USER GUIDE FOR FREQUENCY DOMAIN ADAPTIVE COMPRESSION ON iOS PLATFORM FOR HEARING AID APPLICATIONS

Kashyap Patel, Dr. Issa Panahi
STATISTICAL SIGNAL PROCESSING LABORATORY (SSPRL)
UNIVERSITY OF TEXAS AT DALLAS

April 2019

This work was supported by the National Institute of the Deafness and Other Communication Disorders (NIDCD) of the National Institutes of Health (NIH) under the award number 1R01DC015430-01. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

Table of Contents

INTRODUCTION

Smartphones are used by many people. The very useful features of smartphone make it a suitable stand-alone device for hearing improvement. In this guide, we present a frequency-based multi-band compression strategy implemented on a smartphone working as an assistive hearing device. The contents of this user guide gives you the steps to implement the Frequency based Adaptive Compression algorithm on iOS devices (iPhone and iPad) and the steps to be followed after installing the app on the smartphone. This app will be an open source and portable research platform for hearing improvement studies.

This user guide covers the software tools required for implementing the algorithm, how to run C codes on iOS devices and usage of other tools that are quite helpful in creating audio apps for audio playback in real time. The screenshot of the first look of our app is as shown in figure as below

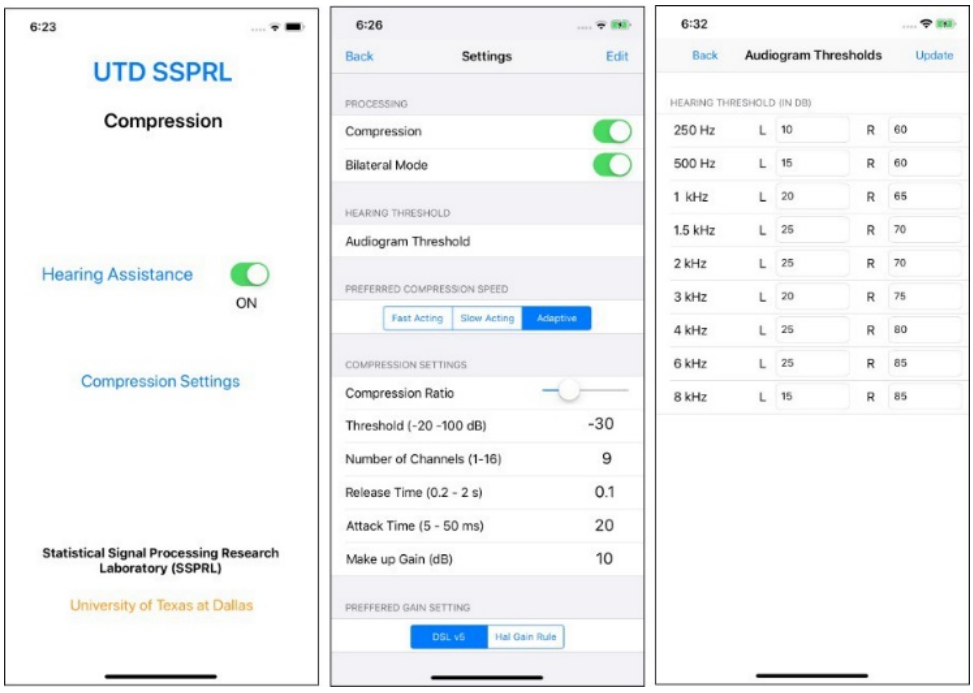

Fig 1. Graphical User Interface of the Compression Application

The codes can be accessed and used with proper consent of the author for further improvements in research activities related to hearing aids.

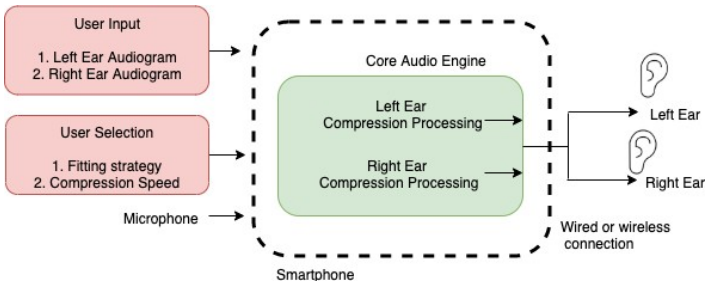

Fig 2: Overall working principle for bilateral mode of the application

1. SOFTWARE TOOLS

iOS is a mobile operating system created and developed by Apple Inc. for devices like iPhones, iPads and iPods. The languages used in this system are C, C++, Objective-C and Swift. All third-party apps are authorized and can be made available through the Apple's app store for devices with iPhone OS 2.0 and higher. The apps must be written in either Swift or Objective-C. We use Objective-C as it has an option of running C or C++ codes within the iOS environment. The codes must be compiled specifically for iOS and the 64-bit ARM architecture (typically using Xcode).

To design apps for iOS devices, a Mac OS is needed. The entire process can be carried out using Xcode IDE (Integrated Development Environment). It is a software package that can be installed for free from the app store. It is not possible to develop iOS apps using a Windows or Linux based computers.

From iOS version 9 and Xcode version 7 onwards, it has become possible to perform app development without the need to enroll or register in the Apple Developer Program. It is worth mentioning that although apps developed can be run on iPhones/iPads, they cannot be published on the App Store without registering as an Apple Developer.

To download the latest version of Xcode

1. Open the App Store app on your Mac (by default it's in the Dock).
2. In the search field in the top-right corner, type Xcode and press the Return key.
   The Xcode app shows up as the first search result.
3. Click Get and then click Install App.
4. Enter your Apple ID and password when prompted.
   Xcode is downloaded into your /Applications directory.

2. BUILD AN IOS APP

2.1 Programming Language

For creating iOS apps, Objective-C is used to create the required shell. Objective-C constitutes a superset of C, allowing one to seamlessly call C functions by just importing a header file. The execution of C codes occurs efficiently within the Objective-C environment due to the absence of any overhead translation or matching.

2.2 Creating Objective-C Shell

The creation of an Objective-C shell starts by creating a GUI to link data to a C code. The steps needed for creating a basic shell are listed below:

- Open Xcode and select "Create a new Xcode project" on the startup splash screen, see Figure 2. If in case, no splash screen comes up, select File->New->Project or press Command + Shift + n.

Figure 2

- On the page that comes up, shown in Figure 3, choose the template of the project as "Single View Application" and click Next.

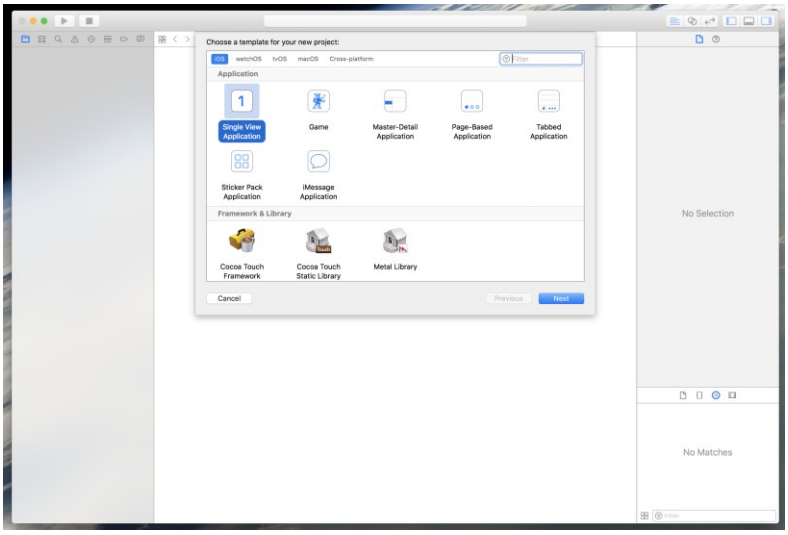

Figure 3

- For options that come up, set the name of the project, select Team as "**None**", set your organization name and the organization identifier. Remember to set the Language to "**Objective-C**" as this option **cannot be reversed**. Select devices as "**Universal**" and deselect all the options.
- Store the project in the Directory of your choice and click **Create**. If desired, deselect the option "**Create Git Repository**".
- To be able to build the app for iPhones, you would need to sign the application and select a Team. In Xcode, select "User Name (Personal Team)" for Team and Xcode will automatically sign the application.

2.3 Adding C File

- To add a C file to the project, navigate to the "JMAP SE" folder in the project navigator and select "New File…", shown in Figure 4.
- In the "iOS" tab, shown in Figure 5, under "Source", select "C File" and click Next.

- On the page that comes up, write the file name and also select the option that states“Also create a header file” and click Next.

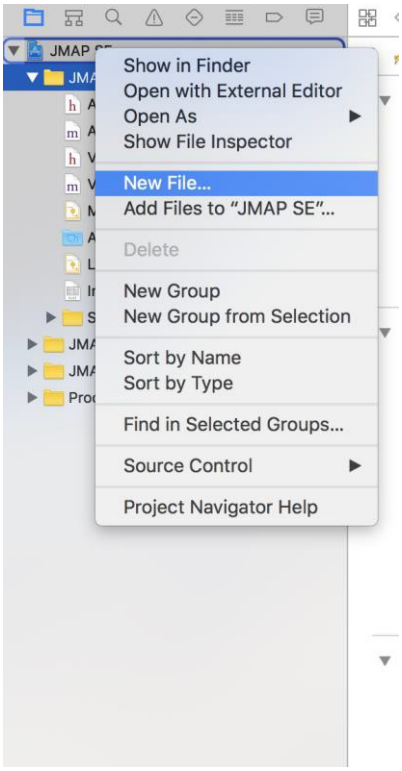

Figure 4

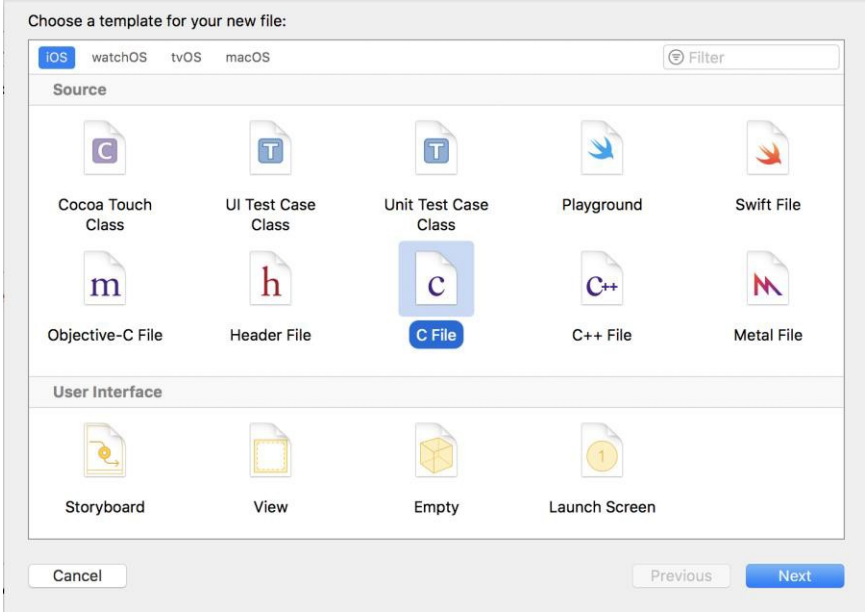

Figure 5

- An existing C code can be directly attached to the app, instead of a writing a new C code. It will be explained in the further topics.
- Once the C code is attached, add the header file in the “View Controller.m” file to use the contents in C code.

3. SUPERPOWERED SDK

- This is a low latency audio SDK for iOS and Android.
- Superpowered accomplishes this using patent-pending DSP optimization technology to achieve desktop grade performance on mobile devices.
- Superpowered technology uses less than half the power of Apple's Core Audio and is more than twice as fast as Apple's vDSP.
- The Superpowered Audio SDK empowers developers to remove CPU resource limitations, and develop cross-platform audio for iOS, Android and wearable devices. It includes:

1. Example apps/projects for iOS, OSX and Android
2. Static library files
3. Decoder for MP3, AAC, WAV, AIFF and STEMS
4. Advanced Audio Player (including time stretching, pitch shifting, resampling, looping, scratching, etc.)
5. HTTP Live Streaming
6. Effects: echo, flanger, gate, reverb, rool, whoosh, 3 band equalizers, biquad IIR filters (low-pass, high-pass, bandpass, high-shelf, low-shelf, parametric, notch)
7. Dynamics: compressor, limiter, clipper.
8. Time Stretching and Pitch Shifting, resampler
9. Polar and Complex FFT
10. Recorder
11. Open-source audio system input/output classes
12. Time-domain to frequency domain class (including inverse)
13. Bandpass filterbank for time-domain frequency analysis
14. Audio analyzer: key detection, bpm detection, beatgrid detection, waveform generation, loudness/peak analysis

15. Stereo and mono mixers
16. Simple audio functions (volume, volume ramp, peak, float-short conversion, interleaving and de-interleaving)

- The code is made as an open source and can be downloaded by going to the following link below,
  http://superpowered.com/
- Once it is downloaded you can go to examples_ios folder as shown in Figure 6, to find various examples provided by superpowered that can be used and modified based on the requirements for our applications.
- When you click on the SuperpoweredFrequencyDomain as shown in Figure 6, the code can be opened on Xcode directly.
- All modifications can be done in "View Controller.mm" file present inside as shown in Figure 7, the data in frequency domain can be modified.

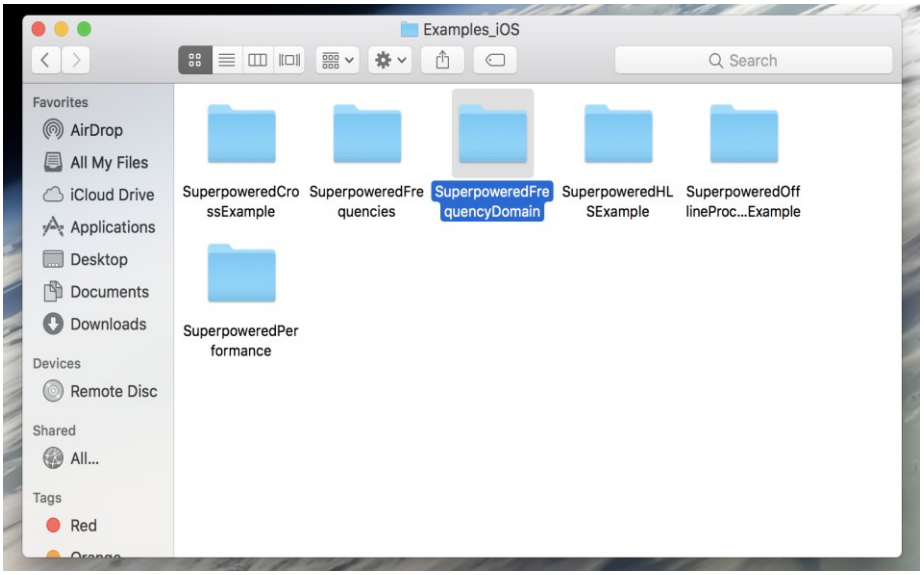

Figure 6

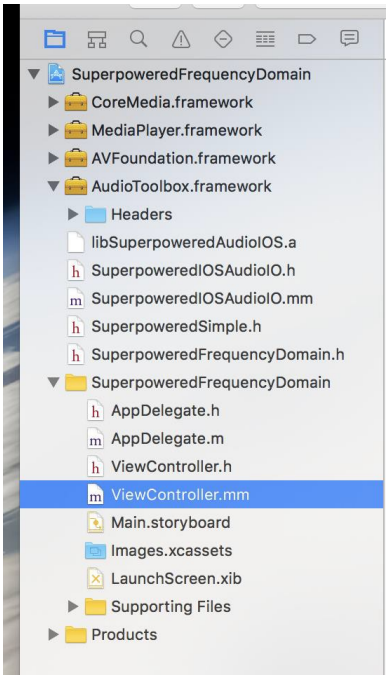

Figure 7

- The input will be considered to be in time domain, then converted to frequency domain and once you are done with the modifications the data will be again converted back to time domain to obtain the output.
- Refer the block diagram shown in Figure 8 for better understanding,

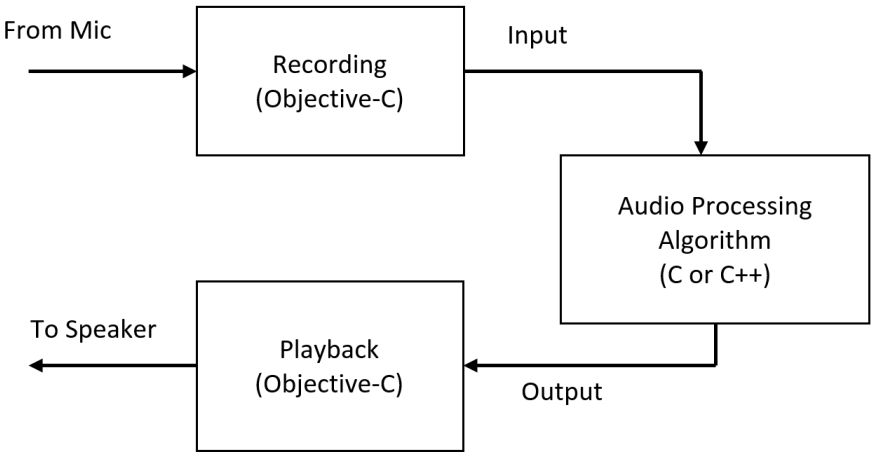

Figure 8

5. Frequency based Multiband Adaptive Compression

To investigate the behavior and performance of the proposed design, the algorithm presented is implemented on an iOS-based smartphone (iPhone). The algorithm is written in C, C++ and the graphical user interface is implemented using Objective C on Xcode IDE. Low-level Core Audio framework was implemented on iOS to achieve real-time recording and playback using device audio hardware for minimum input/output latency. Input/output signal runs at a 16 kHz sampling rate and a frame size of $N = 256$ samples (16 ms), optimized for low latency. One reason to operate at 16 kHz and not 48 kHz is to allow Bluetooth Low Energy (LE) connection to existing traditional HAD's and wireless earbuds. Windowed input frame is transformed into the frequency domain with 1024 point FFT and a HOP size of 512 samples. Analysis and synthesis windows of length 1024 were used to reduce the temporal aliasing. Transformed signal has $K = 512$, frequency bins ranging from 0 to 8000 Hz and resolution of 15.6 Hz. Each frame is processed within 1.8 ms which is less than the 16 ms frame size ensuring that the application operates without any audio glitches.

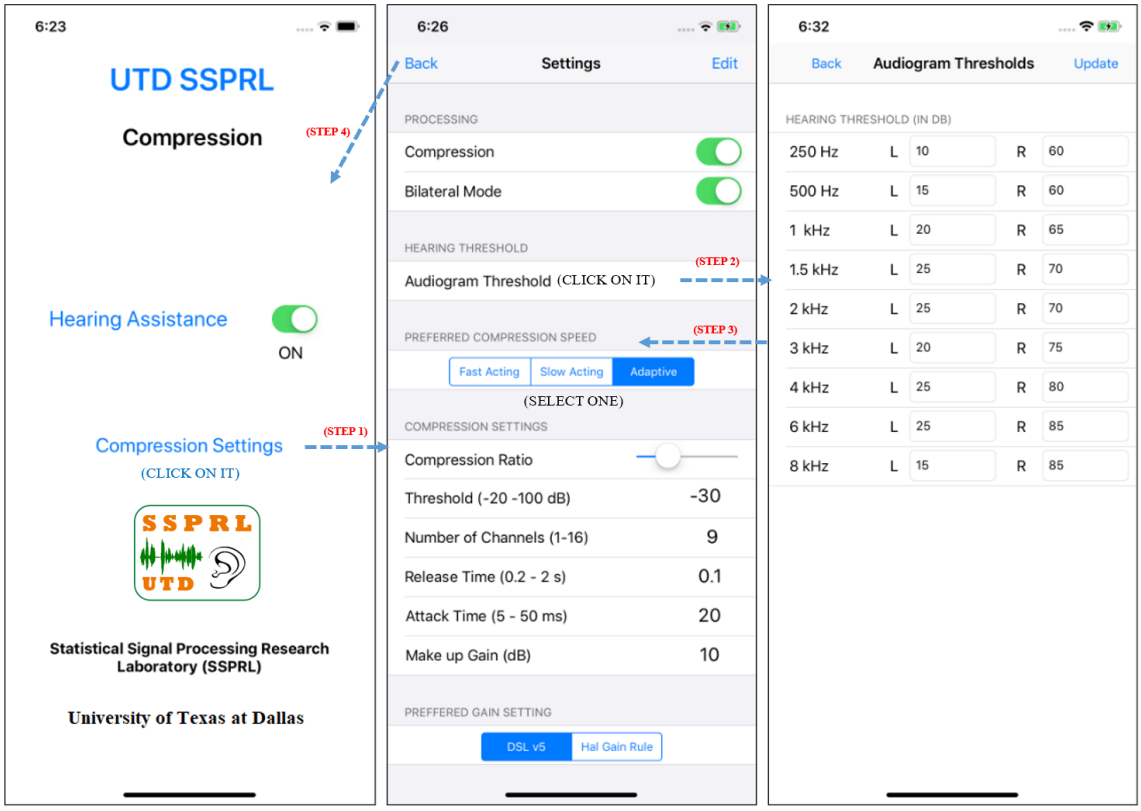

Figure 9: Screenshot of the developed smartphone application

The microphone on the smartphone records the data in a frame-based manner. This input signal is further divided into two channels, right and left for bilateral hearing loss. The block diagram is shown in Fig. 6. Bilateral hearing loss is having a different hearing threshold for each ear. Fig. 5 shows the graphical user interface (GUI) of the application. The main page has a switch for the hearing assistance option and a button for the compression settings page. The compression setting page has options to tune all the parameters for each band. User needs to fill the audiogram thresholds to meet his/her hearing requirements. An audiogram can also be generated online or through many available smartphone applications. Setting page has also options to choose the prescriptive fitting strategy via either ”DSL - v5” or ”Half - gain rule”. User can also select the compression speed from the ”fast-acting compression”, ”slow-acting compression” and the ”adaptive compression”.

To learn more about the application, please refer to our video demos on http://www.utdallas.edu/ssprl/hearing-aid-project/

6. References

[1] Dimitrios Giannoulis, Michael Massberg, and Joshua D Reiss, “Parameter automation in a dynamic range compressor,” *Journal of the Audio Engineering Society*, vol. 61, no. 10, pp. 716– 726, 2013.
[2] Francis Kuk, Chris Slugocki, Petri Korhonen, Eric Seper, and Ole Hau, “Evaluation of the efficacy of a dual variable speed compressor over a single fixed speed compressor,” Journal of the American Academy of Audiology, 2019.
[3] Kates, James M., et al. "Using objective metrics to measure hearing aid performance." *Ear and hearing* 39.6 (2018): 1165-1175.
[4] Hao, Yiya, Ziyan Zou, and Issa M. Panahi, “A robust smartphone based multi-channel dynamic-range audio compressor for hearing aids,” *The Journal of the Acoustical Society of America* 143.3 (2018): 1961-1961.
[5] Z. Zou, Y. Hao and I Panahi, “Design of Compensated Multi-Channel Dynamic-Range Compressor for Hearing Aid Devices using Polyphase Implementation,” *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, HI, USA, 2018, pp. 429-432.