A.6.24 Vuv

It determines if the actual block is voiced or unvoiced. This is done using the centroid of the frequencies of the block. If this centroid surpasses a threshold then is unvoiced, if not then it's voiced. This threshold could be adjustated in the config.h file as well.

A.6.25 Window

A window is used to avoid the introduction of non-desired frequencies between blocks. The coefficients from the window are read from the Coeffs folder.

A.6.26 Coeffs folder

It contains all the coefficients for the different filters. Because they are hard coded it optimizates the processing time.

Bibliography

[AndAct] W. Frank Ableson, Robi Sen, Chris King, C. Enrique Ortiz. *Android in Action, 3rd edition*, Chapters 13 and 19, Manning Publications Co. 2012

[AndADT] http://developer.android.com/tools/index.html (Last visited September 2012)

[AndNDK] http://developer.android.com/tools/sdk/ndk/index.html (Last visited September 2012)

[Android] http://www.android.com/about/ (Last visited September 2012)

[Eclipse] http://www.eclipse.org/org/ (Last visited September 2012)

[Hag09] Martin Hagmüller. *Speech Enhancement for Disordered and Substitution Voices*, Chapters 5 and 6, Graz, Austria, September 2009

[Noist11] Thomas Noisternig. *Real-Time Enhancement of E-Larynx Speech Signals*, Graz, Austria, July 2011

[Notepad++] http://notepad-plus-plus.org/ (Last visited September 2012)

[OpenSLES] The Khronos Group. *OpenSL ES specification version 1.0.1*. September 2009

[Reyn08] Douglas Reynolds. *Gaussian Mixture Models*. February 2008

[SPOKEN] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. May 2001

[SWIG] http://www.swig.org/ (Last visited September 2012)