LUND UNIVERSITY

# DEVELOPMENT OF SMARTPHONE APP FOR REVERBERATION TIME MEASUREMENTS IN ROOMS WITH FURNITURE

YUZHAO CUI and YILIN WANG

Engineering Acoustics

*Master's Dissertation*

DEPARTMENT OF CONSTRUCTION SCIENCES

DIVISION OF ENGINEERING ACOUSTICS

ISRN LUTVDG/TVBA--18/5055--SE (1-72) | ISSN 0281-8477

MASTER'S DISSERTATION

# DEVELOPMENT OF SMARTPHONE APP FOR REVERBERATION TIME MEASUREMENTS IN ROOMS WITH FURNITURE

YUZHAO CUI and YILIN WANG

Supervisor: **DELPHINE BARD**, Associate Professor, Div. of Engineering Acoustics, LTH, Lund.

Examiner: **JUAN NEGREIRA**, Assistant Professor, Div. of Engineering Acoustics, LTH, Lund.



Printed by V-husets tryckeri LTH, Lund, Sweden, November 2018 *(Pl)*.

**For information, address:**
Division of Engineering Acoustics,
Faculty of Engineering LTH, Lund University, Box 118, SE-221 00 Lund, Sweden.

Homepage: www.akustik.lth.se

# Abstract

Room acoustics, as a branch of architectural acoustics, has got more and more attention nowadays. One of the aims of the room acoustics is to develop different method to evaluate the acoustic performance of the rooms and to address human acoustic comfort needs. Meanwhile, the process of reverberation plays an important role in room acoustics, and it is also an evaluation standard of evaluating room acoustics effects.

Reverberation time: The time it takes for a signal (sound) to drop by 60dB, which is an essential parameter about the “echo” in measured room. The reverberation time in a room depends on its absorption value and volume. If the volume of the room is large and the absorption value is small, RT60 is long. On the contrary, with a small volume and large absorption value of a room, RT60 will be short. More importantly, the sound will be unnatural or ambiguous if RT60 is too short or too long.

Reverberation time is one of important parameters in room acoustics. It is employed as a standardized method to evaluate the acoustic comfort of an enclosure, which is one of the reasons why we research on this project.

Nowadays Android phones are very popular around the world. In the statistics of operating systems of worldwide smart-phone in 2017, Android operating system’s market share is 85.9% in the word, which means Android device has the biggest number of users in the world.

From what have been discussed above, the direction of our thesis is designed. The goal of our project is to build a convenient and fast Android app. This app can measure the reverberation time of indoor environment using Android phone, then allows user to store the measured data in this phone for further use.

# Acknowledge

First of all, we would like to thank our supervisor and examiner: Delphine Bard and Juan Negreira. Because of their patient teaching and rich standard experience supporting, that help us fast understanding the related knowledge and overcome the obstacles in this work. What is more, we would like to thank all the help of PHD, teachers, classmates in Wireless Communication and Acoustics in our study period.

I ( Yuzhao Cui ) would like to thank my partner Yilin Wang for his collaboration and effort put in this work firstly. Furthermore I want to express my gratitude to Cheng Qian , who once helped our measurement and provided useful advice .

I (Yilin Wang) want to thank my thesis partner Yuzhao Cui at first, for his help we cooperate and complete the thesis work perfectly. And I also would like to thank my friends, for their good advice and idea let us get out the difficulty in the work.

# Table of Contents

# List of Figures

# List of Tables

# List of Equations

# List of Acronyms

RT60: Time for sound to decay to 60dB of its original intensity. Reverberation time.
T20: Time for sound to decay to 20dB of its original intensity.
T30: Time for sound to decay to 30dB of its original intensity.
FFT: Fast Fourier Transform
IFFT: Inverse Fast Fourier Transform
$L_p$: Sound Pressure Level
SPL: Sound Pressure Level

# Chapter 1 Introduction

In 1994, noise protection becomes a separate topic in the Swedish national building regulations for the first time. Noise sources could be the traffic noise from the outside of the building and it could also be the noise generated from the neighbors or even from the self-generated noise. Among these different kinds of noise, the low frequency noise is paramount annoying, especially the impact noise lower than 100Hz [1]. However, the frequency range of the current standers are restricted from 100Hz to 3150Hz (ISO 140, ISO 171). Although adaptation terms are added in the evaluation, not enough attentions are paid to the low frequency noise [1-3]. In order to evaluate and improve acoustic performance, different standardized parameters are introduced, such as airborne sound reduction index (ISO 717-1, ISO 140-4, EN ISO 12354-1, ISO 16283-1) and airborne sound transmission class (ASTM E413), weighted standardized impact sound pressure level (ISO 717-2, ISO 140-7, EN ISO 12354-2, ISO 16283-2) and impact sound insulation index (ISO 717-2, ISO 140-7, EN ISO 12354-2, ISO 16283-2), etc. Among these different indicators, the reverberation time evaluation is one of important parameter. Since it can not only evaluate the sound decay and describe the acoustic character of a room but can also evaluate the absorption coefficient through standard method (ISO 3382-1).

Reverberation time is one of the most important and stable indicators in room acoustics. It is the basic parameter of sound quality evaluation, material acoustic performance testing, noise control and many other fields. The reverberation time has always been recognized, with a clear concept and objective parameters that are well correlated with subjective feelings [4]. Moderating reverberation can significantly improve the sound quality and change tone and type of the sound.

Except reverberation time there are also many indicators which could characterize the acoustic performance of room, such as speech intelligibility (D50), clarity for speech (C50) and clarity for music (C80), etc.

## 1.1 Decibel

Decibel is a unit of measure the proportion of two identical units, which expresses the ratio of a value to a reference value, and its unit is dB. It is widely used in many fields such as electronics, signal, communication and so on [5]. In this thesis, decibel is used to measure the SPL(sound pressure level), and the formula it is shown below:

$$L_{dB} = 10 \times \log_{10}(\frac{A_1^2}{A_0^2}) = 20 \times \log_{10}(\frac{A_1}{A_0})$$

**Equation 1**: SPL Equation.

In this equation, A1 is the root-mean-square valued of the SPL which can be acquired by the android equipment. It is very useful in measurement part about the sound, which describes a ratio by using modest size number.

## 1.2 Reverberation

Reverberation is an acoustic phenomenon in a confined space. As we all know, when sound waves are reflected in the surrounding walls in indoor environment this phenomenon will be repeated many times. If this kind of reflection continues to exist after the sound reaches directly to the listener for 50 ms, and lasts for a while until it decays and disappears, which sounds to be remnant. This process and phenomenon is called reverberation.

Reverberation is the result of constantly reflection of sound by the interface of the objects sound touches. In indoor environment, it can increase the sound and reduce the speaking clarity. Under some certain conditions, reducing reverberation is vital. For example, in the space for language use such as cinemas, classroom, recording studio, meeting room, etc., reverberation should be reduced to make the speaking more clear [6]. Therefore, reverberation is needed sometimes, for example, playing music in concert halls, theaters and so on, to make music more smooth and enjoyable.

After all, it is very necessary to use the different reverberation effects to the rooms that with different usage requirements.

## 1.3 Reverberation Time

After the sound source stop sounding, the sound energy in the space starts to decay immediately, and the time it takes from the SPL self-stabilize to attenuation of 60 dB is called reverberation time (RT60). The indicator describing the reverberation effect is RT60, with a unit of seconds. More details about Reverberation Time are introduced in chapter 2.1.

### 1.3.1 Reverberation Time Behaviour

There is a mathematical relationship between RT60 and indoor sound absorption, which is the famous Sabine formula in architectural acoustics:

$$T60=KV/(S\times a)$$

**Equation 2**: Sabine Formula.

In this formula, K is a constant related to the space humidity. In our case, the SPL are measured in the ambience condition. As a result, K=0.161 s/m. V is the room volume,

S is total area of the room wall, and a is the effective sound absorption coefficient of the room surface.

It can be seen from Sabine Formula that the larger volume of the room, the longer RT60. And the larger average sound absorption coefficient, the shorter RT60. So for the huge space such as the gymnasium, the RT60 will be very long if the sound absorption is not well performed, which will seriously affect the clarity of the speech.

Since the sound absorption is also related to the frequency, so RT60 is different at different frequencies. In general, the room sound quality index refers to the intermediate frequency reverberation time. According to the research, the ideal RT60 value is 1.8-2.2 seconds for concert hall, 1.3-1.5 seconds for theater, 1.0-1.4 seconds for multi-purpose auditorium, 0.6-1.0 seconds for cinemas, 0.4-0.8 seconds for ordinary classrooms, 0.2-0.4 seconds for studio, and for the gym is less than 2.0 seconds [6].

Therefore, the correct materials can be selected in architectural design to make a control of RT60, and ensure that the sound quality meets the requirement of users.

## 1.4 Android APP Introduction

### 1.4.1 Internal of APP (RT60 Measurement)

In our project, after user pressing the ‘Generate impulse’ button, the app start recording sound for 5 seconds. Sender (sound source) should generate a signal in this period such as clapping hands. When recording is finished, press ‘Analysis’ button to start analyzing the recorded sound. In the process of analyzing, with the added Octave Band filter, this sound will be transformed into the relations between SPL and time of the target Octave Bands.

Then RT60 in different Octave Band can be calculated by using the reverberation time calculation algorithm. This part about calculation algorithm will be introduced in detail in chapter 3.

### 1.4.2 External of APP

This APP provides a simple and convenience operation interface: users can type name, size of the room (length, height and weight), picture of measured room for easily recording and recognizing. Due to different emission intensity (soft or hard clapping) and different background noise interruption, the values of RT60 measured by the same device could be different in the same room. The way to solve this problem is sender and receiver each has at most 5 opportunities to measure the reverberation time. So

there can be up to 25 groups of test results can be measured out and the results can be more accurate after calculating the mean value. After measurement, users can check the line chart of the measured values for comparison, as well as the detailed information about the measured rooms.

Another function of this APP is that it allows the users to choose whether to store the measurement results into his or her Android phone's storage or not. Users can print out or mail the files of measurement results if needed. We will introduce all of these functions in detail in chapter 2.

## 1.5 Previous Work

In the international standard "Acoustics – Measurement of room acoustics parameters", it is figured out that reverberation time of a room is a predominant indicator in acoustical properties. The standards in this book give us a reasonable agreement in the measurement such as SPL, energy calculation, energy ratio, background noise level etc. [7].

In the book "Sound Insulation" [8] written by Carl Hopkins, "Building Acoustics" written by Tor Erik Vigran, a deeper study of room acoustics was taken. The details of sound transmission, sound absorption, sound reflection, reverberation time measurement and so on are described in detail in the book. It provides an abundant standard support on Fourier analysis field in Java coding [9].

## 1.6 Organization of Thesis

The thesis work is organized with:

**Chapter 1** Make a general introduction of reverberation time and our APP design.
**Chapter 2** Shows the method for the internal and external of this APP in detail.
**Chapter 3** Introduce how the java code application works in our project.
**Chapter 4** List the results in different room with different android phones and make a comparison with official equipment.
**Chapter 5&6** Make a conclusion and discussion what has been done in this thesis.
**Chapter 7** Give some suggestions for the future work.

# Chapter 2 Method to the Android APP

## 2.1 Reverberation Time Measurement

As we have known in Chapter 1, reverberation time is the time required for the reflections of a direct sound to decay 60dB [10]. Therefore, because different rooms have different sizes and different materials, we can find out that the reverberation time is always different by Sabine Formula.

Here is the very simplified RT60 measurement diagram when an impulse is generated, such as hands clapping in the room.

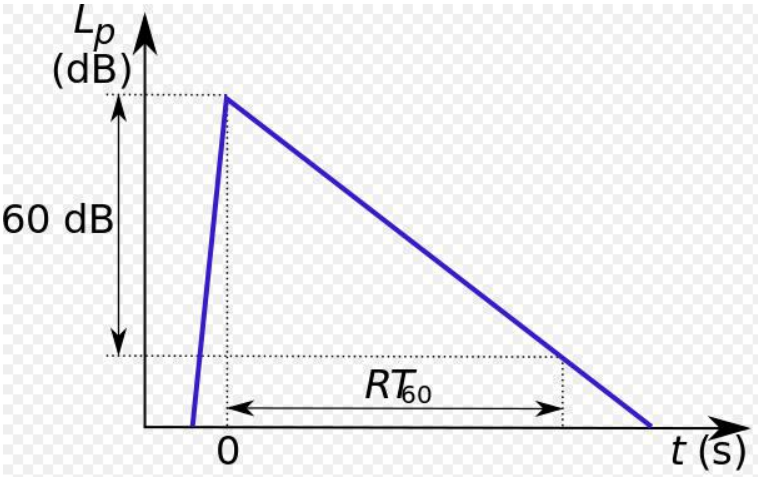


**Figure 1**: Simplified RT60 measurement diagram [10].

Figure 1 shows that when the sound source generates an impulse, the sound pressure level($L_p$) increases and reaches to the highest value after a short time, RT60 is the time it takes for the sound drops from the time point of the highest $L_p$ value to the time point when it decay 60dB. The reason why the slope of $L_p$ deceases process is less steeper than that of $L_p$ increases process is because the sound is reflected by room walls and objects in indoor environment.

There are many ways to measure RT60, the mostly common method is through T20 and T30. The reason is that it is difficult to generate a 'powerful enough' sound which has a decay up to 60dB in a room, especially at lower frequencies. We assume the decay is linear, so RT60 can be measured as the time it takes the sound to decease 20dB from original and multiple 3, or the sound to decease 30dB and multiple 2. Figure 2 is shown that make an easy way to understand how to measure T20 value.

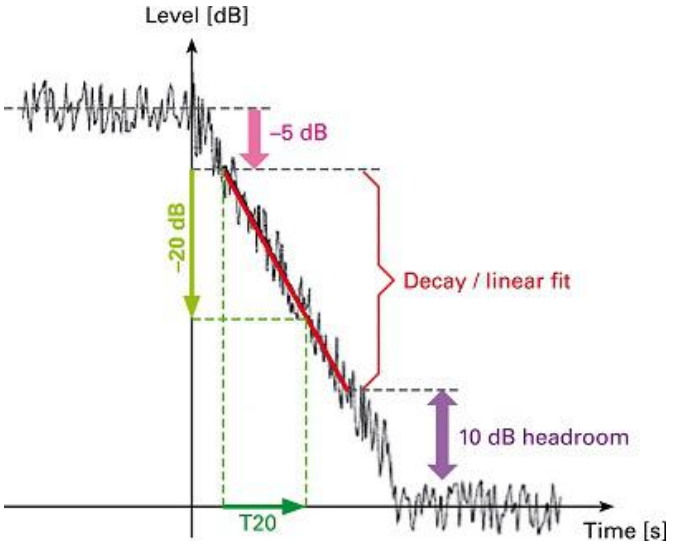


**Figure 2**: T20 Value measurement [10].

For the figure 2 express the SPL decay when a stable sound source turns off. T20 starts to be counted time when $L_p$ decreases 5dB and stops when $L_p$ decreases 20dB again. The time it takes is T20, and the decay is linear.

There are two different methods to measure the reverberation time. They are impulsive method and interrupted noise method (Figure 1 and 2). It has to mention that the interrupted noise method needs extra external equipment to generate the noise whereas the impulse method doesn't need any other specific instrument. In this project, the impulse method is employed to evaluate the reverberation time.

In our app project, hand clapping is taken as the recommended sound source to find out RT60. However, during the practical measurement, it can not be avoided that for some certain measurements, the $L_p$ drop condition of recorded clapping is not as ideal as figure 2 shows. The reason for this can be the limitation of our Android devices' recording hardware, contingency of measurements, influence of the indoor environment in measurements and so on. For this case, we improved our calculation algorithm based on the existed one: Firstly the decay is assumed to be linear. Then a suitable start point and end point are determined which are thought to be accurate enough to express the decay process. Finally, the RT60 results are calculated according to the $L_p$ and time at these two points. The calculation formula is shown as following:

$$RT_{60} = \frac{60}{-\frac{L_{dB\_start} - L_{dB\_end}}{Time_{start} - Time_{end}}}$$

**Equation 3**: Reverberation time measurement equation.

What's more, the solution for how to find the suitable start and end point, and how to apply it to Java coding will be introduced more in detail in chapter 3.3.

## 2.2 Method to the Internal Structure of APP

### 2.2.1 Fast Fourier Transform

This APP measures RT60 in each of selected frequency ranges. The solution is to use Fast Fourier transform to transform the signal (hands clapping sound) from time domain to frequency domain and find out the relation between time and SPL of each frequency range, then calculate the reverberation time using RT60 measurement method.

The fast Fourier transform (FFT) is an algorithm that samples a signal over a period of time (or space) and divides it into its frequency components [11]. Any periodic signal can be represented as an infinite stack of sinusoidal signal at different frequencies.

The mathematical perspective way of FFT is to calculate the frequency, sound press level and phase of different sinusoidal signals by direct measured sound of hands clapping. And for the physical point of FFT, it helps us to change the traditional time domain analysis signal to the thinking of analyzing problems in the frequency domain [12]. And the following three-dimensional graphics can help us understand this conversion better:

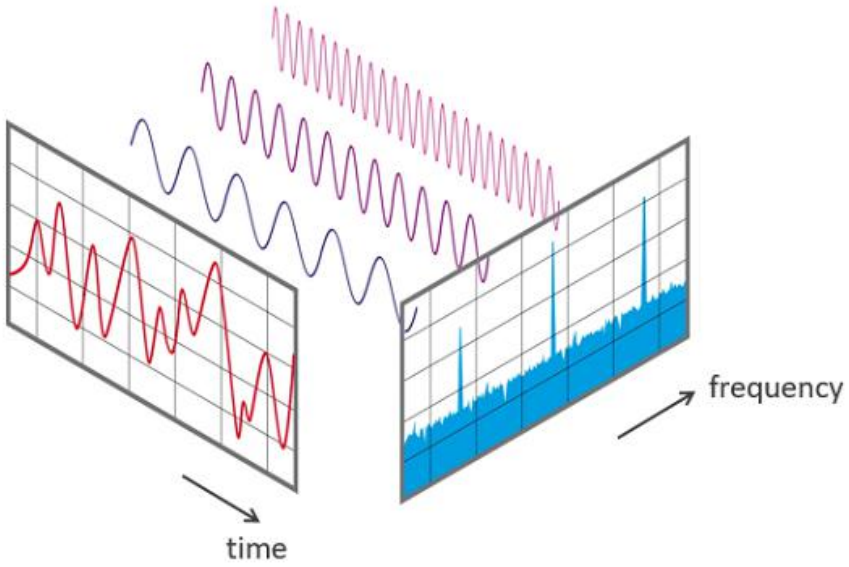


**Figure 3**: Example signal after FFT in time and frequency domain [11].

As we can see from figure 3, the time domain signal after the decomposition of FFT becomes the superposition of different sinusoidal signals, we can analyze the frequency of these signals and transform it to the frequency domain. Then it is clear to measure RT60 in each frequency range.

### 2.2.2 Octave Band Filter

The human ear feels the sound frequency from the lowest 20 Hz to the highest 20 KHz, and different frequency bands gives different feelings to people.

**20Hz-60Hz:** This part can give a strong feeling to the music.
**60Hz-250Hz:** This part is the low-frequency structure of music, which contains the basic sound of rhythm part, including the pitch and rhythm sounds.
**250Hz-4000Hz:** This part contains the low frequency harmonics of most instruments, while affecting the clarity of sounds such as vocals and instruments.
**4000Hz-5000Hz:** This is the frequency band that affects the sense of presence (distance feeling).
**5000Hz-16000Hz:** This part controls the brightness of the sound, macro brightness and clarity [13].

It is possible to analyze the source by frequency, but a waste of time, so people regulate the whole frequency range divided into several parts of the frequency band, and each one has its specific range, which is called octave band or one-third octave band. For the octave band, it is called octave width when upper frequency band is twice the lower frequency band. On the other hand, one-third octave band is the frequency band which upper band-edge frequency is the lower band frequency times the cube root of two [14].

One octave Band measurements are used when the frequency composition of a sound field is needed to be determined. Octave analysis is often used in noise control, hearing protection and sometimes in environmental noise issues [15].

Mainly used in environmental and noise control applications, 1/3 Octave Bands provide a further in-depth outlook on noise levels across the frequency composition [15].

| Band Number | Octave band center frequency(Hz) | One-third octave band center frequency(Hz) | Lower limit(Hz) | Upper limit(Hz) |
|---|---|---|---|---|
| 20 | 125 | 100 | 88 | 113 |
| 21 | 125 | 125 | 113 | 141 |
| 22 | 125 | 160 | 141 | 176 |
| 23 | 250 | 200 | 176 | 225 |
| 24 | 250 | 250 | 225 | 283 |
| 25 | 250 | 315 | 283 | 353 |
| 26 | 500 | 400 | 353 | 440 |
| 27 | 500 | 500 | 440 | 565 |
| 28 | 500 | 630 | 565 | 707 |
| 29 | 1000 | 800 | 707 | 880 |
| 30 | 1000 | 1000 | 880 | 1130 |
| 31 | 1000 | 1250 | 1130 | 1414 |
| 32 | 2000 | 1600 | 1414 | 1760 |
| 33 | 2000 | 2000 | 1760 | 2250 |
| 34 | 2000 | 2500 | 2250 | 2825 |
| 35 | 4000 | 3150 | 2825 | 3530 |
| 36 | 4000 | 4000 | 3530 | 4400 |
| 37 | 4000 | 5000 | 4400 | 5650 |
| 38 | 8000 | 6300 | 5650 | 7070 |
| 39 | 8000 | 8000 | 7070 | 8800 |
| 40 | 8000 | 10000 | 8800 | 11300 |

**Table 1**: Official frequency range of octave band filter and one-third octave band from 125Hz to 10000Hz.

In our thesis we require to measure RT60 from 125Hz to 4000Hz, as we checked the table 1 above the frequency range is 125Hz to 5650Hz actually. The way to measure RT60 in each specific frequency domain is to take sample points as much as we can, it helps to improve the accuracy of result, in each certain Octave Band, a mean value of Sound Pressure Level is got for each single time point, finally the one-to-one relations between Sound Pressure Level and time point for each Octave Band are used to calculate RT60 value for every Octave Band.

### 2.2.3 Pink Noise

Acoustic noise is any sound in the acoustic domain, either deliberate (e.g., music and speech) or unintended. In contrast, noise in electronics may not be audible to the human ear and may require instruments for detection [16].
Pink noise is one of the most commonly used in signal processing, the frequency component power of it is mainly distributed in the middle and low frequency band.

Pink noise has the same or similar energy compare to the audio data in some frequency ranges, and the energy is attenuated from low frequency to high, which the curve is 1/f.
In our thesis work, noise influences us greatly in the production process. The noise is not only pink noise, but also other kinds of it such as white noise, environmental noise, recording device noise and so on. We spent a lot of time adding a function called 'Speex' by using C++ coding language to help reduce the influence of all kinds of noise, and data of RT60 becomes more correct. This coding application will be introduced more detail in chapter 3.1.

## 2.3 External Operate Structure of APP

As is designed, this app mainly focuses on user experience. When users run this APP, there are two options will be displayed: 'Measurement' and 'View Old'.

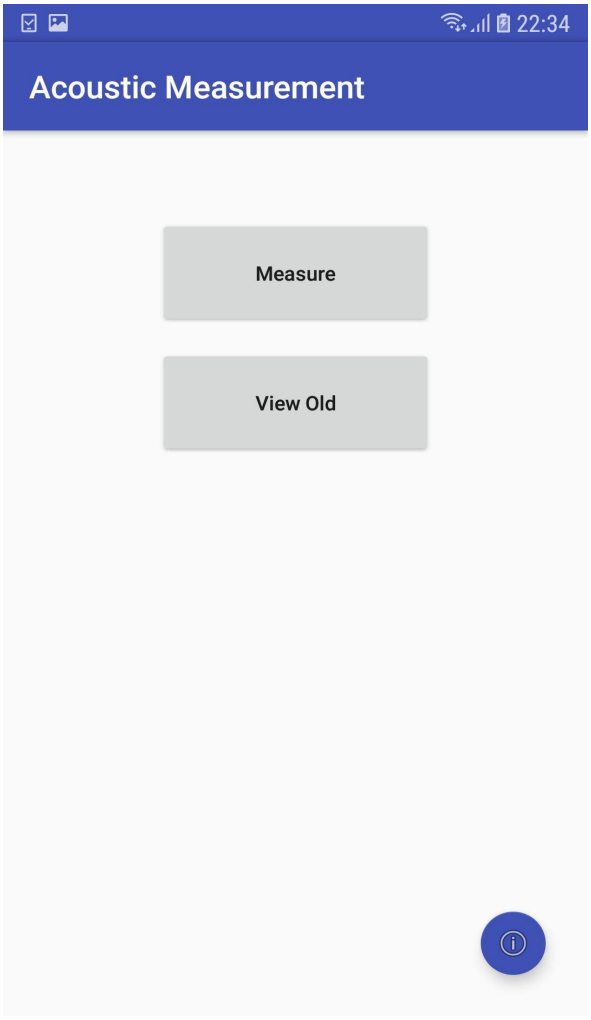


**Figure 4**: Screenshot for access screen in app.

In the Measurement options, people should type in the name and size (length, width and height) of the measured room. Then set the number of EP and MP which can be 5 at most for each, EP means number of the sender (impulse such as hands clapping) different positions, MP means number of the receiver (Android phone holder) different positions. Furthermore, users can take a photo by camera of the phone and add some notes that marks this room so that it is easy to recognize which room has been measured.

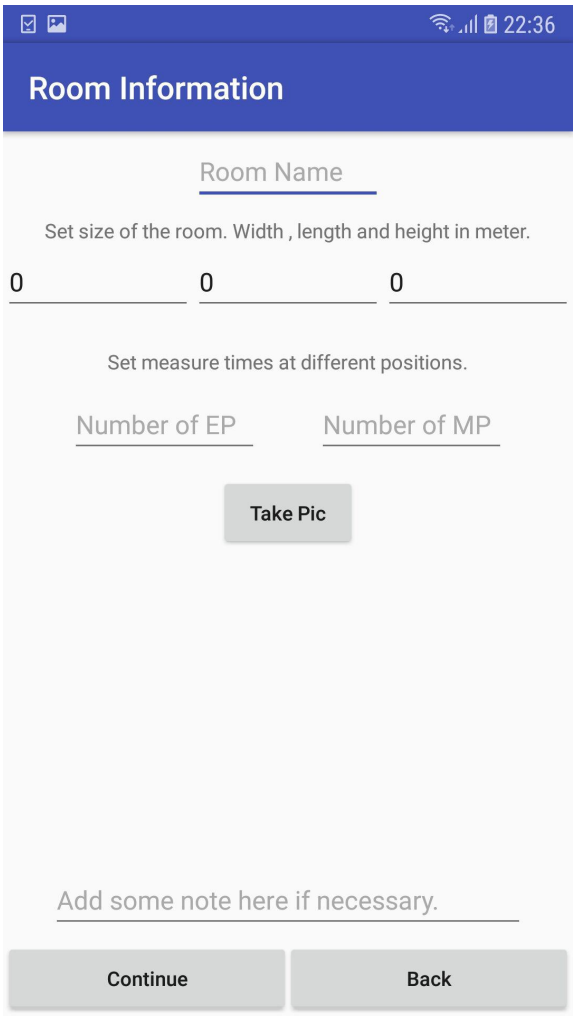


**Figure 5**: Screenshot for setting room information screen in app.

After users input the information and press ‘Continue’ button, Measurement operation interface will be displayed and there are two options named ‘Generate impulse’ and ‘Evaluate’. As we introduced in Chapter 1, Senders are recommended to clap the hand as an impulse after receiver press ‘Generate impulse’ button, the app will record the sound during 5 seconds after pressing the generate button. The next step is to start playing and analyzing the recorded sound when ‘Evaluate’ button is pressed, after a while the results of RT60 will be measured out and displayed.

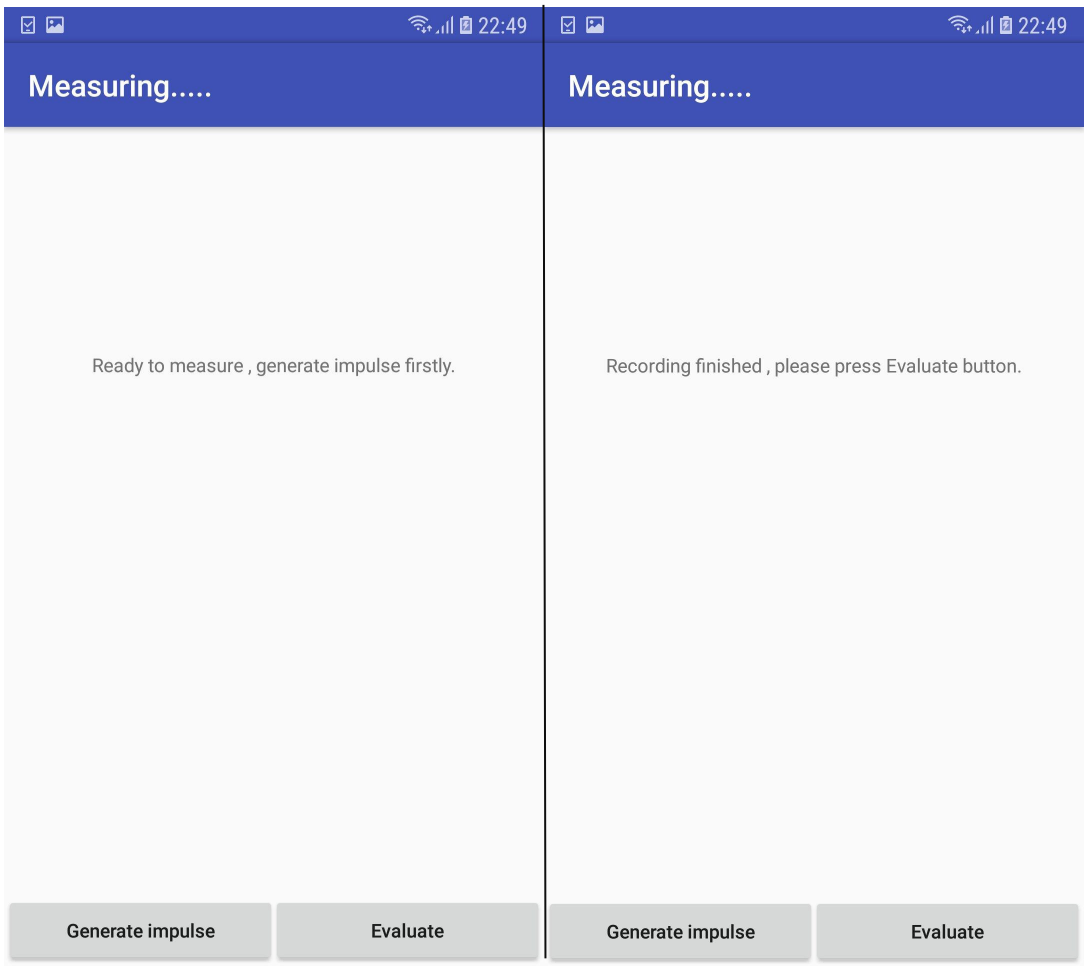


**Figure 6**: Screenshot for the process in measuring RT60 in app.

This measurement part will be repeated several times which is determined by Number of EP times Number of MP. At last a mean value of RT60 at each specific frequency band will be calculated out after the sender and receiver have done all measurements. Here is an example mean value of RT60 that we have measured 2×3 times by Samsung Galaxy S7 phone in bedroom:

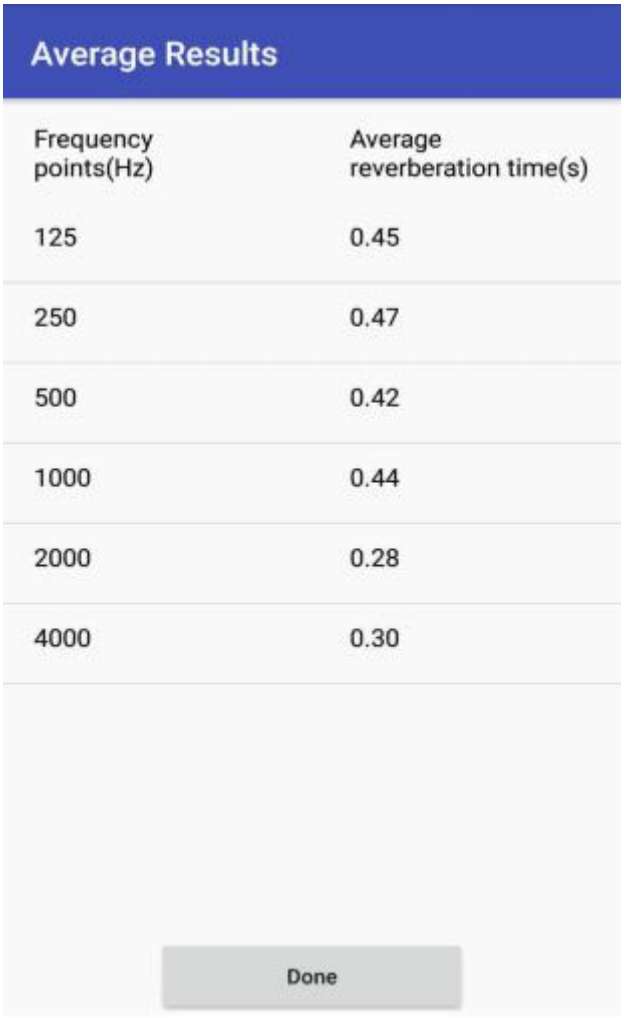


**Figure 7**: RT60 of Bedroom by Samsung Galaxy S7 Phone.

After all the measurement for one room is completed and all of these data will be stored in phone's storage, the users can repeat this measurement in other rooms and the system will record all of it. The result can be checked if user press 'View old' button in the operate interface and select data of any wanted room. Average RT60 of the selected room will be displayed at first and remain information of this room will be displayed after pressing 'Room Information in Detail' button. The function of 'Show Chart' is to display all measured RT60 data at each octave band and each measured position in the form of line chart in one figure.

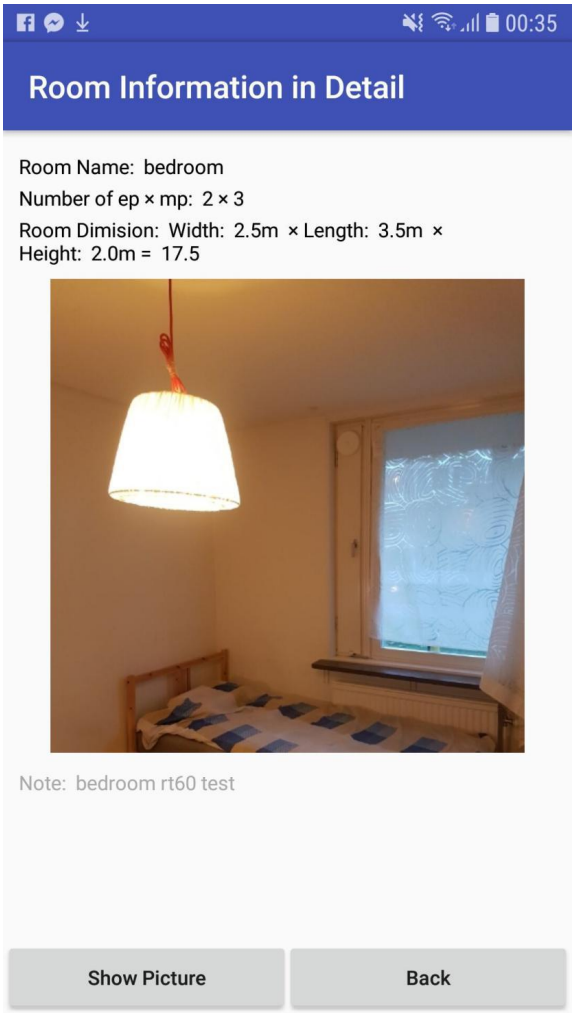


**Figure 8**: Function of 'Room Information in Detail' button.

All the measured RT60 data and room information can also be found as a word file named 'Exported data' with suffix as date and time when you measured this room, users can store, print or mail this file if it is needed.

All in all, this APP is accurate because users can measure several times at different positions in a room and then calculate the mean value, which will approach to the data precision instrument measured. On the other hand, this APP is very convenient because it can store all the information about the room, which is helpful to the researcher to make a decision.

# Chapter 3 Detailed Introduction about Working Process

In this session, we will introduce the core algorithm in this app in detail. The whole Java code part can be divided into two main phases: sound recording phase and data analysis phase. While working, the target Android device will record sound for five seconds firstly. Six groups of data about SPL in dB and time points are generated for the six wanted frequency bands as the output of one time's sound recording process. Then the code will analyze all the outcome data and finally figure out six RT60 results of the six octave frequency bands. The whole working process of this app is shown as following:

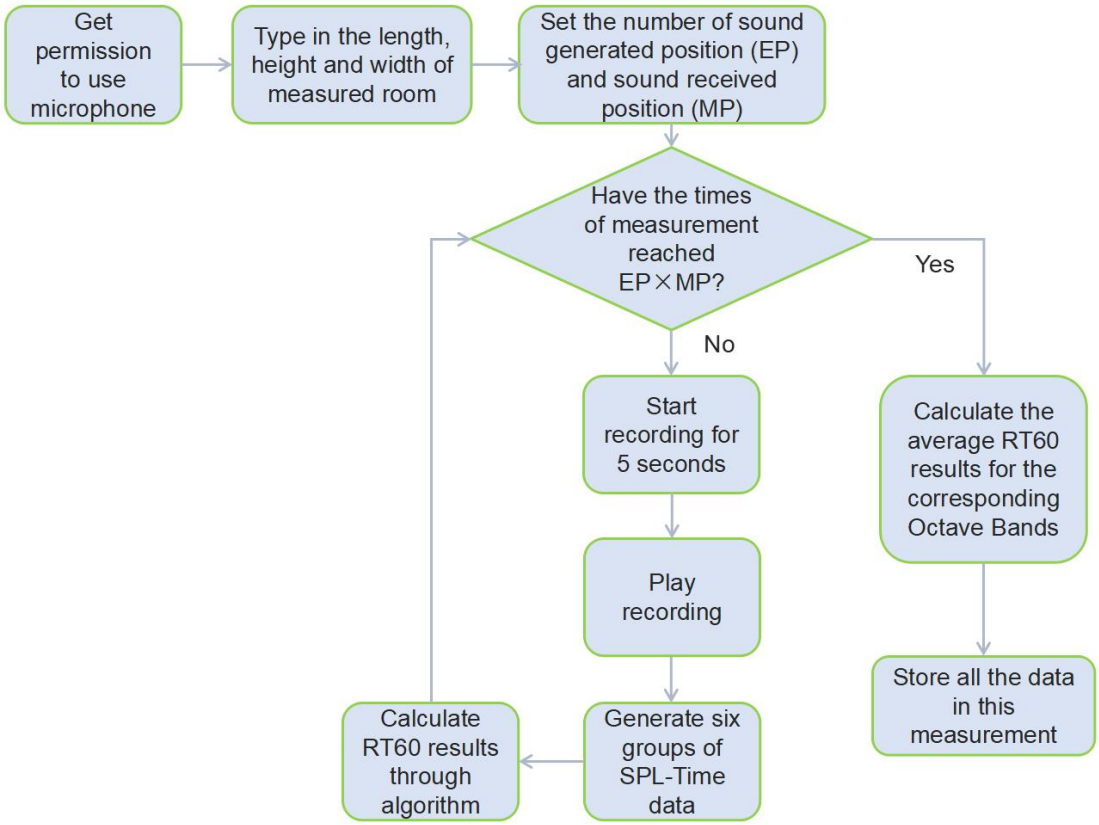


**Figure 9**: Flow chart of the whole measurement process of this app.

## 3.1 Programming Language and Development Tool

This project is developed based on the programming language of Java and the development tool is Android Studio 3.1.2.

Java is a high-level object oriented programming language. It is a derivation of C language, not only containing a lot of advantages of C, but also excluding some of the C's concepts which are hard to understand. So Java has powerful functions and is

simple to use at the same time. The main features of Java include object oriented, robustness, safety, platform independence, multi-threading, dynamics and so on.

Android Studio is a integrated development tool released by Google for developing and testing applications that are targeted at the Android platform. Compared with the old development tool 'Eclipse', Android Studio is more convenient at constructing program interface, prints more detailed and all round coding messages, has more powerful intelligent recognition function, generates smaller project items and provides a better managing system.

## 3.2 Sound Recording

' AudioRecord ' is a class provided by Android which is used as the method of recording sound in this app. To start recording, the target device will firstly send a request to the user for the permission to use its microphone. After getting the permission, a data stream is created for input data. Then set an array in order to meet the shortest length demand to save the byte data created in the sound recording process. The shortest length is determined through the method 'Audio Record get Min Buffer Size ', also there are some parameters such as sampling rate ( set as 44100 Hz in this project ) , recording channel and encoding method , etc. This array is the object of 'AudioRecord' and the data in it will finally be imported to the established data stream. After recording for five seconds, the record will end and the data stream will also be closed . The whole recording process is shown in the block plot below:

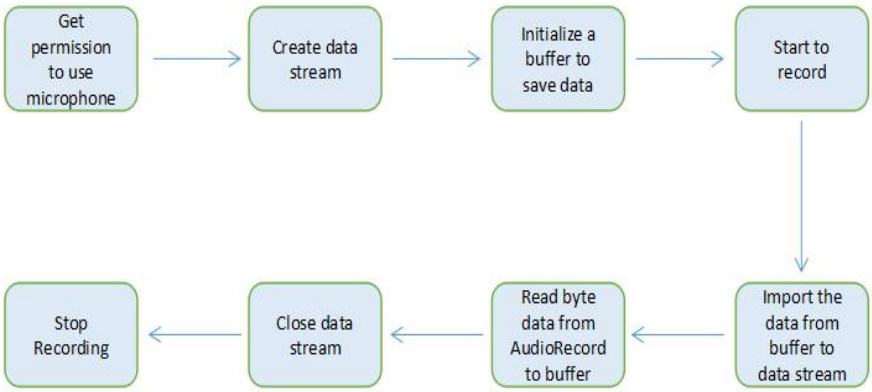


**Figure 10**: Flow chart of sound recording.

So as to increase the accuracy of the data gotten from sound recording, another two methods are used to reduce noise in measure environment.

### 3.2.1 Apply Noise Suppression

In order to reduce the influence from natural or contrived noise, applying noise suppression function is an important step. The Noise Suppression function in‘ Speex ’ is one of the improvement applied in this app.

‘Speex ’is a free, open-source audio compression format which is used for speeching and communicating. ‘Speex ’provides professional proprietary speech encoder. In addition, it is designed for Internet applications and contains many useful functions including Noise Suppression, Voice Activity Detection, Acoustic echo canceler and so on.

‘Speex’ uses a kind of ‘Short Time Spectral Amplitude-Minimum Mean-Square Error’ (STSA-MMSE) algorithm to achieve noise suppression. ‘STSA-MMSE’ is the optimal estimation under non-stationary condition, which is based on ‘Weiner Filtering’. ‘STSA-MMSE’ algorithm takes advantage of the insensitive characteristics of human ear to audio’s phase transformation to achieve short-time amplitude spectrum through estimation. After that, it uses the phase information of the audio with noise to finally compose the enhanced audio with the suppressed noise.

The codes for‘ Speex ’ in this app is written with the coding language of ‘ c++ ’ instead of Java and part of the code about the setting of noise suppression function is as follows:

```
SpeexPreprocessState *state = speex_preprocess_state_init(1024, SAMPLE_RATE);
int denoise = 1;
int noiseSuppress = -25;
speex_preprocess_ctl(state, SPEEX_PREPROCESS_SET_DENOISE, &denoise);
speex_preprocess_ctl(state, SPEEX_PREPROCESS_SET_NOISE_SUPPRESS, &noiseSuppress);
```

**Figure 11**: Core codes about noise suppression in ‘Speex’.

The parameter of ‘noiseSuppress’ in the plot above stands for the level of noise reduction and the value of -25 is the default value for this parameter, the smaller this parameter, the better the effect of noise reduction but at the same time the condition of audio’s distortion will be more serious, so it is important to keep a balance between these.

The difference between the accuracy of measurement without and with ‘Speex’ is shown in the following plots:

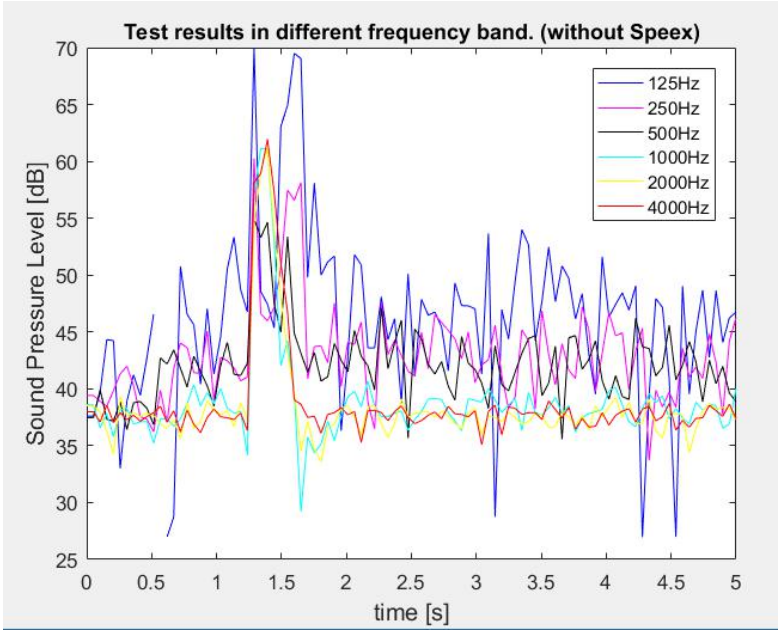


**Figure 12**: SPL-time plot (without Speex).

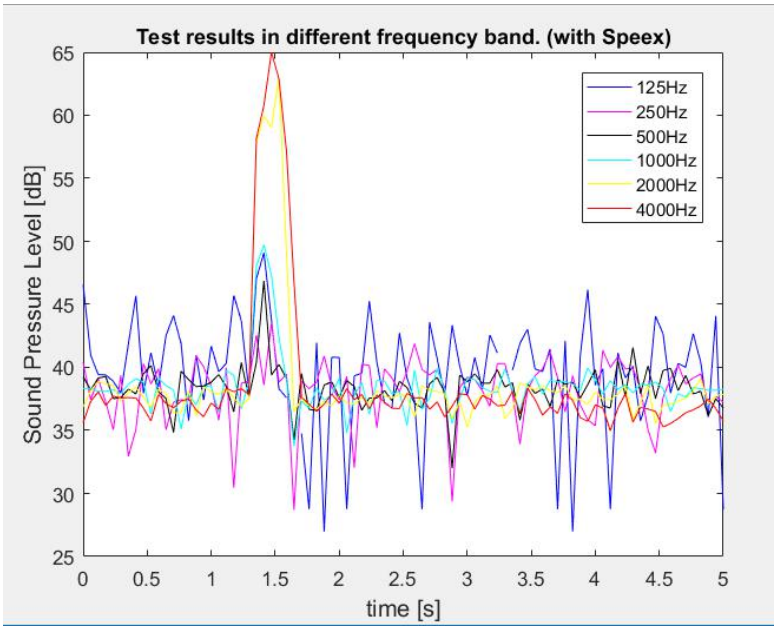


**Figure 13**: SPL-time plot (with Speex).

It can be seen that the effect of noise is reduced a lot and the impulse becomes more obvious in the plot when Speex is applied, especially in the frequency bands of 125 Hz, 250 Hz and 500 Hz. The noise suppression function of 'Speex' is proved to be helpful.

### 3.2.2 Select Audio Input Source

As a function of 'AudioRecord', it provides several options as the source of recorded sound for Android device. All the options and theirs corresponding audio sources are listed below:

| Option | Audio Source |
|---|---|
| CAMORDER | Microphone audio source tuned for video recording, with the same orientation as the camera if available |
| DEFAULT | Default audio source |
| MIC | Microphone audio source |
| REMOTE_SUBMIX | Audio source for a submix of audio streams to be presented remotely |
| UNPROCESSED | Microphone audio source tuned for unprocessed (raw) sound if available, behaves like DEFAULT otherwise. |
| VOICE_CALL | Voice call uplink+downlink audio source Capturing from VOICE_CALL source requirs the CAPTURE_AUDIO_OUTPUT permission |
| VOICE_COMMUNICATION | Microphone audio source tuned for voice communication such as VoIP |
| VOICE_DOWNLINK | Voice call downlink (Rx) audio source |
| VOICE_UPLINK | Voice call uplink (Tx) audio source |
| VOICE_RECOGNITION | Microphone audio source tuned for voice recognition |

**Table 2**: Audio source options.

According to some existed code examples, 'the microphone of target device (the option of 'MIC' in the table above) is used mostly as the audio input source when recording. However during the process of testing all the mentioned options, the option of 'Voice Communication' has the highest accuracy of the tested SPL-Time data. Switching to this mode is achieved by the code '*AUDIO_INPUT* = MediaRecorder.AudioSource.*VOICE_COMMUNICATION* '. Through the testing of the other options except for 'MIC' and 'VOICE_COMMUNICATION', the measurement accuracy was either too low or the app got collapsed with those options. Let's take a comparison between the testing accuracy of option 'MIC' and 'VOICE _COMMUNICATION' below:

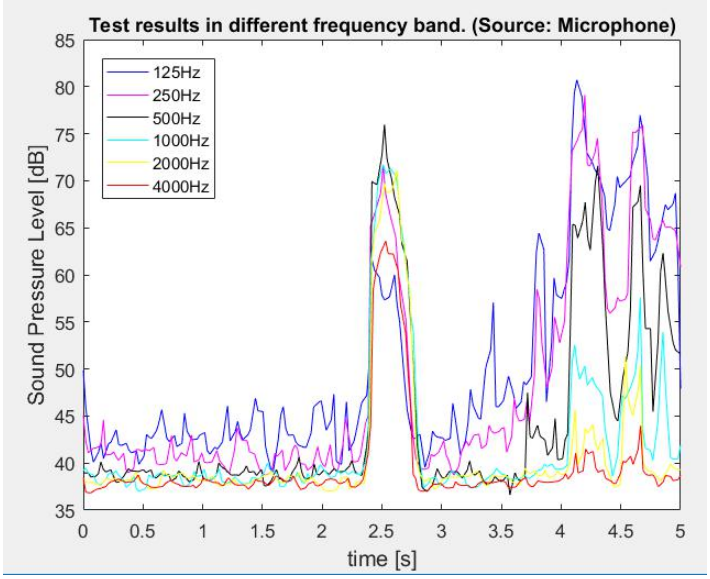


**Figure 14**: SPL-time plot (Microphone as source).

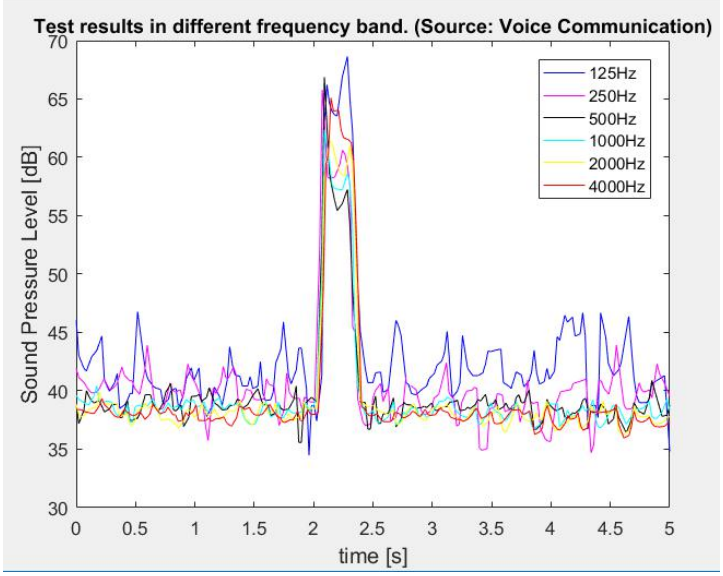


**Figure 15**: SPL-time plot (Voice Communication as source).

It can be seen that when 'Voice Communication' is set as input radio resource, it performs better in the measurement accuracy of SPL than 'Microphone'. The lower the Octave Bands, the more obvious this difference in accuracy will be. The reason why this app works better with the source of 'VOICE_COMMUNICATION' is that when this option is selected, based on the original functions of 'MIC', the built-in Acoustic Echo Canceler (AEC) and Automatic Gain Control (AGC) functions are also opened to improve the quality of recorded sound.

To sum up, two improvement are applied based on original sound recording function: 'Speex' and 'Voice Communication source'. As can be observed from Figure 8 to Figure 11, the accuracy improved a lot especially in the frequency bands of 125 Hz,

250 Hz and 500 Hz. This data is relatively accurate to be used in the subsequent data analysis part.

## 3.3 Data Acquisition

After recording sound successfully, the recorded sound will be played once. During this playing process, the whole five seconds' period is sampled at around two hundred time points. In this project, the sampling rate is set as 44100Hz. According to Nyquist-Shannon Criterion, the highest frequency of audio is 20000Hz and if the sampling frequency is set as twice of that highest frequency of audio, the original analog signals can be perfectly recovered. So 44100Hz is chose as the standard frequency in most conditions. When sampling frequency is set as 44100Hz, it means that 44100 samples will be taken in every second. In addition, every frame contains 1024 samples. So the length of time for every frame is 1024 divides 44100, which equals to around 23.22 ms. In this project, the recording period is 5 seconds and the number of frames is 5000 divides 23.22 which equals to around 215. In other words, each group of data about SPL and time contains around 215 SPL values and each SPL value has a corresponding time point and the time interval between every pair of time points is around 23.22 ms. The time interval of 23.22 ms is small enough compared with the whole 5 seconds recording period to describe the condition of SPL in the whole recording process well.

After taking samples, the next step is to divide the sampled sound into different frequency bands according to 'Octave Band Filter Standard ', then the SPL in dB will be calculated for all the sampled points in each frequency band. Finally, all the SPL values will be connected and ranked according to their corresponding time points, which are showed in the form of SPL-time plots in the figures from section 3.2.

Two steps to get the relation between Sound Press Level and time of each target frequency band will be introduced in detail.

### 3.3.1 Frequency Band Separation

At each sampled time point, to preliminarily get the frequency domain information of the recorded sound, a 'Visualizer' is set. 'Visualizer' is a class of Java and it is used for the visualization of audio data in time domain. The length of the data it gets for once, in other words, sampling length is set to be 1024 bytes( the max value ) which means that for one sampled time point, an array of 1024 bytes in time domain is gotten from this ' Visualizer '.

Then a function called 'on FFT Data Capture' is used to do 'Fast Fourier Transformation' and get the data in frequency domain. This function contains three input parameters:

```
@Override
public void onFftDataCapture(Visualizer visualizer, byte[] fft, int sampling_rate) {
```

**Figure 16**: Code of 'onFftDataCapture' function.

The first parameter stands for the 'Visualizer' mentioned above, the second parameter 'FFT' is the name of the output array of this function, the frequency domain data transformed from those of time domain is saved in the array called 'FFT', also the length of it is 1024 (set as above). The last parameter is the sampling rate of 'Fast Fourier Transformation' , which is set to be 44100 Hz in this project.

After using the function of 'onFftDataCapture ' , an array of 1024 bytes is obtained . It contains 512 complex values , 512 real parts and 512 imaginary parts , which are uniformly distributed between 0 to 44100 Hz.

However , according to Equation 4 below :

$$Frequency\ Spectrum\ \text{Re}solution = \frac{Sampling\ Frequency}{Number\ of\ Samples} = \frac{44100}{512} \approx 86.13\ Hz$$

**Equation 4**: Calculation of Sampling Interval (before increasing frequency spectrum resolution).

This means that in frequency domain , from 0 to 44100 Hz, samples are taken every 86.13 Hz. However, according to 'Octave Band Filter Standard', lower frequency bands have narrower bandwidth. For example, in the band with center frequency of 125 Hz (88 to 176 Hz), only one sample is located in this band, which effects accuracy of measurement, especially for lower frequency bands. In other words, frequency spectrum resolution is not enough.

To increase frequency spectrum resolution, the following steps are applied in this project:

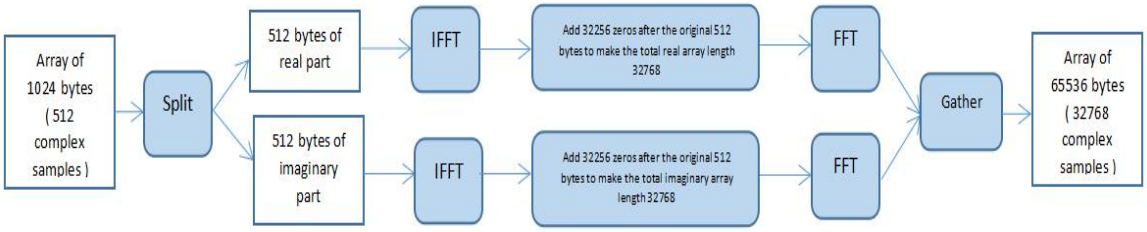


**Figure 17**: Steps to increase frequency spectrum resolution.

In the steps introduced above, several zeros were added after the original 512 real / imaginary bytes to make the total length of real / imaginary array to be 32768. According to the theory of 'FFT', the length of the array to do 'FFT' must be the power of two and if this length is too long, it will affect the whole time to do 'FFT'.

That is why the value of 32768 is selected, improving frequency spectrum resolution and effecting little on processing time at the same time.

The array length is increased to 65536 ( with 32768 complex elements ), the frequency spectrum resolution becomes:

$$Frequency\ \ Spectrum\ \ \mathrm{Re}\, soultion = \frac{Sampling\ \ Frequency}{Number\ of\ Samples} = \frac{44100}{32768} \approx 1.345\, Hz$$

**Equation 5**: Calculation of Sampling Interval (after increasing frequency spectrum resolution).

That is to say, samples are picked around every 1.345 Hz uniformly from 0 to 44100 Hz. Now even the narrowest band of 125 Hz (88 to 176 Hz) has relatively enough samples and frequency band separation will be done next.

According to 'Octave Band Filter Standard ', the starting and end samples as well as their corresponding frequency points for the six target frequency bands as calculated as following:

| Octave Band | Start Sample ( No.) | End Sample ( No.) | Start Frequency ( Hz ) | End Frequency ( Hz ) |
|---|---|---|---|---|
| 125 Hz | 67 | 131 | 88.770 | 174.850 |
| 250 Hz | 132 | 263 | 176.195 | 352.390 |
| 500 Hz | 264 | 526 | 353.735 | 706.125 |
| 1000 Hz | 527 | 1052 | 707.470 | 1413.595 |
| 2000 Hz | 1053 | 2101 | 1414.940 | 2824.500 |
| 4000 Hz | 2102 | 4201 | 2825.845 | 5649.000 |

**Table 3**: Start and end samples for each Octave Band in this project.

The numbers in the table above is calculated according to:

$$Frequency = (Number\ of\ Sample - 1) \times Frequency\ \ Spectrum\ \ \mathrm{Re}\, solution$$

**Equation 6**: Calculation formula of specific frequency value for each sample.

In the complex number array of 65536 sampling time point, the elements are picked due to Table 2 and they are divided into six groups (corresponding to the six target Octave Bands). Then each group is sent into the SPL calculation function to get the value in dB as their own Octave Band for each time point.

### 3.3.2 Calculate Sound Pressure Level

So as to calculate SPL value for each target group, the storage order of the bytes in each group has to be changed to 'Little Endian 'order firstly. 'Little Endian' stands for an order that the least significant byte is addressed firstly and all the bytes are arranged in increasing significance order. The highest address has the most significant byte ( the last ).

Then, the SPL values are calculated according to:

$$L_p = 20 \times \log_{10}(\frac{p_{rms}}{p_{ref}}) \qquad Unit: dB$$

**Equation 7**: Calculation formula of sound's decibel value.

In the theoretical equation above, the parameter '$P_{rms}$' is the measured SPL. The parameter '$P_{ref}$' is a referenced SPL and theoretically it is set according to the strength of background sound in the testing environment, for example, 20 micro Pascals, which is the smallest SPL that human ear can hear.

However, in the coding of this app, the quantity provided by the senors of Android devices is sound field's amplitude. So in the code, the '$P_{rms}$' in the equation above is replaced by the average amplitude calculated based on the amplitudes of elements in each byte group and the parameter of '$P_{ref}$' is replaced by the number of one, which is the smallest amplitude value that Android devices can 'hear'. In addition, each SPL for Android devices in this project at each time point is calculated based on all the byte elements divided according to Octave Band Filter mentioned before. After that, for one measurement, six groups of data about SPL for Android devices-Time points are formed to be resolved to get the six RT60 results.

So far, for each sampled time point, six SPL values of the target Octave Bands have been calculated out. For one Octave Band, connecting all its SPL values in the order of time points, we can get a SPL -Time relation of that band and there are six of this kind of relations in total. This kind of relations are the key gist to calculate RT60 values of those target Octave Bands.

## 3.4 Data Analysis

In this session, the algorithm to calculate RT60 will be introduced in detail. The whole algorithm is based on the generated six groups of data about SPL-Time, using these groups of data to calculate the six RT60 results for the six target Octave Bands.

### 3.4.1 Find the highest Sound Pressure Level point

After obtaining all the six groups of data about Sound Press Level –Time, the point with the highest SPL value will be found firstly.

The following plot is an example coming from a measurement for 4000Hz band:

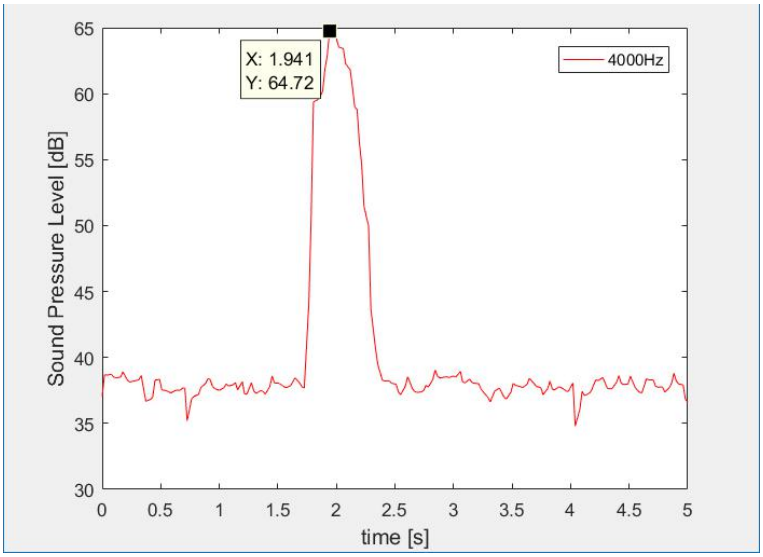


**Figure 18**: Maximum SPL point in 4000 Hz from the example measurement.

The max SPL point in 4000 Hz group is firstly fixed to be 64.72 dB, at 1.941 s as shown in the plot above.

### 3.4.2 Fix the starting point of linear decay

The core of this algorithm is to determine a linear decay line and use the negative value of its slope to calculate RT60.

The starting point of this determined linear decay line is fixed as the point, the gap in SPL between which and the formerly fixed highest point is the closest to 7dB. In the standards, this gap is set to be 5dB and the reason why it is increased in this project is for the consideration of setting the determined starting point a little bit lower than the true starting point of linear decay in SPL value and increasing the possibility of completely skipping the echo period.

The starting point of decay from this example is fixed to be 56.31 dB, at 2.196 s, as shown below:

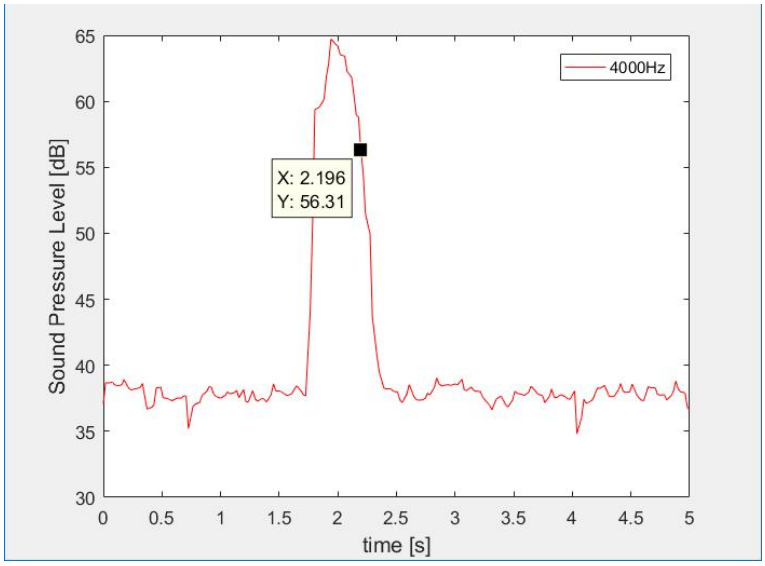


**Figure 19**: The starting point of determined linear decay line in 4000 Hz from the example measurement.

### 3.4.3 Fix the ending point of linear decay

Afterwards, the lowest point from maximum SPL point and the end of array is fixed. As shown below, it is 34.81 dB, at 4.039 s.

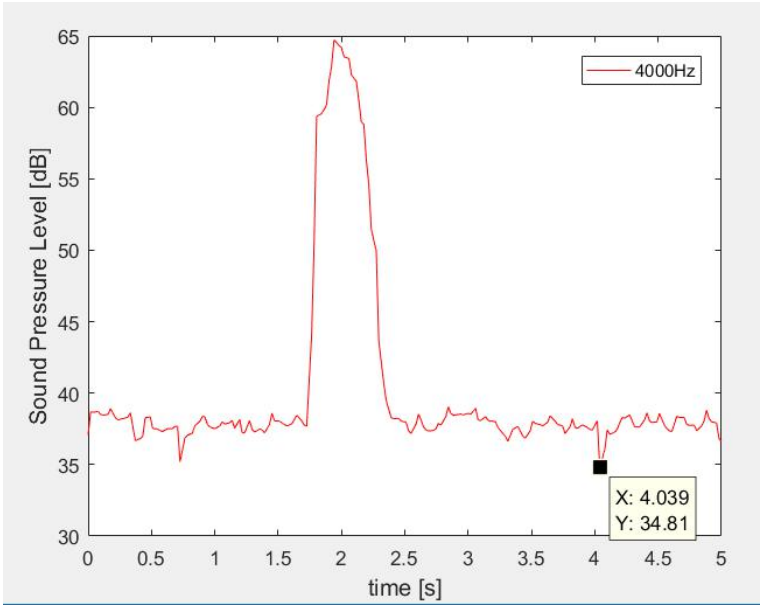


**Figure 20**: The lowest SPL point in 4000 Hz from the example measurement.

Then searched from the lowest point to the determined starting point, the point whose gap in SPL is the closest to 12 dB from the lowest point is fixed as the ending point of the determined linear decay line. The reason why the threshold value is taken to be twelve instead of twenty as described in standards is for the consideration that the sound impulse for this app is the normal forms of impulse in daily life, such as hand clapping, and sometimes they can not be as loud as the impulse generated by the specific equipment in professional acoustic testing rooms, which means that if this 'headroom' value is set too large, the ending point will be wrongly determined too close to the determined starting point, even sometimes above the determined starting point in the algorithm of this app, which is obviously wrong. Also according to the testing experience during developing process, it was found that will this value being 12 dB, the determined ending point is more likely to be close to the true ending point, describing the linear decay process better. Another principle added in the algorithm to avoid the situation that the determined starting point and ending point are too close to each other is that if the difference of SPL values between the determined starting point of decay and the ending point fixed by 12 dB on the lowest point is less than 10 dB, which also means the determined decay process is too short to be typical, the end point will be determined again by 8 dB on the lowest SPL. Both of the value of 12 dB and 8 dB is carefully selected based on the huge number of testing results and are considered to be most suitable for this app. The ending point in the example measurement is finally fixed at 42.21 dB, at 2.314 s, 8 dB larger than the lowest value in the example measurement as showed below:

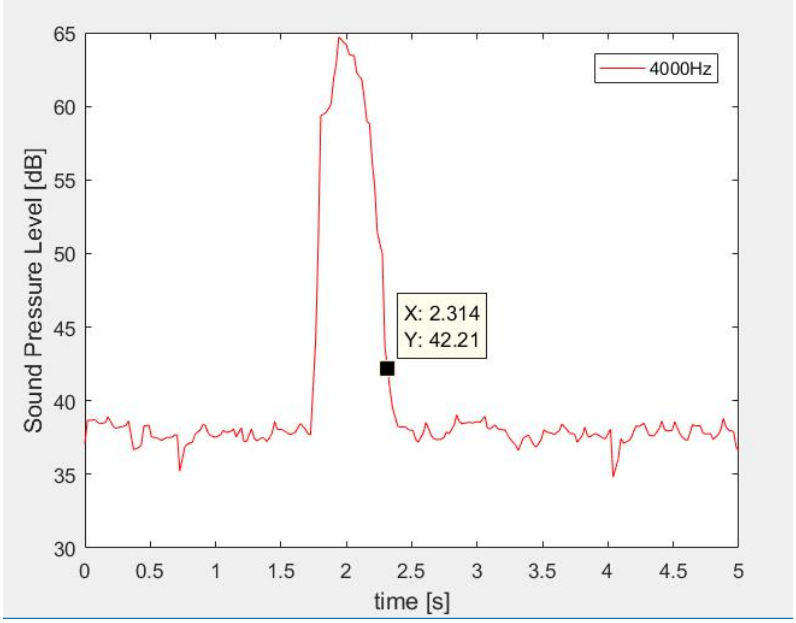


**Figure 21**: The ending point of determined linear decay line in 4000 Hz from the example measurement.

### 3.4.4 Calculate RT60 results

So far, both of the determined starting points and ending points have been fixed. The final step in calculation is to use the slope of the connection line between the estimated starting point and estimated ending point of linear decay process to get RT60 value. The formula and result are shown below:

$$RT_{60} = \frac{60}{-\frac{L_{dB\_start} - L_{dB\_end}}{Time_{start} - Time_{end}}} = \frac{60}{-\frac{56.31 - 42.21}{2.196 - 2.314}} \approx 0.502\,s$$

**Equation 8**: Calculation formula of RT60 (with slope) in this project.

The reason for using the slope to calculate RT60 instead of directly calculate the time difference of -20 dB is that this app is designed to ordinary clap measuring, not the professional high pitch equipment , the SPL of clap is sometimes not loud enough as the situation shown in the standards and the difference of SPL in linear decay process is often less than 20 dB. Trying to ensure measurement accuracy, we decide to use the slope between estimated starting point and ending point of the linear decay process to calculate RT60.

# Chapter 4 Data Comparison

This chapter mainly makes a data comparison between app runs at different android phones and precision instrument named Nor140. At first we need to point out that when we come across some RT60 that is not so close to the official data unfortunately, we have to repeat the measurement again. The reason is that there are many limitations while measuring: such as phone hardware limitations (microphone, loudspeaker etc.), different sound press level with different gesture of hands clapping, noise (environmental noise, white & pink noise etc.) and so on. These limitations are also working on Nor140 but less than Android phones.

## 4.1 Nor140

Norwegian company Norsonic, founded in 1967, is one of the word's leading manufactures of high quality acoustic and vibration measuring equipment. And Nor140 noise monitoring system has the advantage of simple operation, easy to use and large dynamic range. The minimum resolution of the SPL versus time analysis of Nor140 sound level meter is 50 milliseconds. From 1/1 or 1/3 octave band analysis in the range of 0.4Hz to 20KHz and 9.6KHz FFT analysis can be achieved, ICP type vibration sensor is directly connected to the vibration signal. It suits ISO 10052, ISO 140, ISO 717 standard building acoustic measurement, and compiles with IEC and ANSI level 1 sound level meter [17].
In the following measurements, Nor140 has been calibrated before measurements.


**Figure 22**: Nor140.

## 4.2 RT60 comparison

We made a RT60 data comparison between two android phone(Samsung Galaxy S7, Huawei Mate9 Pro), one android virtual phone with Nor140 in two rooms(Specific RT60 measurement room, normal bedroom).

### 4.2.1 Measurement rooms

We chose two rooms as our RT60 measurement room, one was specially designed with huge number of tiles covering the wall around it for great reverberation effect. It is located in the basement of V-building of Lund University and it is also the receiving room of a step-sound insulation lab at the same time. Another room for test was a bedroom in an ordinary house. Here are the figures for these two rooms.

**Figure 23**: Specific RT60 measurement room.

From figure 18 we can see that ceramic tiles are all cover on the room walls, so the average sound absorption coefficient is very low and the size of this room is 5.1×5.8×3.2. According to Sabine formula mentioned before, RT60=0.161×V/(S×a), so there will be a longer RT60 in this room and it is a good place to test, we assume it as room 1.

**Figure 24**: Normal bedroom.

Figure 19 shows a normal bedroom and the size of it is 2.5×3.5×2, which makes the speaker's clarity into consideration, the sound absorption coefficient of this room should be high enough. What's more, there are many objects in this room so absorption coefficient will be higher. But we can also test RT60 and check how is going on about this value, and we assume this room as room 2.

### 4.2.2 RT60 for two rooms by Nor140

Here is the result that we measured 6 times(2×3) in each room by Nor140.

| Nor140 | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 1.44 | 1.89 | 2.27 | 1.95 | 1.6 | 1.15 |
| NO.2 | 1.31 | 1.54 | 2.17 | 1.77 | 1.39 | 1.14 |
| NO.3 | 1.6 | 2.01 | 2.11 | 1.64 | 1.49 | 1.03 |
| NO.4 | 1.5 | 1.88 | 2 | 1.8 | 1.3 | 1.03 |
| NO.5 | 1.2 | 1.45 | 2.43 | 1.87 | 1.52 | 1.06 |
| NO.6 | 2.74 | 1.93 | 2.45 | 1.82 | 1.57 | 1.04 |
| Mean | 1.632 | 1.783 | 2.238 | 1.808 | 1.478 | 1.075 |
| StdDev. | 0.561 | 0.230 | 0.179 | 0.095 | 0.114 | 0.055 |

**Table 4**: RT60 for Room 1 by Nor140.

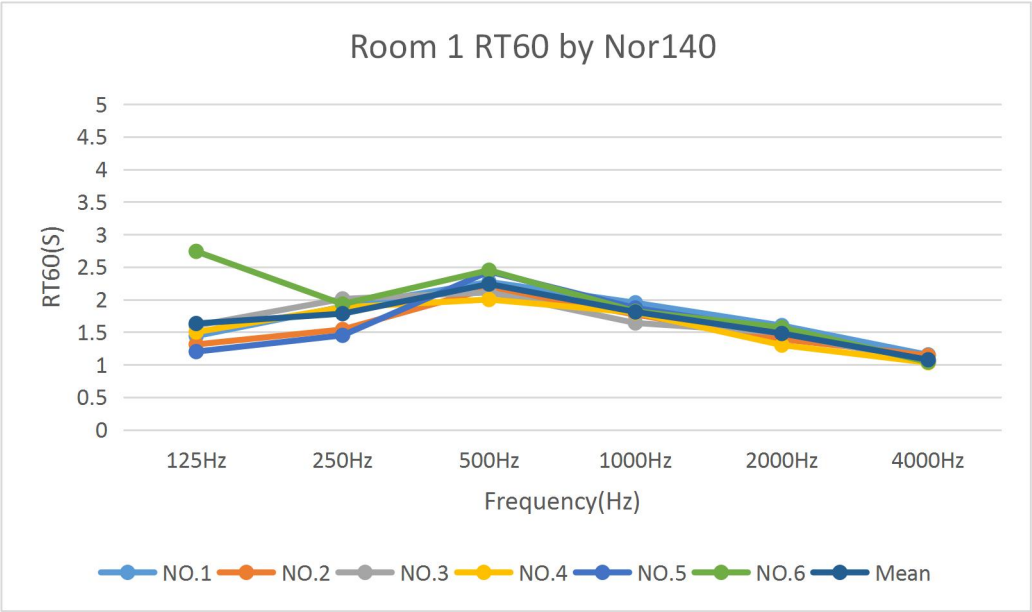


**Figure 25**: RT60 comparison in Room 1 by Nor140.

| Nor140 | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 0.52 | 0.41 | 0.41 | 0.31 | 0.31 | 0.34 |
| NO.2 | 0.3 | 0.31 | 0.44 | 0.31 | 0.34 | 0.34 |
| NO.3 | 0.52 | 0.43 | 0.45 | 0.64 | 0.41 | 0.38 |
| NO.4 | 0.47 | 0.37 | 0.44 | 0.4 | 0.42 | 0.36 |
| NO.5 | 0.52 | 0.41 | 0.44 | 0.3 | 0.37 | 0.33 |
| NO.6 | 0.5 | 0.42 | 0.39 | 0.34 | 0.37 | 0.35 |
| Mean | 0.472 | 0.392 | 0.428 | 0.383 | 0.37 | 0.35 |
| StdDev. | 0.086 | 0.045 | 0.023 | 0.131 | 0.041 | 0.018 |

**Table 5**: RT60 for Room 2 by Nor140.

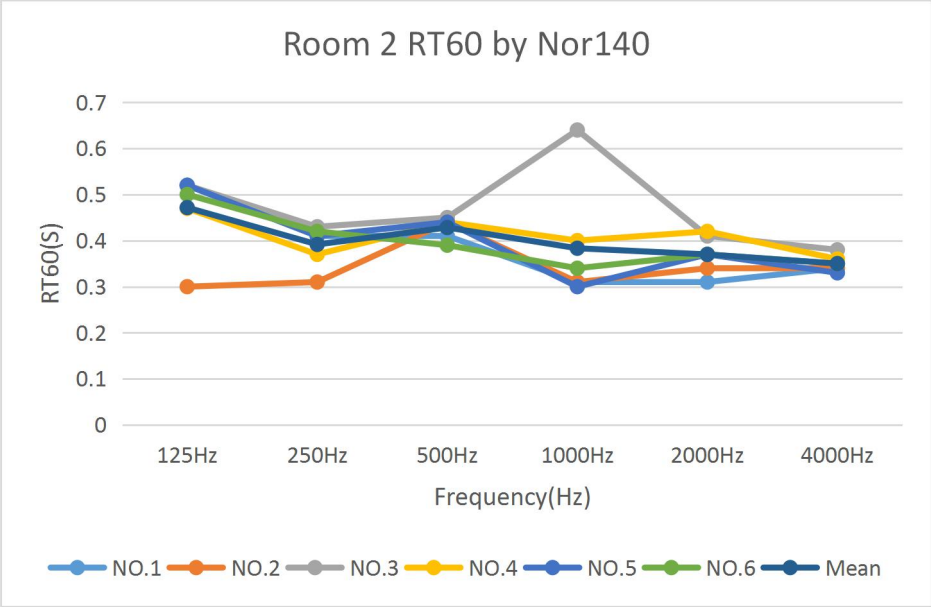


**Figure 26**: RT60 comparison in Room 2 by Nor140.

From these two figures we can find out that the RT60 value of room 1 is much higher than that of room 2 and higher RT60 value definitely means longer echo period. On the other hand, it presents how important about building acoustic design.

### 4.2.3 RT60 for two rooms by virtual phone on laptop

Android virtual phone is created by Android studio coding system, it has all functions as the normal cellphone. Virtual phone is a good sample for programmers debug and test APP. In the virtual phone, our app is runned and the teast results of each room are showed as following:

| Virtual phone | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 1.91 | 1.698 | 2.072 | 1.747 | 1.56 | 0.847 |
| NO.2 | 1.161 | 1.999 | 2.556 | 1.847 | 1.108 | 1.121 |
| NO.3 | 2.094 | 2.045 | 1.909 | 1.748 | 1.059 | 1.368 |
| NO.4 | 1.923 | 2.101 | 1.785 | 1.708 | 1.738 | 1.187 |
| NO.5 | 2.015 | 1.327 | 2.076 | 1.672 | 1.849 | 1.264 |
| NO.6 | 1.772 | 1.841 | 2.086 | 1.323 | 1.506 | 1.617 |
| Mean | 1.813 | 1.835 | 2.081 | 1.674 | 1.47 | 1.234 |
| StdDev. | 0.337 | 0.289 | 0.262 | 0.182 | 0.324 | 0.257 |

**Table 6**: RT60 for Room 1 by Virtual phone.

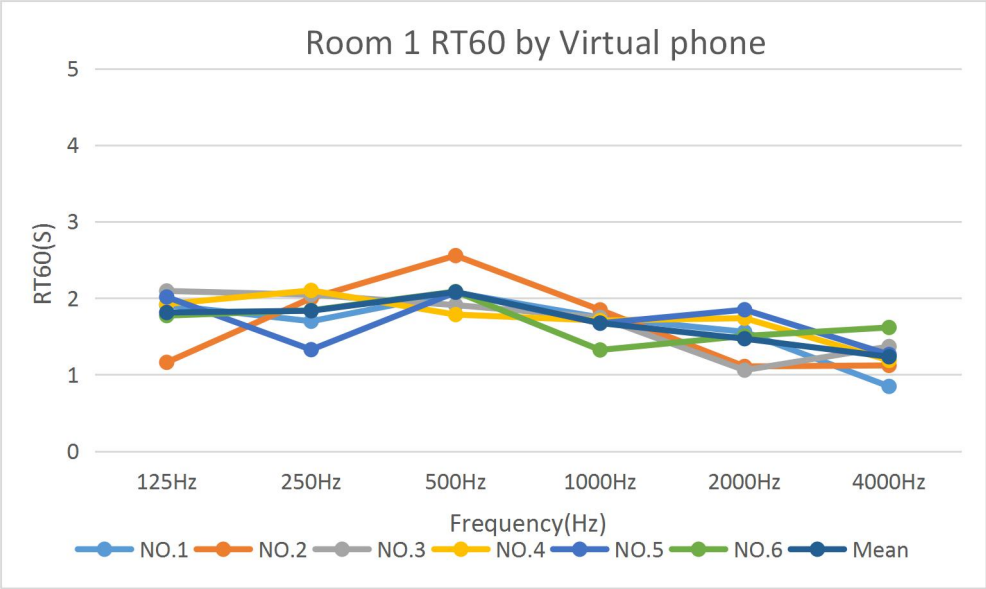


**Figure 27**: RT60 comparison in Room 1 by Virtual phone.

| Virtual phone | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 0.502 | 0.456 | 0.442 | 0.304 | 0.333 | 0.401 |
| NO.2 | 0.373 | 0.453 | 0.39 | 0.431 | 0.343 | 0.28 |
| NO.3 | 0.402 | 0.394 | 0.353 | 0.384 | 0.343 | 0.393 |
| NO.4 | 0.443 | 0.343 | 0.421 | 0.363 | 0.374 | 0.321 |
| NO.5 | 0.344 | 0.444 | 0.445 | 0.314 | 0.372 | 0.3 |
| NO.6 | 0.382 | 0.385 | 0.453 | 0.405 | 0.345 | 0.383 |
| Mean | 0.408 | 0.413 | 0.417 | 0.367 | 0.352 | 0.346 |
| StdDev. | 0.057 | 0.046 | 0.039 | 0.050 | 0.017 | 0.052 |

**Table 7**: RT60 for Room 2 by Virtual phone.

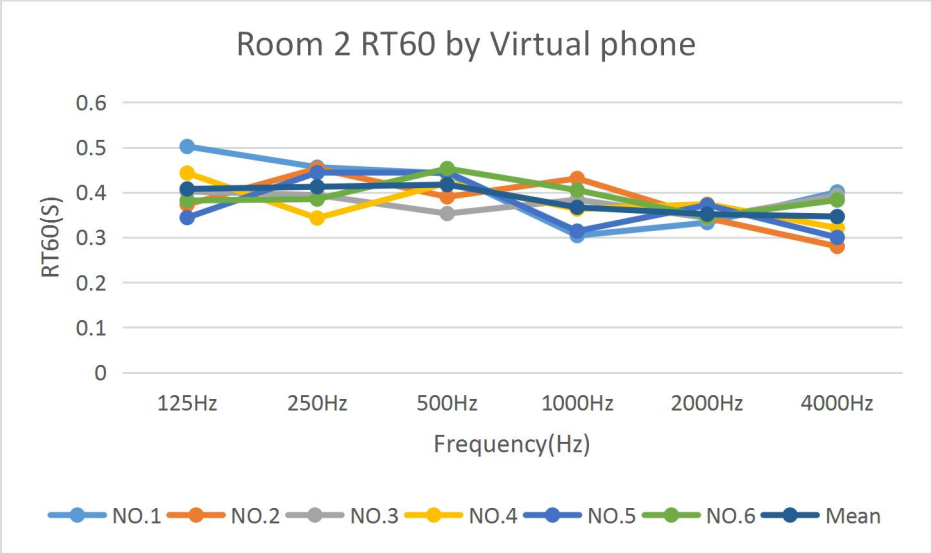


**Figure 28**: RT60 comparison in Room 2 by Virtual phone.

Compared with Nor140, the RT60 value we got by virtual phone is slightly different and this difference in result is acceptable.

### 4.2.4 RT60 for two rooms by two android phones

We tested other two different Android phones (Samsung Galaxy S7, Huawei Mate9 Pro) in order to confirm that our app works well.

| Samsung Galaxy S7 | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 1.546 | 1.427 | 2.065 | 1.645 | 1.496 | 1.109 |
| NO.2 | 1.474 | 1.395 | 1.661 | 1.801 | 1.74 | 1.063 |
| NO.3 | 1.558 | 1.353 | 1.914 | 1.422 | 1.596 | 1.04 |
| NO.4 | 1.753 | 1.467 | 1.916 | 1.886 | 1.64 | 0.963 |
| NO.5 | 1.588 | 1.323 | 1.637 | 1.694 | 1.87 | 0.829 |
| NO.6 | 1.716 | 1.565 | 1.891 | 1.531 | 1.379 | 0.786 |
| Mean | 1.606 | 1.422 | 1.847 | 1.663 | 1.620 | 0.965 |
| StdDev. | 0.107 | 0.087 | 0.166 | 0.170 | 0.174 | 0.131 |

**Table 8**: RT60 for Room 1 by Samsung.

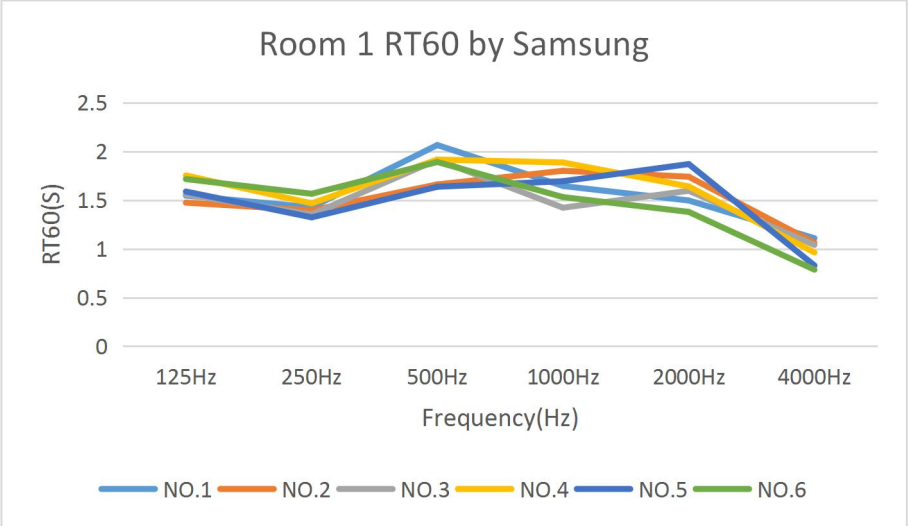


**Figure 29**: RT60 comparison in Room 1 by Samsung.

| Samsung Galaxy S7 | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 0.5 | 0.45 | 0.43 | 0.373 | 0.334 | 0.381 |
| NO.2 | 0.432 | 0.382 | 0.388 | 0.333 | 0.323 | 0.354 |
| NO.3 | 0.399 | 0.389 | 0.448 | 0.315 | 0.294 | 0.383 |
| NO.4 | 0.447 | 0.379 | 0.455 | 0.339 | 0.301 | 0.293 |
| NO.5 | 0.412 | 0.412 | 0.375 | 0.328 | 0.321 | 0.324 |
| NO.6 | 0.384 | 0.424 | 0.462 | 0.415 | 0.358 | 0.372 |
| Mean | 0.429 | 0.406 | 0.426 | 0.351 | 0.322 | 0.351 |
| StdDev. | 0.041 | 0.028 | 0.037 | 0.037 | 0.023 | 0.036 |

**Table 9**: RT60 for Room 2 by Samsung.

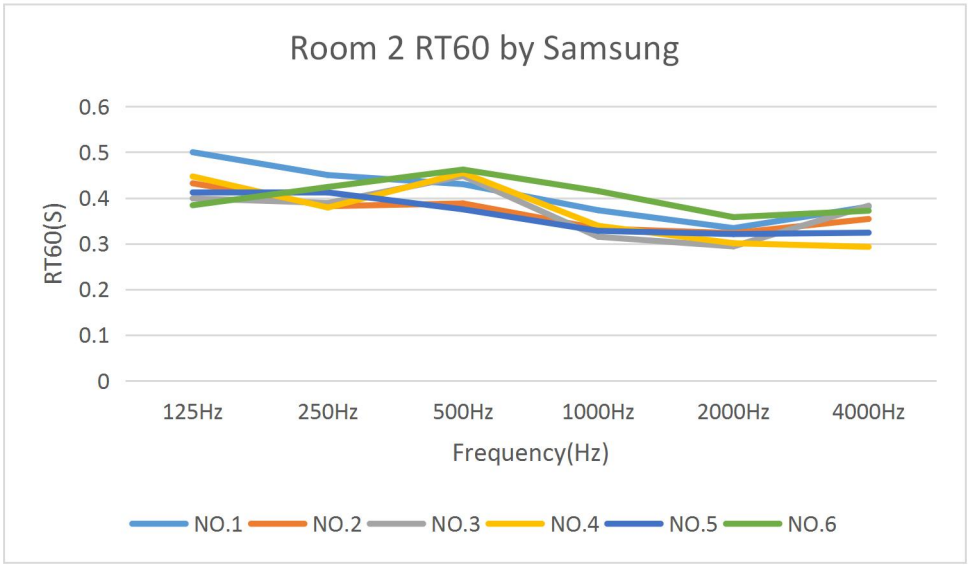


**Figure 30**: RT60 comparison in Room 2 by Samsung.

As the result by Samsung phone measuring, mean value of RT60 is also slightly difference with NOR140, and the standard deviation is not so large. Now let us look at how the app runs by Huawei.

| Huawei Mate9 Pro | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 1.883 | 1.921 | 1.933 | 1.729 | 1.811 | 0.795 |
| NO.2 | 1.998 | 2.009 | 1.789 | 1.817 | 1.387 | 1.384 |
| NO.3 | 1.748 | 1.752 | 1.933 | 1.952 | 1.526 | 0.898 |
| NO.4 | 1.961 | 1.981 | 2.192 | 1.842 | 1.595 | 0.998 |
| NO.5 | 2.086 | 1.953 | 2.247 | 1.582 | 1.437 | 1.239 |
| NO.6 | 2.165 | 1.865 | 1.856 | 1.621 | 1.685 | 0.945 |
| Mean | 1.974 | 1.914 | 1.992 | 1.757 | 1.574 | 1.043 |
| StdDev. | 0.148 | 0.094 | 0.185 | 0.144 | 0.158 | 0.223 |

**Table 10**: RT60 for Room 1 by Huawei.

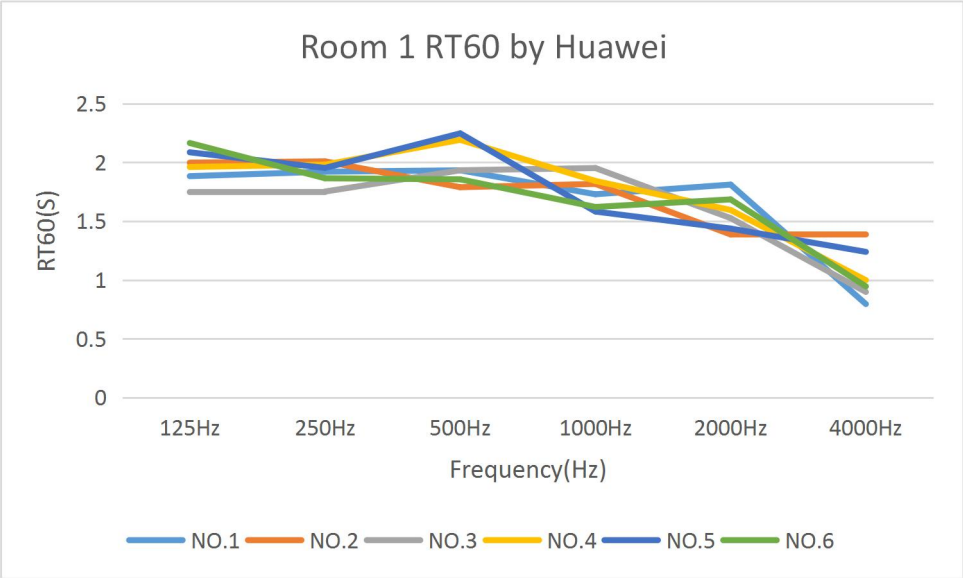


**Figure 31**: RT60 comparison in Room 1 by Huawei.

| Huawei Mate9 Pro | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 0.399 | 0.412 | 0.418 | 0.41 | 0.39 | 0.374 |
| NO.2 | 0.379 | 0.432 | 0.472 | 0.423 | 0.373 | 0.33 |
| NO.3 | 0.485 | 0.375 | 0.352 | 0.351 | 0.427 | 0.364 |
| NO.4 | 0.477 | 0.335 | 0.394 | 0.372 | 0.445 | 0.392 |
| NO.5 | 0.386 | 0.451 | 0.427 | 0.401 | 0.388 | 0.402 |
| NO.6 | 0.453 | 0.377 | 0.396 | 0.414 | 0.369 | 0.36 |
| Mean | 0.430 | 0.397 | 0.410 | 0.395 | 0.399 | 0.370 |
| StdDev. | 0.047 | 0.043 | 0.040 | 0.028 | 0.030 | 0.026 |

**Table 11**: RT60 for Room 2 by Huawei.

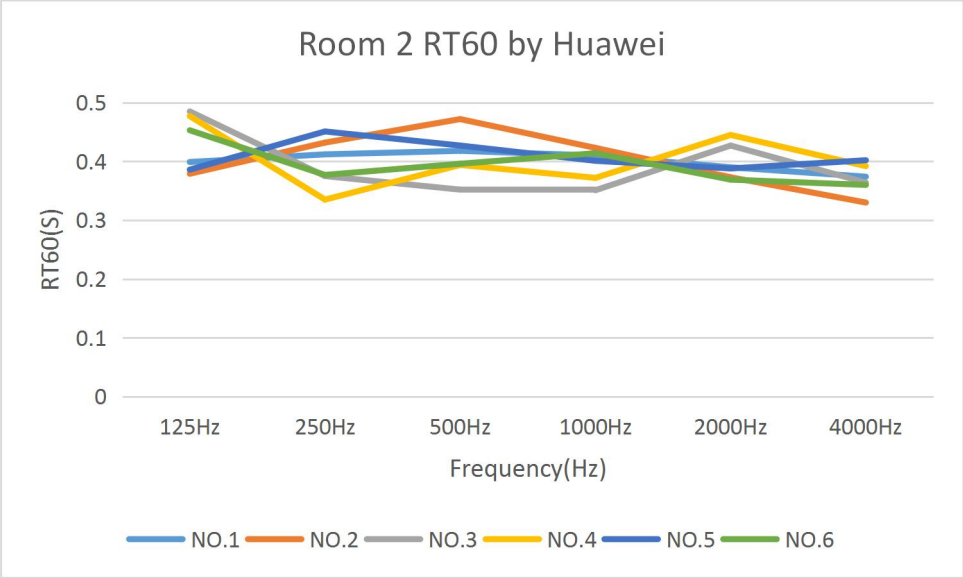


**Figure 32**: RT60 comparison in Room 2 by Huawei.

Comparing these results, it can be easily found out that the measurement results at same test rooms for different Android phone are different. The main reasons causing this phenomenon are external factors such as hardware and noise as we have discussed, and another very important factor is the difference of every time's hands clapping such as guesture and strength.

## 4.3 Comparison by each test equipment in each room

Now let us combine all the mean RT60 values of these equipment in each room to verify our app works fine.

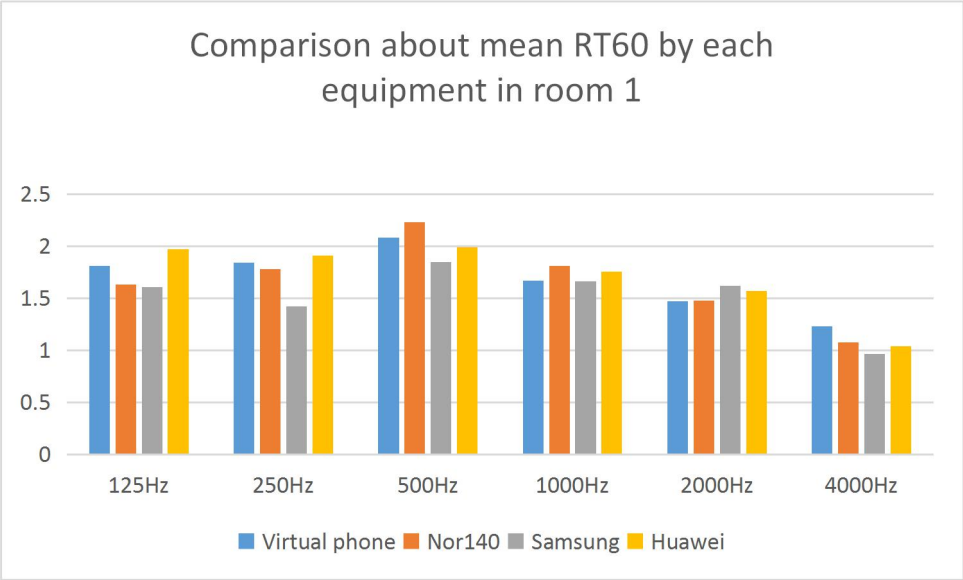


**Figure 33**: Comparison about mean RT60 by each equipment in room 1.

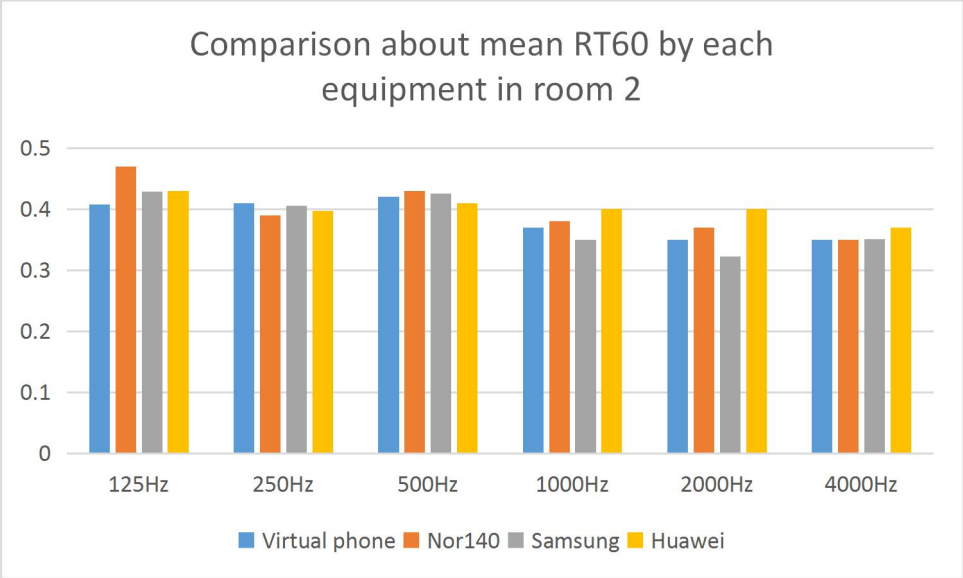


**Figure 34**: Comparison about mean RT60 by each equipment in room 2.

We think we have tried our best to reduce the external influences and the app in each phone works fine from figure 28 and 29.

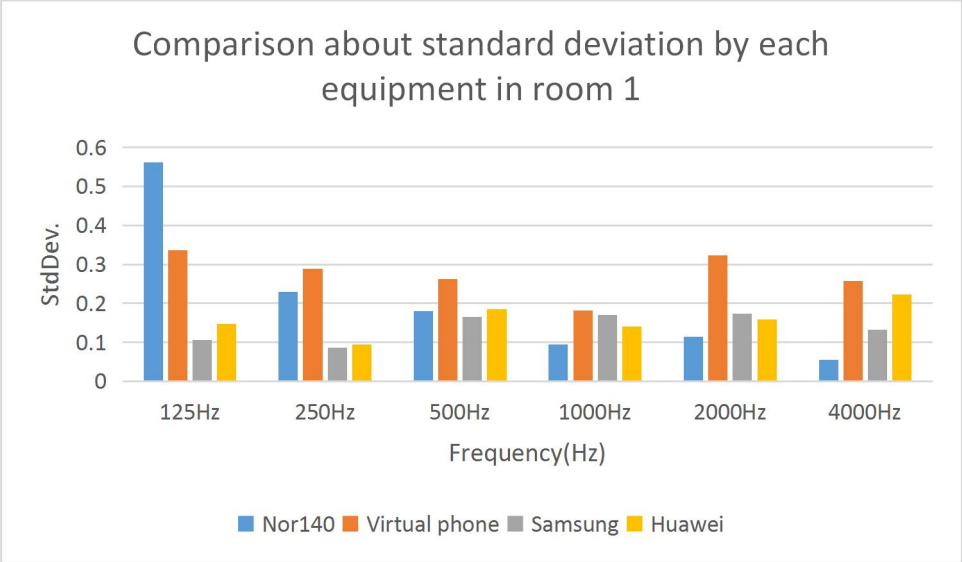


**Figure 35**: Comparison about standard deviation in room 1.

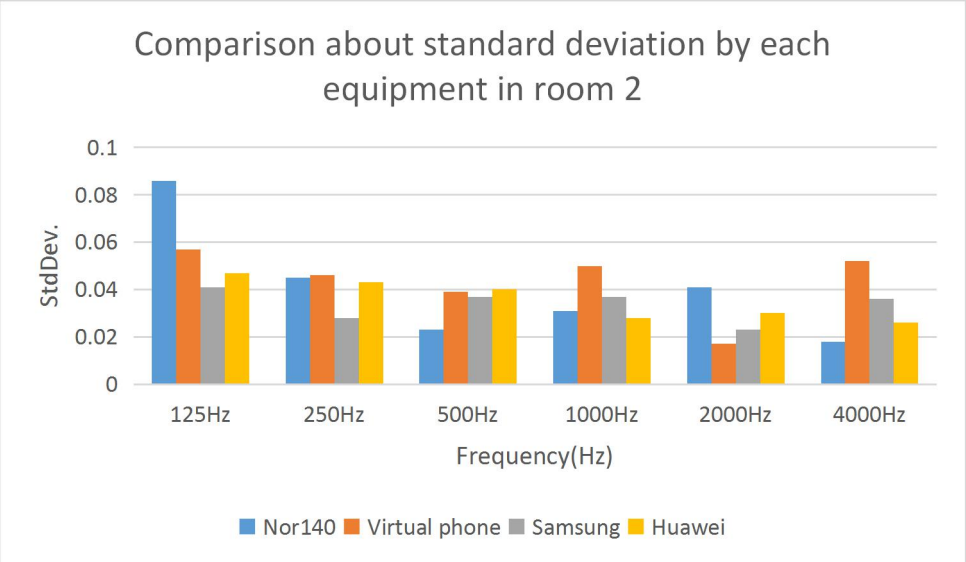


**Figure 36**: Comparison about standard deviation in room 2.

Figure 35 and 36 show the comparison about standard deviation by each equipment in each room. We think the standard deviation is good and our app goes well. However as all the effect factors mentioned before, the results of our app can not be one hundred per cent accurate. Sometimes too high or too low RT60 results occur in certain Octave Bands in one group of results and it is advised to skip the inaccurate data group and repeat the measurement one more time again.

## 4.4 Inaccuracy analysis

Unfortunately, according to the experience of our huge number of tests, the results are inaccurate sometimes, just as shown in the figure below, this one is tested in the Room 1 (reverberation basement):

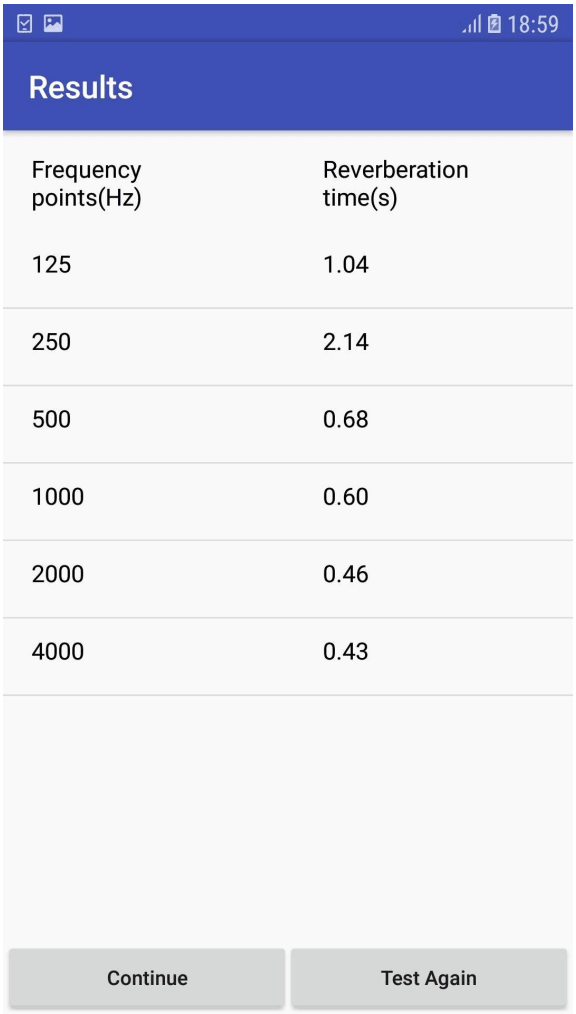


**Figure 37**: An example of an inaccurate test.

It can be easily detected from the group of results above that except for the result for 250Hz, all the other five results are obviously too small compared with the standard data from NOR140.

More groups of inaccurate test results from Room 1 (reverberation basement) in our testing process are shown in the table below:

| | 125Hz | 250Hz | 500Hz | 1000Hz | 2000Hz | 4000Hz |
|---|---|---|---|---|---|---|
| NO.1 | 1.565 | 1.365 | 1.456 | 0.921 | 0.745 | 0.685 |
| NO.2 | 1.753 | 1.367 | 1.416 | 1.066 | 0.64 | 0.63 |
| NO.3 | 1.546 | 1.327 | 1.565 | 1.445 | 1.496 | 1.079 |
| NO.4 | 1.328 | 1.352 | 1.533 | 1.252 | 0.526 | 0.898 |
| NO.5 | 2.988 | 2.609 | 1.789 | 0.817 | 1.387 | 1.984 |

**Table 12**: Groups of inaccurate data in Room 1.

To give a more visible comparison of these inaccurate above, a line chart is made, showing all the inaccurate data above and including the standard result of NOR140:

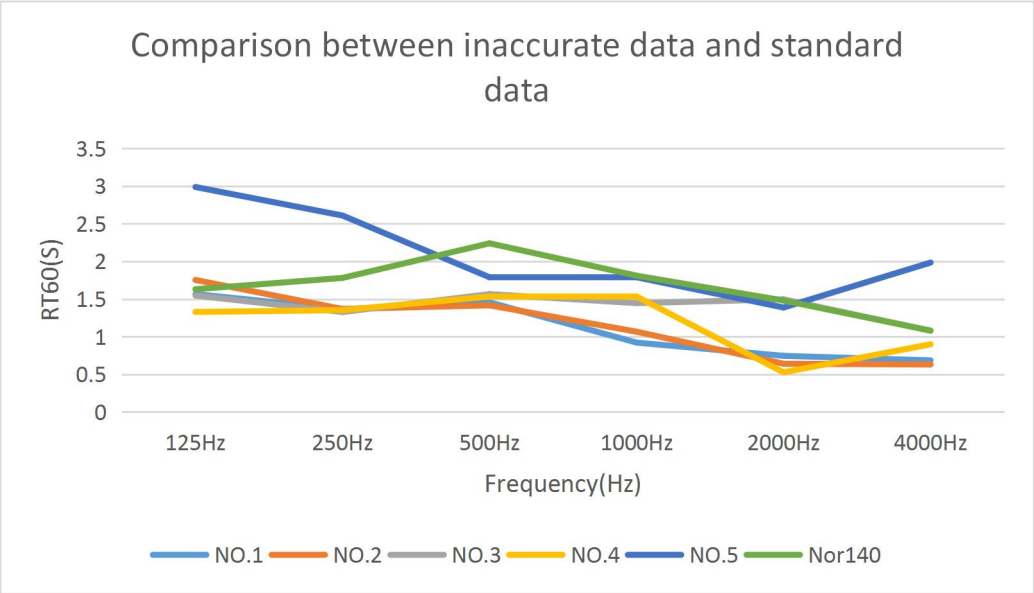


**Figure 38**: Comparison between inaccurate data and standard data.

As can be observed, in some groups of inaccurate data, the overall trend of data from 125Hz to 4000Hz is rough the same as the standard one but all the six results are generally inaccurate. For example, the data of Group 1 and Group 2 in the table above, all the six results are slightly smaller than the standard values but the overall trend of the data from 125Hz to 4000Hz is similar to that of standard results. The reason results in this phenomenon is that the conditions of impulse in these inaccurate measurements are different from that for the standard one. For example, the hand-clapping, if the strength to generate it is more powerful, the time for reverberation is longer and the RT60 results are larger, on the other hand, if the strength is more tiny than the strength of the clapping in the standard test, the RT60 results will all be a little bit smaller than the standard results but the overall trend of data will be similar, just as Group 1 and Group 2 show.

In some other tests with only one or some inaccurate results, just as the data in Group 3 and Group 4 above show, the result for 2000Hz is too larger than the standard one in

Group 3 and the result for 2000Hz is too smaller than the standard one in Group 4 but all the other five results are similar with the standard ones. The reason for this phenomenon is the calculation of RT60 with the algorithm introduced in Chapter 3 going wrong. For example, if one or both of the determined starting point and end point that fix the slope is chosen improperly, the RT60 will definitely be affected. In addition, this kind of inaccuracy can also be caused by the error in recording process and the calculation of SPL. In the development and test process, according to what we checked, some of the SPL values in the group of SPL-Time data are inaccurate in some measurements and if the algorithm use these groups of data with wrong SPL values, the RT60 results for those Octave Bands will also be affected possibly.

Also, if both of the error conditions introduced above happen as the same time, more inaccurate results are gotten, just as the Group 5 above shows. To sum up, at the present stage, the algorithm in this app is still too simple and unstable compared with the professional equipment to give the test SPL and calculate RT60 100% accurately and this is a big issue to be improved in the future.

# Chapter 5 Discussion

Here it is a discussion about the comparison of all results got in Chapter 4.

Figure 28 and figure 29 show the comparison about mean RT60 by each equipment(Nor140, Virtual phone, Samsung Galaxy S7, Huawei Mate9 pro) in room 1 and room 2, it is clear to see the value of RT60 in room 1 is larger than room 2 in each Octave Band. It can found out that the larger the room volume and smaller average sound absorption coefficient of room surface, the longer RT60 will be. This result also suits Sabine Formula.

We compare RT60 in one room by the same equipment and find out the result is different in each test, the reason is we can not strictly keep the condition of sound source(hand clapping) of every measurement to be same. Further, environmental factors such as noise play an important role in making this difference.

Because Nor140 has a higher accuracy measurement level as we have introduced in chapter 4.1, the data it measured is regarded as ideal ones. We compare the results that NOR140 measured in room 1(Table 3) with other three equipment(Table 5,7,9), it is found out the result is different because each equipment has different hardware(also works in room 2). What's more, the results measured in two rooms by virtual phone are more closed to the ideal results of Nor140, the reason we think is that laptop has a better hardware such as microphone and loudspeaker compare to other two android phones.

After all, the standard deviation by each Android phone measured is relatively small , and RT60 is also closed to Nor140 measured. We think we have tried our best to reduce the external influences and it is a perfect app so far.

# Chapter 6 Conclusion

Our thesis project is to build an app that works for measuring RT60 by Android phone, based on applying the theoretical RT60 measurement thoughts to Java codes to the greatest extent, some improvements are also applied to our own Java algorithm, such as noise suppression in the recording process. With this app, measuring RT60 becomes easier and quicker, users are allowed to measure simply with their Android phones.

Beside the acoustic background knowledge, some theories in wireless communication are also used in the algorithm of this app such as FFT/IFFT, sampling theory, frequency spectrum resolution, linear interpolation and so on.

In addition, several practicable functions are built in this app. Setting the number of locations where the impulse is generated and received and taking the average result of several measurements decreases the influence of inaccurate measurement results, reflecting the acoustic condition of tested rooms better. Drawing line chart of measurement results function gives users a clearer view about the level and trends of RT60 results at each target Octave Bands. Exporting measurement results function allows users to store test results and ensures them a more convenient further use of all the data. Setting detailed test room information and taking picture functions help describe testing environment better.

However, the measurement accuracy of this app is limited by some factors, one of them is the gap between Android devices and professional acoustic equipment, both on hardware and algorithm. It has to be admitted that so far the algorithm of this app is still too simple, unstable and inflexible to get accurate measurement results 100% and solve all kinds of testing environment, as well as different impulse conditions. This shortage can be detected in the data comparison section of this theory and also found during our whole testing experience. When we were in the developing process, the greatest efforts were put on solving this issue and it is also an important work for the future. When it comes to one or more obviously wrong measurement results, users are strongly recommended to give up that group of data and restart a new test at the same location, that is the reason why the function of ‘Measure Again’ is added in this app.

To sum up, the testing results of this app are close to the standard results gotten from professional acoustic equipment sometimes but can not be as stable and accurate as them.So the results from this app are more suitable to be a reference of our acoustic measurement and for the more accurate and detailed measurement results, professional acoustic equipment are sometimes better options.

# Chapter 7 Future Work

Some further improvements to be applied to our work are listed as following.

Firstly, the function of scanning the test room with sound and extracting the information that has been calculated to build 3D model of the test room is to be applied, extending the app to do building acoustics insulation (airborne and step sound) measurements, which will be very helpful to architectural acoustics for designer.

Secondly, Java coding still needs to be optimized to decrease the influence of noise further and provide more stable and accurate measurement results.

Thirdly, a data base is needed in this app, which can store all the measurement results in this app even if users have completely closed the it. Database can make the use of this app more convenient for users.

Last but not least, beautify the operation interface to provide users with a better use experience.

# Reference


[1] Ljunggren, F. and A. Ågren, *Potential solutions to improved sound performance of volume based lightweight multi-storey timber buildings*. Applied Acoustics, 2011. **72**(4): p. 231-240.

[2] Flodén, O., et al., *The effect of modelling acoustic media in cavities of lightweight buildings on the transmission of structural vibrations*. Engineering Structures, 2015. **83**(Supplement C): p. 7-16.

[3] Negreira, J., A. Sjöström, and D. Bard, *Low frequency vibroacoustic investigation of wooden T-junctions*. Applied Acoustics, 2016. **105**(Supplement C): p. 1-12.

[4] XiaoMin Li &Haitao Sun. (2015). Introduction of Reverberation Time and Measurement Method. Available at: http://www.sosoas.com/new_content.php?id=59976

[5] Available at: http://www.animations.physics.unsw.edu.au/jw/dB.htm

[6] Yichuan Zhang. (2008). Reverberation Time Introduction. Available at: https://wenku.baidu.com/view/87e40fecaeaad1f346933f3c.html

[7] IOS 3382-1. (2009). Acoustics - Measurement of Room Acoustic Parameters

[8] Carl Hopkins. (2007). Sound Insulation

[9] Tor Erik Vigran. (2008). Building acoustics

[10] Available at: https://en.wikipedia.org/wiki/Reverberation

[11] Available at: https://en.wikipedia.org/wiki/Fast_Fourier_transform#cite_ref-1

[12] Henzi Shen. (2016). Standard about FFT. Available at: https://blog.csdn.net/shenziheng1/article/details/52891807

[13] Available at: https://wenku.baidu.com/view/156305d384254b35eefd346a.htm

[14] Available at: https://en.wikipedia.org/wiki/Octave_band

[15] Available at: https://www.castlegroup.co.uk/guidance/octave-bands/

[16] Available at: https://en.wikipedia.org/wiki/Noise

[17] Instruction Manual of NOR140. Available at: https://nor140.files.wordpress.com/2016/09/nor-140-instruction-manual-v3r0.pdf

# Appendix

## Recording

```
public class AudioRecorder {
    private static AudioRecorder audioRecorder;
    private final static int AUDIO_INPUT =
MediaRecorder.AudioSource.VOICE_COMMUNICATION ;
    private final static int AUDIO_SAMPLE_RATE = 44100;
    private final static int AUDIO_CHANNEL = AudioFormat.CHANNEL_IN_MONO;
    private final static int AUDIO_ENCODING = AudioFormat.ENCODING_PCM_16BIT;
    private int bufferSizeInBytes = 0;
    private AudioRecord audioRecord;
    private Status status = Status.STATUS_NO_READY;
    private String fileName;
    private List<String> filesName = new ArrayList<>();
    private AudioRecorder() {
    }
    public static AudioRecorder getInstance() {
        if (audioRecorder == null) {
            audioRecorder = new AudioRecorder();
        }
        return audioRecorder;
    }
    public void createDefaultAudio(String fileName) {
        bufferSizeInBytes = AudioRecord.getMinBufferSize(AUDIO_SAMPLE_RATE,
                AUDIO_CHANNEL, AUDIO_ENCODING);
        audioRecord = new AudioRecord(AUDIO_INPUT, AUDIO_SAMPLE_RATE,
AUDIO_CHANNEL, AUDIO_ENCODING, bufferSizeInBytes);
        this.fileName = fileName;
        status = Status.STATUS_READY;
    }
    public void startRecord(final RecordStreamListener listener) {

        if (status == Status.STATUS_NO_READY || TextUtils.isEmpty(fileName)) {
            throw new IllegalStateException("Recording hasn't been initialized,
check recording permission. ");
        }
        if (status == Status.STATUS_START) {
            throw new IllegalStateException("Recording...");
        }
        Log.d("AudioRecorder","===startRecord==="+audioRecord.getState());
```

```
        audioRecord.startRecording();

        new Thread(new Runnable() {
            @Override
            public void run() {
                writeDataTOFile(listener);
            }
        }).start();
    }
public void stopRecord() {
    Log.d("AudioRecorder","===stopRecord===");
    if (status == Status.STATUS_NO_READY || status == Status.STATUS_READY) {
        throw new IllegalStateException("Recording hasn't begun.");
    } else {
        audioRecord.stop();
        status = Status.STATUS_STOP;
        release();
    }
}
}
```

## IFFT

```
public class IFFT {

    int n, m;
    double[] cos;
    double[] sin;
    double[] window;
    public IFFT(int n) {
        this.n = n;
        this.m = (int) (Math.log(n) / Math.log(2));
        if (n != (1 << m))
            throw new RuntimeException("FFT length must be power of 2");
        cos = new double[n / 2];
        sin = new double[n / 2];
        for (int i = 0; i < n / 2; i++) {
            cos[i] = Math.cos(2 * Math.PI * i / n);
            sin[i] = Math.sin(2 * Math.PI * i / n);
        }
        makeWindow();
    }
          protected void makeWindow() {
        window = new double[n];
        for (int i = 0; i < window.length; i++)
            window[i] = 0.42 - 0.5 * Math.cos(2 * Math.PI * i / (n - 1))
                    + 0.08 * Math.cos(4 * Math.PI * i / (n - 1));
    }
    public double[] getWindow() {
        return window;
    }
    public void ifft(double[] x, double[] y) {
        int i, j, k, n1, n2, a;
        double c, s, e, t1, t2;
        j = 0;
        n2 = n / 2;
        for (i = 1; i < n - 1; i++) {
            n1 = n2;
            while (j >= n1) {
                j = j - n1;
                n1 = n1 / 2;
            }
            j = j + n1;
            if (i < j) {
                t1 = x[i];
```

```
                x[i] = x[j];
                x[j] = t1;
                t1 = y[i];
                y[i] = y[j];
                y[j] = t1;
            }
        }
        n1 = 0;
        n2 = 1;
        for (i = 0; i < m; i++) {
            n1 = n2;
            n2 = n2 + n2;
            a = 0;
            for (j = 0; j < n1; j++) {
                c = cos[a];
                s = sin[a];
                a += 1 << (m - i - 1);
                for (k = j; k < n; k = k + n2) {
                    t1 = c * x[k + n1] - s * y[k + n1];
                    t2 = s * x[k + n1] + c * y[k + n1];
                    x[k + n1] = x[k] - t1;
                    y[k + n1] = y[k] - t2;
                    x[k] = x[k] + t1;
                    y[k] = y[k] + t2;
                }
            }
        }
    }
}
```

## FFT

```
public class FFT {
    int n, m;

    double[] cos;
    double[] sin;
    double[] window;
    public FFT(int n) {
        this.n = n;
        this.m = (int) (Math.log(n) / Math.log(2));
        if (n != (1 << m))
            throw new RuntimeException("FFT length must be power of 2");
        cos = new double[n / 2];
        sin = new double[n / 2];
        for (int i = 0; i < n / 2; i++) {
            cos[i] = Math.cos(-2 * Math.PI * i / n);
            sin[i] = Math.sin(-2 * Math.PI * i / n);
        }
        makeWindow();
    }
    protected void makeWindow() {
            window = new double[n];
        for (int i = 0; i < window.length; i++)
            window[i] = 0.42 - 0.5 * Math.cos(2 * Math.PI * i / (n - 1))
                    + 0.08 * Math.cos(4 * Math.PI * i / (n - 1));
    }
    public double[] getWindow() {
        return window;
    }
    public void fft(double[] x, double[] y) {
        int i, j, k, n1, n2, a;
        double c, s, e, t1, t2;
        j = 0;
        n2 = n / 2;
        for (i = 1; i < n - 1; i++) {
            n1 = n2;
            while (j >= n1) {
                j = j - n1;
                n1 = n1 / 2;
            }
            j = j + n1;
            if (i < j) {
                t1 = x[i];
```

```
                x[i] = x[j];
                x[j] = t1;
                t1 = y[i];
                y[i] = y[j];
                y[j] = t1;
            }
        }
        n1 = 0;
        n2 = 1;
        for (i = 0; i < m; i++) {
            n1 = n2;
            n2 = n2 + n2;
            a = 0;
            for (j = 0; j < n1; j++) {
                c = cos[a];
                s = sin[a];
                a += 1 << (m - i - 1);
                for (k = j; k < n; k = k + n2) {
                    t1 = c * x[k + n1] - s * y[k + n1];
                    t2 = s * x[k + n1] + c * y[k + n1];
                    x[k + n1] = x[k] - t1;
                    y[k + n1] = y[k] - t2;
                    x[k] = x[k] + t1;
                    y[k] = y[k] + t2;
                }
            }
        }
    }
}
```

## Analyze

```
mVisualizer.setDataCaptureListener(new Visualizer.OnDataCaptureListener() {
    @Override
    public void onWaveFormDataCapture(Visualizer visualizer, byte[] waveform, int
samplingRate) {
    }
    @Override
    public void onFftDataCapture(Visualizer visualizer, byte[] fft, int
sampling_rate) {
        //1。IFFT
        double fft1r[] = new double[512];
        double fft1i[] = new double[512];
        double ifft1r_zero[] = new double[32768];
        double ifft1i_zero[] = new double[32768];
        double fft2[] = new double[ifft1i_zero.length * 2];
        byte fft2_byte[] = new byte[fft2.length];
        for (int i = 0; i < 512; i++) {
            fft1r[i] = fft[i * 2];
            fft1i[i] = fft[2 * i + 1];
        }
        IFFT ifft_My = new IFFT(512);
        IFFT.beforeAfter(ifft_My, fft1r, fft1i);
        for (int i = 0; i < fft1r.length; i++) {
            fft1r[i] = fft1r[i] / 512;
            fft1i[i] = fft1i[i] / 512;
        }
        for (int i = 0; i < fft1r.length; i++) {
            ifft1r_zero[i] = fft1r[i];
            ifft1i_zero[i] = fft1i[i];
        }
        //add zeros
        for (int i = fft1r.length; i < ifft1r_zero.length; i++) {
            ifft1r_zero[i] = 0;
            ifft1i_zero[i] = 0;
        }
        //fft again
        Fff fft_My = new Fff(ifft1r_zero.length);
        Fff.beforeAfter(fft_My, ifft1r_zero, ifft1i_zero);

        //Im Re cross
        for (int i = 0; i < ifft1r_zero.length; i++) {
            fft2[2 * i] = ifft1r_zero[i];
            fft2[2 * i + 1] = ifft1i_zero[i];
```

```
        }
        //double2byte
        for (int i = 0; i < fft2.length; i++) {
            fft2_byte[i] = (byte) fft2[i];
        }
        updateVisualizer(fft2_byte, sampling_rate / 1000,
mMediaPlayer.getCurrentPosition());
    }
}, mVisualizer.getMaxCaptureRate(), false, true);
mVisualizer.setEnabled(true);

public void updateVisualizer(byte[] fft, int fs, int playTime) {
    byte[] fftdata125 = new byte[130];
    byte[] fftdata250 = new byte[264];
    byte[] fftdata500 = new byte[512];
    byte[] fftdata1000 = new byte[1052];
    byte[] fftdata2000 = new byte[2098];
    byte[] fftdata4000 = new byte[4200];

    for (int i = 132; i < 262; i++) {
        fftdata125[i - 132] = fft[i];
    }
    double recordBeanList12 = doublecalculateVolume(fftdata125);

    for (int i = 262; i < 526; i++) {
        fftdata250[i - 262] = fft[i];
    }
    double recordBeanList22 = doublecalculateVolume(fftdata250);

    for (int i = 526; i < 1038; i++) {
        fftdata500[i - 526] = fft[i];
    }
    double recordBeanList32 = doublecalculateVolume(fftdata500);

    for (int i = 1052; i < 2104; i++) {
        fftdata1000[i - 1052] = fft[i];
    }
    double recordBeanList42 = doublecalculateVolume(fftdata1000);

    for (int i = 2104; i < 4202; i++) {
        fftdata2000[i - 2104] = fft[i];
    }
    double recordBeanList52 = doublecalculateVolume(fftdata2000);
```

```
    for (int i = 4202; i < 8402; i++) {
        fftdata4000[i - 4202] = fft[i];
    }
    double recordBeanList62 = doublecalculateVolume(fftdata4000);

    double db_125 = recordBeanList12;
    mRecordBeans_125.add(new RecordBean(db_125, playTime, 125));

    double db_250 = recordBeanList22;
    mRecordBeans_250.add(new RecordBean(db_250, playTime, 250));

    double db_500 = recordBeanList32;
    mRecordBeans_500.add(new RecordBean(db_500, playTime, 500));

    double db_1000 = recordBeanList42;
    mRecordBeans_1000.add(new RecordBean(db_1000, playTime, 1000));

    double db_2000 = recordBeanList52;
    mRecordBeans_2000.add(new RecordBean(db_2000, playTime, 2000));

    double db_4000 = recordBeanList62;
    mRecordBeans_4000.add(new RecordBean(db_4000, playTime, 4000));
}
private double calcRT60forLowFreq(List<RecordBean> listbean, int a, double b)
{   Log.d("Here", "lp = " +listbean .get(a ).lp +"   "+listbean .get(a).time);
    double slope_new =0;
    double RT20 = 0;
    int number=0;
    int start_for_high=0;
    int end_for_high=0;
    double RT60 = 0;
    int start=0;
    int larger[]=new int [listbean .size() ];
    int true_start=0;
    int end=0;
    double end_time=0;
    double end_lp=0;
    double lp_final[]=new double [listbean .size() *50];
    double time_final[]=new double [listbean .size() *50];
    if ((a-5)<=40){
        start_for_high  =40;
    }
    else {start_for_high =(a-5);}
    double max = listbean.get(0).lp;
```

```
    for(int i=0;i<listbean .size() ;i++){
        if (listbean .get(i).time>b  ){
            end_for_high =i;
            break;
        }
    }
    Log.d("Here 2", "lp = " +b +" "+listbean .get(end_for_high ).lp );
    for (int i = start_for_high  ; i < end_for_high ; i++) {
        if (max < listbean.get(i).lp) {
            max = listbean.get(i).lp;
            start = i;
        }
    }
    Log.d("max point", "lp = " +max );
    Log.d("max point", "time = " +listbean .get(start).time  );
    for (int i=1;i <listbean  .size()-1 ;i++){
        if (listbean .get(i).lp==Double.NEGATIVE_INFINITY  ){
            if (listbean .get(i+1).lp==Double.NEGATIVE_INFINITY  ){
                for (int j=i+1;j<listbean .size() ;j++)
                {
                    if (listbean.get(j).lp!=Double .NEGATIVE_INFINITY){
                        double lp[]
=interpolate(listbean.get(i-1).lp,listbean.get(j).lp,j-i+1 ) ;
                        for(int q=0;q<lp.length;q++ ){
                            for(int p=1;p<lp.length ;p++){
                                listbean .get(i+p-1) .lp=lp[p];
                            }
                        }
                        break;
                    }
                }}
            else if (listbean .get(i+1).lp!=Double .NEGATIVE_INFINITY  ){
             listbean .get(i).lp=(listbean .get(i-1).lp+listbean .get(i+1).lp)/2;
            }
        }
    }
    for(int j=0;j<2;j++) {
        for (int i = 1; i < listbean.size() - 1; i++) {
            if (listbean.get(i - 1).lp > listbean.get(i).lp && listbean.get(i +
1).lp > listbean.get(i).lp) {
                listbean.get(i).lp = (listbean.get(i - 1).lp + listbean.get(i +
1).lp) / 2;
            }
        }
```

```
    }
    for (int i=0 ;i<listbean .size() ;i++){
        Log.d("all new", "lp = " +listbean .get(i) .lp +"
"+listbean .get(i).time );
    }
    for(int i=start+1;i<end_for_high;i++){
        if(max -listbean .get(i).lp<7){
            true_start =i;
            number=number +1;
        }
    }
    if (number==0){
        if(max -listbean .get(start+1 ).lp>7 ){
            true_start =start ;}
        else{true_start =start+1;}
    }
    if (listbean.get(true_start).lp-listbean.get(true_start +1).lp<4){
        for (int i=true_start ;i<end_for_high ;i++ ) {
            if (listbean.get(true_start).lp - listbean.get(i).lp < 4) {
                true_start = i;
            }
        }
    }
    Log.d("How many", "true start 1 = " + number+"  "
+listbean .get(true_start ).lp +" "+listbean .get(true_start  ).time);
    for (int i=true_start;i<listbean .size()-1;i++){
        double lp[] =interpolate(listbean.get(i).lp,listbean.get(i+1).lp,51 ) ;
        double time[]
=interpolate(listbean.get(i).time ,listbean.get(i+1).time,51 ) ;
        for ( int j=0 ;j<lp .length ;j++ ){
            lp_final[(i-true_start)*51+j]=lp [j];
            time_final[(i-true_start)*51+j]=time[j];
        }
    }
    double lowest_time=listbean.get(start).time;
    double lowest_lp=max;
    int lowest=0;
    for (int i=start ;i< listbean.size() ;i++){
        if(listbean .get(i).lp<lowest_lp && listbean .get(i).lp>30 ){
            lowest_lp =listbean .get(i).lp;
            lowest_time=listbean .get(i).time;
            lowest=i;
        }
    }
```

```
    Log.d("lowest point", "lp = " +lowest_lp  );
    Log.d("lowest point", "time = " +lowest_time  );
    for (int i=0;i<lp_final.length ;i++){
        if(lp_final [i]<lowest_lp +12){
            end =i;
            end_lp=lp_final [i];
            end_time =time_final [i];
            if(Math.abs(end_lp-listbean .get(true_start ).lp)<10 ){
                for (int j=0;j<lp_final.length ;j++){
                    if(lp_final [j]<lowest_lp +8){
                        end =j;
                        end_lp=lp_final [j];
                        end_time =time_final [j];
                        break;
                    }
                }
            }
            break;
        }
    }
    Log.d("end point", "lp = " +end_lp);
    Log.d("end point", "time = " +end_time);
    if(end_lp ==listbean .get(true_start ).lp){
        slope_new =Double .POSITIVE_INFINITY ;
    }
    else{
    slope_new =(lp_final
[end ]-listbean .get(true_start ).lp )/(listbean .get(true_start ).time
-time_final [end]);}

    Log.d("slope", "slope new = " +slope_new);
    RT20 =  20/slope_new;
    Log.d("calcRT30", "RT60===" +listbean .get(listbean .size()-1 ).time );
    RT60 = RT20 * 3*5000/(1000*listbean .get(listbean .size() -1).time  );
    Log.d("calcRT30", "RT60===" + RT60);
    DecimalFormat df = new DecimalFormat("0.000");
    RT60=Double.parseDouble(df.format(RT60));
    return   RT60 * 1.0;
}
private double calcRT60(List<RecordBean> listbean) {
    double slope_new =0;
    double RT20 = 0;
    int number=0;
    double RT60 = 0;
```

```
    int true_start=0;
    int end=0;
    int larger[]=new int [listbean .size() ];
    double end_time=0;
    double end_lp=0;
    double lp_final[]=new double [listbean.size()*50];
    double time_final[]=new double [listbean.size()*50];
    int start = findMaxIndex(listbean) ;
    start_point =start;
    start_time =listbean.get(start ).time ;
        double max = listbean.get(start).lp;
    Log.d("max point", "lp = " +max );
    Log.d("max point", "time = " +listbean .get(start).time  );

    for (int i=1;i <listbean  .size()-1 ;i++){
        if (listbean .get(i).lp==Double.NEGATIVE_INFINITY  ){
                    if (listbean .get(i+1).lp==Double.NEGATIVE_INFINITY  ){
                for (int j=i;j<listbean .size() ;j++)
                {
                    if (listbean.get(j).lp!=Double .NEGATIVE_INFINITY){
                        double lp[]
=interpolate(listbean.get(i-1).lp,listbean.get(j).lp,j-i+1 ) ;
                        for(int q=0;q<lp.length;q++ ){
                            for(int p=1;p<lp.length ;p++){
                                listbean .get(i+p-1) .lp=lp[p];
                            }
                        }
                        break;
                    }
                }}
            else if (listbean .get(i+1).lp!=Double .NEGATIVE_INFINITY  ){

listbean .get(i).lp=(listbean .get(i-1).lp+listbean .get(i+1).lp)/2;
            }
        }
    }
    for(int j=0;j<2;j++) {
        for (int i = 1; i < listbean.size() - 1; i++) {
            if (listbean.get(i - 1).lp > listbean.get(i).lp && listbean.get(i +
1).lp > listbean.get(i).lp) {
                listbean.get(i).lp = (listbean.get(i - 1).lp + listbean.get(i +
1).lp) / 2;
            }
        }
```

```
    }
    for (int i=0 ;i<listbean .size() ;i++){
        Log.d("all new", "lp = " +listbean .get(i) .lp +"
"+listbean .get(i).time );
    }
    for(int i=start+1;i<listbean .size() ;i++){
        if(max -listbean .get(i).lp<7){
            true_start =i;
            number=number +1;
        }
    }
    if (number==0){
        if(max -listbean .get(start+1 ).lp>7 ){
            true_start =start ;}
        else{true_start =start+1;}
    }
    Log.d("How many", "avg = " + number+"  " +listbean .get(true_start ).lp+"
"+listbean .get(true_start  ).time);
    //两点间穿插 20 个数
    for (int i=true_start;i<listbean .size()-1;i++){
        double lp[] =interpolate(listbean.get(i).lp,listbean.get(i+1).lp,51 ) ;
        double time[]
=interpolate(listbean.get(i).time ,listbean.get(i+1).time,51 ) ;
        for ( int j=0 ;j<lp .length ;j++ ){
            lp_final[(i-true_start)*51+j]=lp [j];
            time_final[(i-true_start)*51+j]=time[j];
        }
    }
    double lowest_time=listbean.get(start).time;
    double lowest_lp=max;
    int lowest=0;
    for (int i=start ;i< listbean.size() ;i++){
        if(listbean .get(i).lp<lowest_lp && listbean .get(i).lp>30 ){
            lowest_lp =listbean .get(i).lp;
            lowest_time=listbean .get(i).time;
            lowest=i;
        }
    }
    Log.d("lowest point", "lp = " +lowest_lp  );
    Log.d("lowest point", "time = " +lowest_time  );
    for (int i=0;i<lp_final.length ;i++){
        if(lp_final [i]<lowest_lp +12){
            end =i;
            end_lp=lp_final [i];
```

```
            end_time =time_final [i];
            if(Math.abs(end_lp-listbean .get(true_start ).lp)<10 ){
                for (int j=0;j<lp_final.length ;j++){
                    if(lp_final [j]<lowest_lp +8){
                        end =j;
                        end_lp=lp_final [j];
                        end_time =time_final [j];
                        break;
                    }
                }
            }
            break;
        }
    }
    end_point =end ;
    end_time_point  =end_time ;
    Log.d("end point", "lp = " +end_lp);
    Log.d("end point", "time = " +end_time);
    slope_new =(lp_final
[end ]-listbean .get(true_start ).lp )/(listbean .get(true_start ).time
-time_final [end]);
    Log.d("slope", "slope new = " +slope_new);
    RT20 =  20/slope_new;
    Log.d("calcRT30", "RT60===" +listbean .get(listbean .size()-1 ).time );
    RT60 = RT20 * 3*5000/(1000*listbean .get(listbean .size() -1).time  );
    Log.d("calcRT30", "RT60===" + RT60);
    DecimalFormat df = new DecimalFormat("0.000");
    RT60=Double.parseDouble(df.format(RT60));
    return   RT60 * 1.0;
}
private double doublecalculateVolume(byte[] buffer) {

    double sumVolume = 0.0;
    double avgVolume = 0.0;
    double volume = 0.0;
      for (int i = 0; i < buffer.length; i += 2) {
       int v1 = buffer[i] & 0xFF;
       int v2 = buffer[i + 1] & 0xFF;
       int temp = v1 + (v2 << 8);// 小端
       if (temp >= 0x8000) {
           temp = 0xffff - temp;
       }

       sumVolume += Math.abs(temp * temp);
```

```
    }
    avgVolume = sumVolume / buffer.length / 2;
    volume = Math.log10(avgVolume) * 10;
    if (volume < 0 && volume != Double.NEGATIVE_INFINITY) {
        volume = 30;
    }
    return volume;
}
```