CSE 570 Final Project Report
Sonus Demutator

Akhil Bhutani (110898687) abhutani@cs.stonybrook.edu,

Siddharth Jain (111425239) sijain@cs.stonybrook.edu

Department of Computer Science, Stony Brook University.

Abstract

Occasionally, we find ourselves in noisy environments like subways, public places, streets etc. In these scenarios, we have to manually adjust the media volume if listening to our favorite song, while on a call etc. which is sometimes inconvenient. Our application solves this problem by automatically adjusting the volume level, based on the ambient noise levels.

The project is called “Sonus-Demutator” which is a Latin phrase meaning, ‘*The change in sound*’. It is an Acoustic Sensing based application that automatically changes the media volume, based on background noise. When the user is in a loud environment, it increases the volume and vice versa. Once triggered, the application runs smoothly in the background and works as intended.

The project is an android based application, developed using the Android Studio. The programming language is Java. The purpose of the application is to help making the music listening experience as effortless as possible.

1 Introduction

An audio signal is an analog or digital representation of sound. Acoustic Sensing concerns the analysis and synthesis of sound signals by utilizing the capabilities of specialized instruments which can detect the changes in these signals.

Sonus Demutator is an android based Acoustic Sensing application that automatically changes the media volume based on background noise by utilizing the hardware capabilities of a smartphone. It increases the volume when the user is in noisy environment and decreases it to the preset level, when the noise subsides.

We chose android as our application development platform due to its sundry features such as high customizability of application interface and seamless interaction of services with UI, ease of development and also variety of inbuilt APIs to support an application that uses data from various sensors to make system wide changes.

Development in android is done using Activities in the system, which are managed as an *activity stack*. When a new activity is started, it is placed on the top of the stack and becomes the running activity the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

The following diagram shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The colored ovals are major states the Activity can be in.

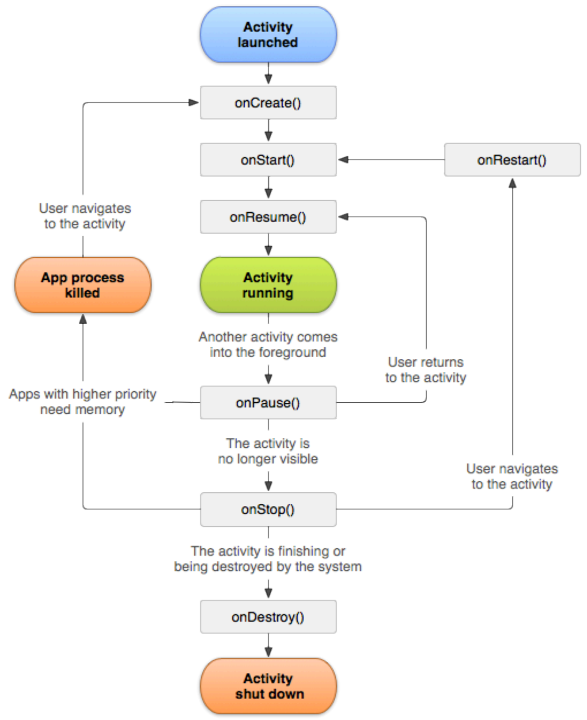

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 1: Activity Life Cycle of a Sample Android App

1.1 Android Components Used.

Media Recorder
Used to record audio and video.

Audio Manager
Audio Manager provides access to volume and ringer mode control.

Android TextView
A user interface element that displays text to the user

Android Intent
An intent is an abstract description of an operation to be performed

Background Service
A Service is an application component that can perform long-running operations in the background, and it does not provide a user interface.

2 Design

2.1 Application Life Cycle

App in initialized.
When initialized, the app generates a view and starts the thread once the trigger is pressed.

Mic Recording starts.
App starts listening to the data input by the microphone and displays it on the screen.

Media level modulation
Starts the system service and changes the media levels. The app continues to run in the background even if it is minimized.

Close
When the app is terminated by the user or if the trigger switch is turned off, then the app stops listening and changing the media levels.

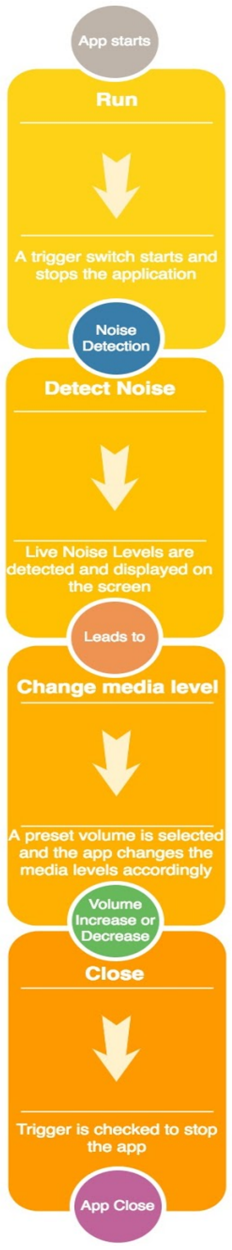
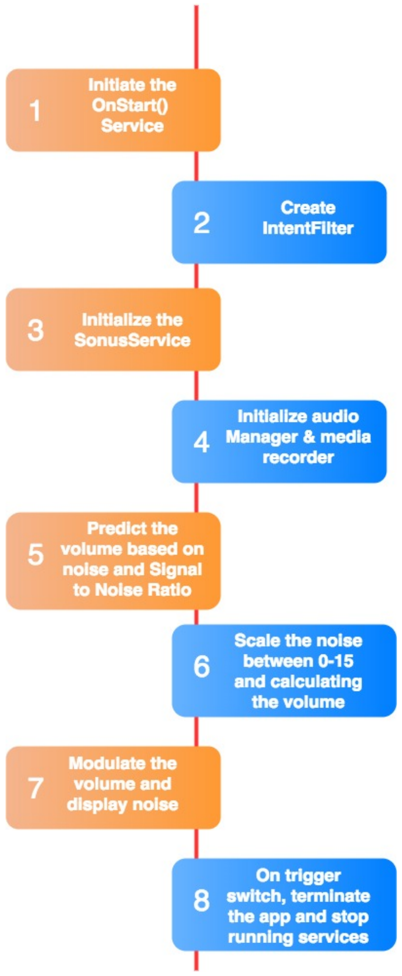

App starts
Run
A trigger switch starts and stops the application
Noise Detection
Detect Noise
Live Noise Levels are detected and displayed on the screen
Leads to
Change media level
A preset volume is selected and the app changes the media levels accordingly
Volume Increase or Decrease
Close
Trigger is checked to stop the app
App Close

Figure 2: Activity Life Cycle of Sonus Demutator

2.2 Application Flow Chart



1 Initiate the OnStart() Service
2 Create IntentFilter
3 Initialize the SonusService
4 Initialize audio Manager & media recorder
5 Predict the volume based on noise and Signal to Noise Ratio
6 Scale the noise between 0-15 and calculating the volume
7 Modulate the volume and display noise
8 On trigger switch, terminate the app and stop running services

Figure 3: Application flow chart of Sonus Demutator

2.3 Application Logic and Implementation

The application life cycle and flow chart has already been presented above to give a brief idea about the application. There are two ends of this Android Application - UI and Service. We have initialized the various Android Components in onCreate function in UI which includes audioManager, mediaRecorder, TextView and so on. Then we request for necessary permissions from the user as Android 6.0 requires explicit permissions to be granted.

In case the permissions are not authorized, we exit the application. Intent is used to pass the data between the UI and the service through the onStart() method. Also, we register a receiver to listen for broadcast request from the service as we intent to update the UI based on the service data gathered.

In the receiver, we get the current MIC and volume readings and update it in UI if the UI is visible to the user. In the service end, we have initialized the intent and other android services including mediaRecorder and audioManager. We create a thread in the service onStartCommand() method which gets the current Amplitude reading from the MIC and scales the volume between the minimum and maximum volume level supported by the device. To get the decibel reading, we perform some logarithmic calculations on the amplitude obtained from the MIC and display the reading in the Android TextView to help user gain an insight on the current amplitude values.

After obtaining the volume level based on the background noise, we broadcast the amplitude reading and the volume level to the UI thread if it is in the foreground. These values are received by the receiver and are used to update UI components. To avoid stressing the service, we have imposed a delay of 300 ms in our thread. When the application is placed in the background, the service keeps running and performs the necessary calculations to update the volume level in case of background noise. We have also provided a volume seekbar which helps set a minimum volume that the user wishes to maintain.

The volume of the audioManager is updated only when the volume level calculated by considering the background noise is greater than the minimum volume set by the user. Otherwise, there is no change in the volume levels. The user can set the minimum or preset volume that needs to be maintained by using the hardware buttons or the seekbar provided in the UI.

2.4 Important Formulae

This is used to calculate predicated volume based on max amplitude from MIC.

$$Predicted\ Volume =$$

$$SNR * Amplitude + (1 - SNR) * (PredictedVolume)$$

This is used to scale the volume according to device volume levels.

$$Scaled\ Volume =$$

$$\frac{MicReading}{AmpHigh - AmpLow} * (VolumeHigh - VolumeLow)$$

3 Results

3.1 Data Readings from the app

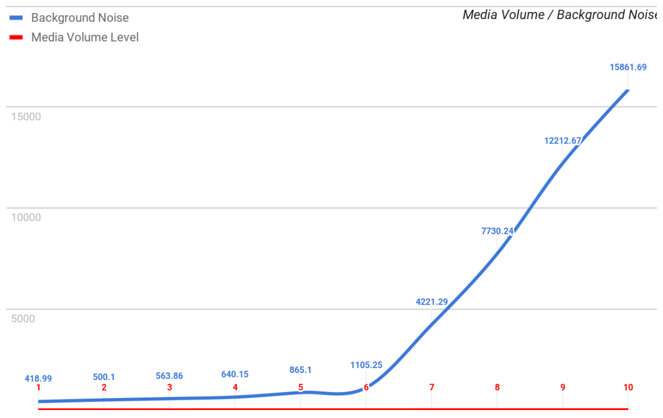


Figure 4: Background vs Volume Level when preset is 0

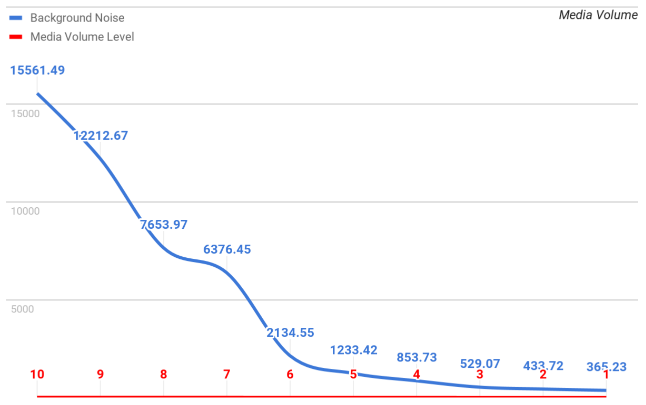


Figure 5: Background vs Volume Level when preset is 5

3.2 Background Test

Custom 1

We tested the application by rapidly increasing the sound levels in the background (we used a JBL GO and increased the volume of the song playing on it) and the app responded by altering the volume levels accordingly.

Background Noise (16-bit digitalization of electrical output from mic)	Media Volume Level
418.99	1
500.1	2
563.86	3
640.15	4
865.1	5
1105.25	6
4221.29	7
7730.24	8
12212.67	9
15861.69	10

Table 1: Decrement of media volume based on background noise levels.

Custom 2

In this case we tested the application by turning the speaker on full volume and then decreased the volume step by step. The app then responded by decreasing the media volume accordingly.

Background Noise (16-bit digitalization of electrical output from mic)	Media Volume Level
15561.49	10
12212.67	9
7653.97	8
6376.45	7
2134.55	6
1233.42	5
853.73	4
529.07	3
433.72	2
365.23	1

Table 2: Increment of media volume based on background noise levels.

Subway
We took our phones out in the Union Square Subway to test its capabilities, it increased the media volume to a particular level where the music increased gradually.

Background Noise (16-bit digitalization of electrical output from mic)	Media Volume Level
10542.98	9
11523.85	9
10567.32	9
12564.43	9
13453.79	9
15767.19	10
12989.01	9
18928.08	10
23839.50	10
25829.02	10

Table 3: Media levels corresponding to background levels in the New York Subway.

Library
Now to check if the app works correctly in the silent zone as well, we played music in our earphones and moved to the silent zone of the Melville Library. We set our preset volume to 5 and as expected, the app didn't increased the media volume any further.

Background Noise (16-bit digitalization of electrical output from mic)	Media Volume Level
478.08	3
490.31	3
489.04	3
602.62	4
503.42	4
407.48	3
460.79	3
528.134	4
378.91	2
398.27	2

Table 4: Media levels corresponding to background levels in the Melville Library Silent Zone.

3.3 App Screenshots

The app has a minimal UI which contains a trigger switch to start or stop the service, a volume slider with which user can set a predefined minimum volume.

We created two live progress bars which show the current volume level as well as the background noise levels.

We also displayed the live background noise level (electric output from microphone) from the readings recorded using the inbuilt microphone of the smartphone.

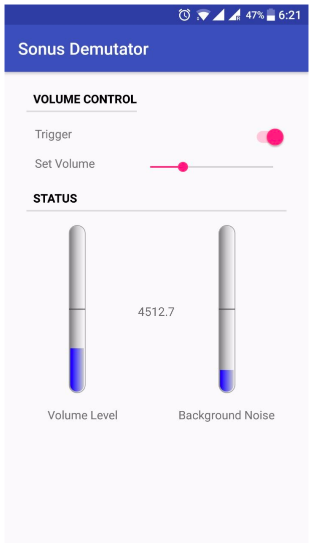


Figure 6: Shows that the Volume level remains stagnant until the background noise reaches the preset.

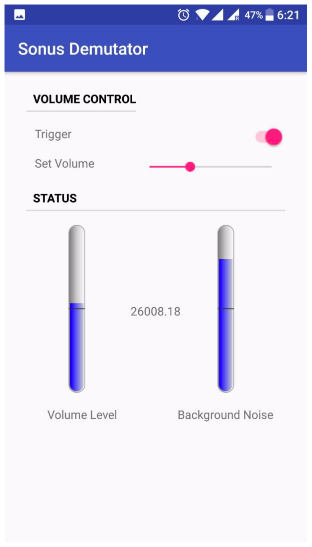


Figure 7: Shows that the Volume level increases as the background noise increases.

4 Conclusion

With this application, we are able to use the principles of acoustic sensing in a real-world scenario by providing hassle free music listening experience to the user. By using a smartphone, the app exploits the microphone for sensing the environment using acoustic signals. Our app detects the background noise and records the amplitude, then it calculates the signal to noise ratio and changes the media level accordingly. If the app senses that the background levels are too high, it increases the media volume and when the noise subsides it reduces the volume to predefined preset set by the user.

The app also runs in the background if minimized.

5 Future Work

The future scope of this application is characterizing different media modes based on the environment profiles. For any particular environment user can save his desired settings and simply enable that profile to match his/her requirements. Also, this application can be beneficial for the people with hearing needs, in which the app can automatically max out the media levels as per requirement.

6 Acknowledgements

We would like to thank Professor Jie Gao for her support and encouragement during the course of the project which helped us to deliver the project with astonishing results and working model of the intended application.

7 References

[1] https://developer.android.com/training/index.html

[2] https://sites.google.com/cs.stonybrook.edu/cse570

[3] https://en.wikipedia.org/wiki/Signal-to-noise_ratio

[4] https://en.wikipedia.org/wiki/Signal_processing

[5] https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle

8 Project Contribution

Akhil Bhutani	50%
Siddharth Jain	50%