User's Guide:
How to Use Two External Microphones in Smartphone Apps for Hearing Improvement Studies

A. SEHGAL AND N. KEHTARNAVAZ
UNIVERSITY OF TEXAS AT DALLAS

MARCH 2018

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

Table of Contents

Introduction

This user's guide describes how to use two microphones in smartphone apps for hearing improvement studies. As this capability currently only exists in iOS mobile devices, the user's guide is limited to the use of such devices. More specifically, two iOS smartphone apps are developed in which two external microphones are used. These apps consist of two microphones noise classification and two microphones noise reduction compared to the previously developed single microphone noise classification and single microphone noise reduction. The technical details of the two microphones smartphone apps and their real-time implementation aspects are covered in the following conference paper:

A. Sehgal and N. Kehtarnavaz, "Utilization of Two Microphones for Real-Time Low-Latency Audio Smartphone Apps" *Proceedings of IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, Jan 2018.

This guide is divided into two parts. The first part covers the two microphones noise classification app and the second part covers the two microphones noise reduction app. Each part consists of two sections. In the first section, the GUI of the app is explained. The second section explains the code flow and how an audio stream is processed.

The two microphones noise classification app developed uses Subband Features and a Random Forest classifier with the technical details appearing in the following papers and documents:

Algorithm: http://ieeexplore.ieee.org/document/6854270/

Smartphone Implementation: http://ieeexplore.ieee.org/document/7472068/

User's Guide: https://www.utdallas.edu/ssprl/files/UsersGuide-NoiseClassifier-App1.pdf

The two microphones noise reduction app developed uses the Wiener filtering algorithm with post-filtering whose technical details appear in the following papers and documents:

Smartphone Implementation: http://ieeexplore.ieee.org/document/8001429/

User's Guide: https://www.utdallas.edu/ssprl/files/Users-Guide-Noise-Reduction.pdf

In the above papers and documents, only a single microphone is used. This is due to the existing limitations in iOS and Android mobile devices. In Android mobile devices, it is possible to use two microphones but that approach violates the warranty and generate an unacceptably

high audio latency (>200ms). In iOS mobile devices (iPhones and iPads), it is possible to use external microphones with low-latency via the lightning port

The apps developed require the use of an external dual microphone. In our case, the following dual microphones were used, see Fig. 1:

1. Zoom iQ7
2. Sennheiser Ambeo Smart Headset

Zoom iQ7

Sennheiser Ambeo Smart Headset

Fig. 1

Two Microphone App Folder Description

Fig. 2 lists the contents of the folder that contains the codes for the apps developed. These contents include:

- "Noise-Classification-2-Mic" is the two microphone noise classification project. To open the project, double click on "Noise_Classification.xcodeproj" inside the folder.
- "Noise-Reduction-2-Mic" is the two microphone noise reduction project. To open the project, double click on "Noise_Reduction.xcodeproj" inside the folder.

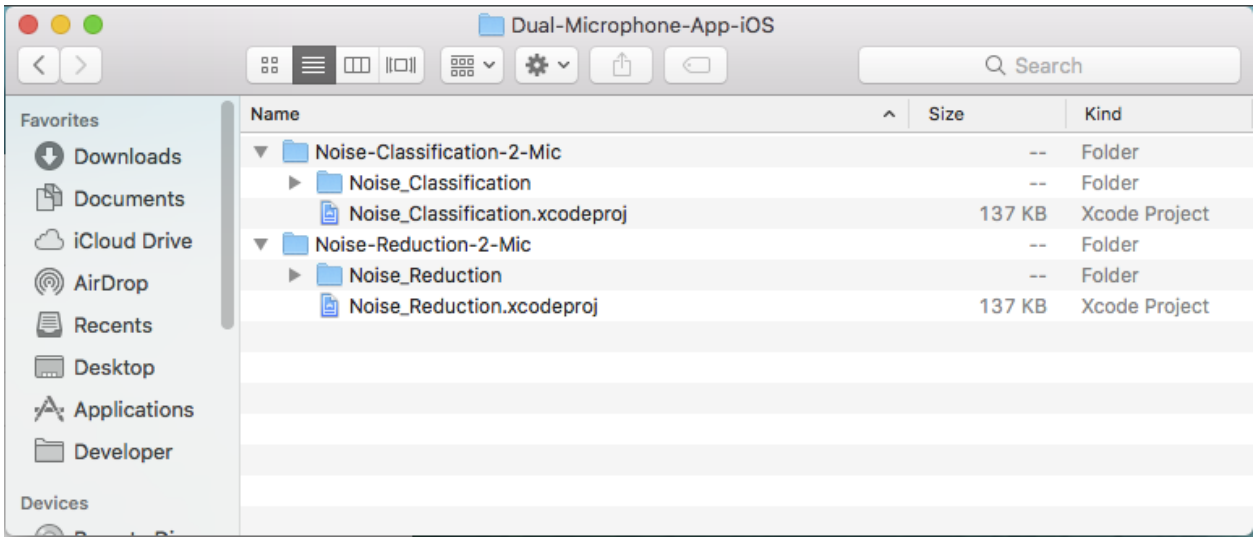

Dual-Microphone-App-iOS
Search
Favorites
Downloads
Documents
iCloud Drive
AirDrop
Recents
Desktop
Applications
Developer
Devices
Name
Size
Kind
Noise-Classification-2-Mic -- Folder
Noise_Classification -- Folder
Noise_Classification.xcodeproj 137 KB Xcode Project
Noise-Reduction-2-Mic -- Folder
Noise_Reduction -- Folder
Noise_Reduction.xcodeproj 137 KB Xcode Project

Fig. 2

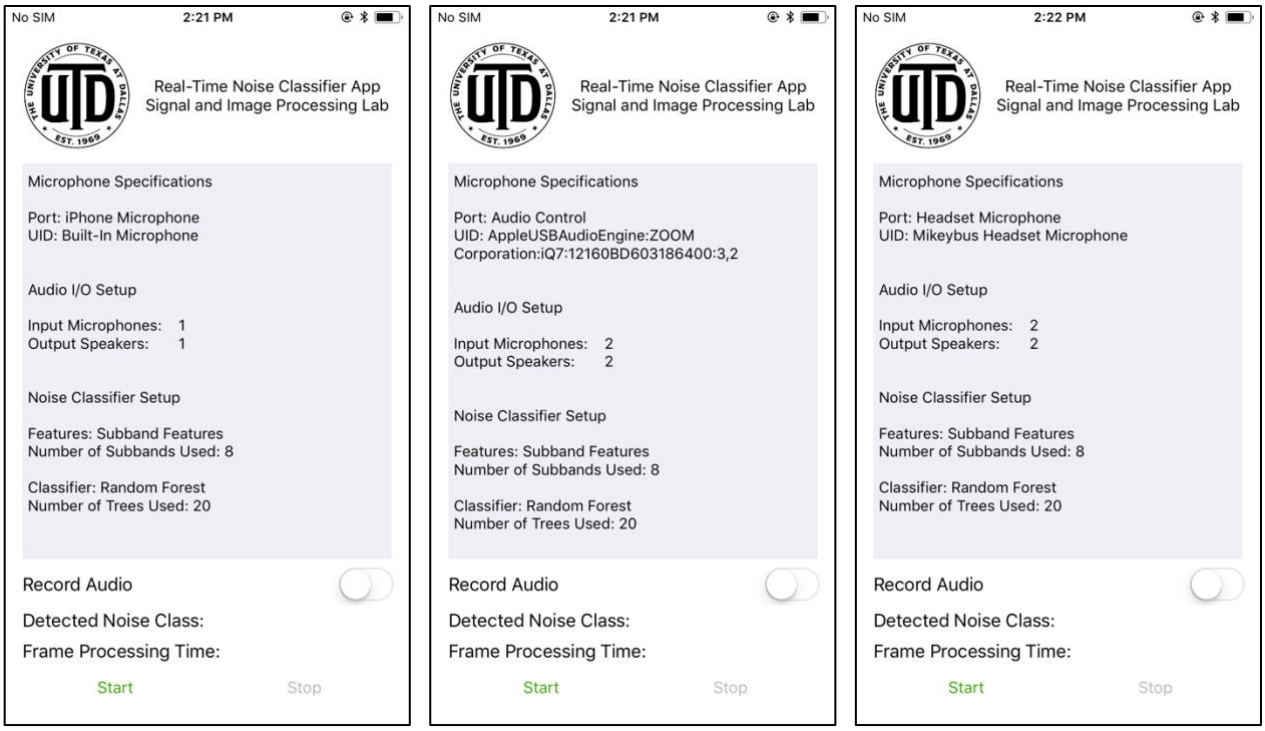
Part 1

Two Microphones

Noise Classification App

Section 1: Noise Classification App

Fig. 3 below shows the main screen of the Noise Classification app when the external microphone is attached and without it.



No external microphone attached | Zoom iQ7 attached | Sennheiser Ambeo Smart Headset Attached

Fig. 3

As can be seen from the GUI, the microphone specifications change depending on the external microphone attached. NOTE: When changing the microphone, it is necessary to restart the app.

An option to record audio is also available by turning on the “Record Audio” switch. The fields “Detected Noise Class” and “Frame Processing Time” display the detected noise class and frame processing time, respectively. The app is started by pressing the “Start” button and stopped by pressing the “Stop” button.

Section 2: Code Flow

This section states the app code flow. The app is designed to make the components modular and the code blocks or modules can be easily replaced. One can view the code by running “Noise_Classification.xcodeproj” in the folder “Dual-Microphone-App-iOS” as shown in Fig. 4. The code is divided into 3 sections as indicated in the project navigator:

- View
- Algorithm
- AudioIOCode

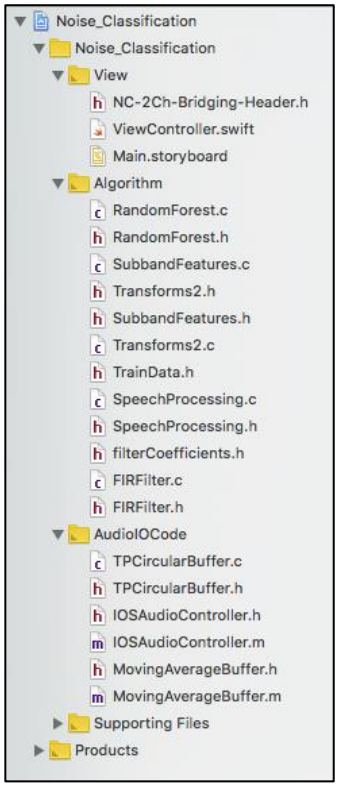


Fig. 4

2.1 View

This involves the app GUI. The initial setup of the GUI along with the screen interaction methods, e.g. button touch, slider adjust, etc., are mentioned here. The GUI consists of 3 components:

- **Main.storyboard**: This component denotes the layout of the GUI.

- **ViewController.swift**: This component connects the GUI elements to actions. It is coded in Swift.
- **NC-2Ch-Bridging-Header.h**: This component is a bridging header that allows calling the Objective-C functions declared in Audio IO in the Swift ViewController file.

2.2 Noise Classification Algorithm

The feature extraction code processes each incoming frame and outputs to which noise environment it belongs. It is divided into the following components:

- **SpeechProcessing**: This component is the main component of the native code. It initializes all the settings and then assigns how each module should be called in order to classify audio frames. The following modules are used in the noise classification:
    - **FIRFilter**: This module is used to bandlimit the incoming and outgoing audio signal.
    - **Transforms2**: This module computes the incoming FFT of audio frames.
    - **SubbandFeatures**: This module uses the FFT to extract the subband features. The number of bands to be used can be set in the code. It is either 4 or 8.
    - **RandomForest**: This module classifies the noise environment based on the subband features.
    - **filterCoefficients**: These are the filter coefficients for the FIRFilter module.
    - **TrainData**: This module is the random forest classifier designed to carry out the classification. It is obtained by performing training in MATLAB on actual collected data.

2.3 AudioIOCode

The Audio IO section of the code handles the initialization of the audio setup of the app and the synchronous callbacks that collect and transmit audio from the smartphone:

- **MovingAverageBuffer**: This is a multi-purpose Objective-C class that creates an object to calculate the moving and total average of a data stream. In this app, it is used to average the processing time and noise classification output to display on the GUI.
- **IOSAudioController**: This is the main component of the Audio IO. This class creates the audio i/o setup and initiates the input and output synchronous callbacks. The audio setup is designed to collect and play audio at the rate of 64 samples at 48 kHz in order to have the lowest latency.
- **TPCircularBuffer**: This is a data structure that synchronizes the audio i/o with the audio processing framework. The noise classification is a multi-rate signal processing

application and the circular buffer used enables synchronization of the modules. The input circular buffer collects 64 samples from the microphone till 12.5 ms or 600 samples of audio data are collected. These samples are then processed and fed into the output circular buffer and played back via the speaker/headphones at the rate of 64 samples.

Section 3: Two-Microphone Setup

3.1 Audio Setup

To enable two-microphone audio I/O, in the IOSAudioController.m file, it is required to setup the “AudioStreamBasicDescription” for two channels per frame. This allows recording interleaved data which can then be processed as two separate independent channels.

3.2 Algorithm Setup

The algorithm processes each stream of the stereo input independently. The noise classifier gets the probability of each class from both streams and then the class with the highest probability after fusion is selected.

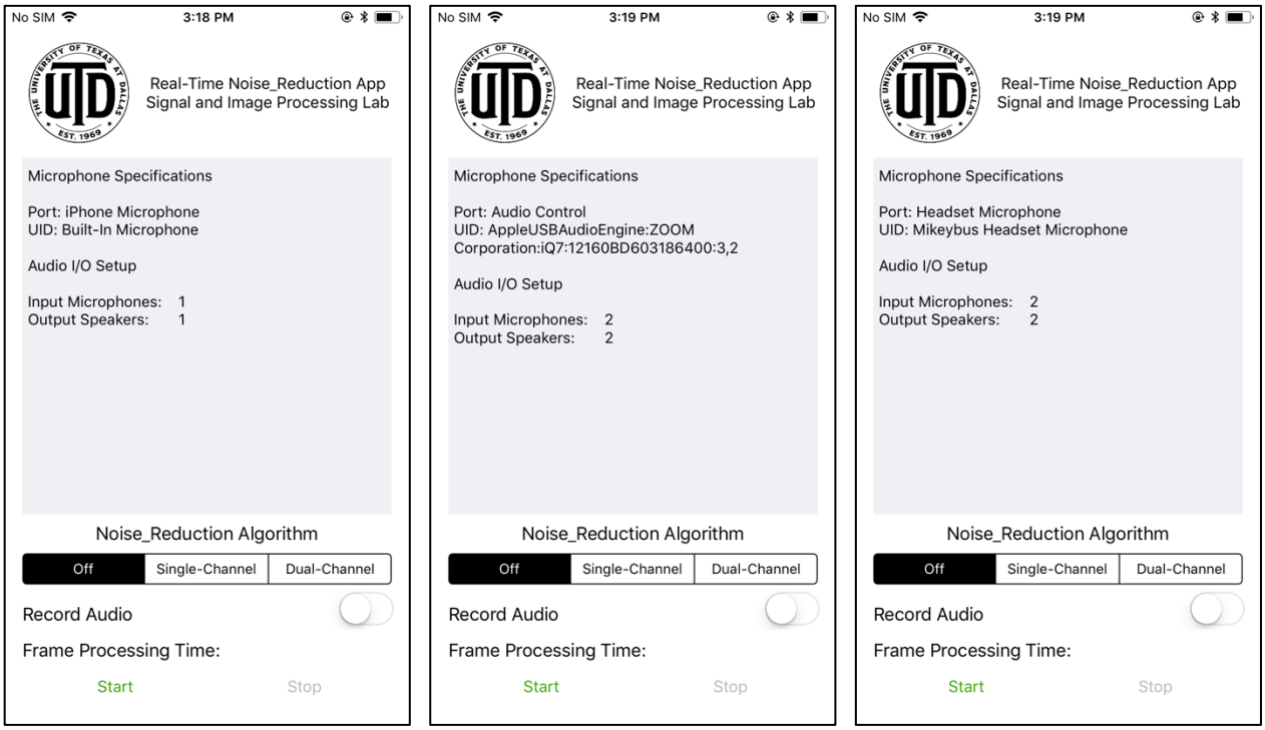
Part 2

Two Microphones

Noise Reduction App

Section 1: Noise Reduction App

Fig. 5 below shows the main screen of the Noise Reduction app when the external microphone is attached and without it.



No external microphone attached | Zoom iQ7 attached | Sennheiser Ambeo Smart Headset Attached

Fig. 5

As can be seen in the GUI, the microphone specifications change depending on the external microphone attached. NOTE: When changing the microphone, it is necessary to restart the app.

An option to record audio is also available by turning on the “Record Audio” switch. In the “Noise Reduction Algorithm”, the execution can be done without noise reduction, with the single channel algorithm (“Single-Channel”), and with the dual-channel algorithm (“Dual-Channel”) which uses both microphones to boost the SNR. The app is started by pressing the “Start” button and stopped by pressing the “Stop” button.

Section 2: Code Flow

This section discusses the app code flow. The app is designed to be modular allowing the code blocks or modules to be easily replaced. One can view the code by running "Noise_Reduction.xcodeproj" in the folder "Dual-Microphone-App-iOS" as shown in Fig. 6. The code is divided into 3 sections as indicated in the project navigator:

- View
- Algorithm
- AudioIOCode
- MATLAB Coder

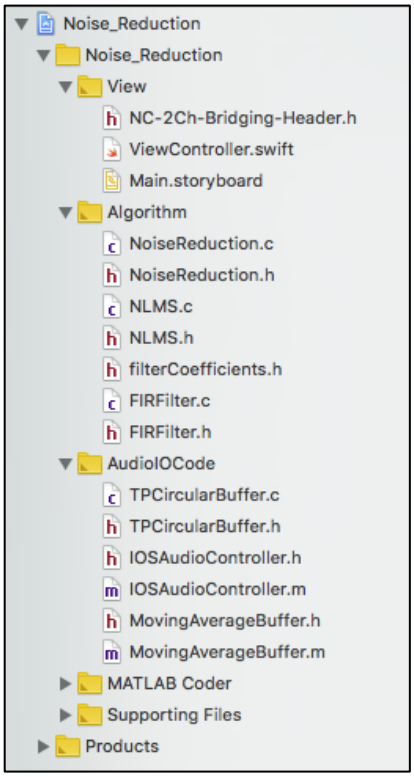


Fig. 6

2.1 View

This involves the app GUI. The initial setup of the GUI along with the screen interaction methods, e.g. button touch, slider adjust, etc., are mentioned here. The GUI consists of the following three components:

- **Main.storyboard**: This component denotes the layout of the GUI.
- **ViewController.swift**: This component connects the GUI elements to actions. It is coded in Swift.
- **NC-2Ch-Bridging-Header.h**: This component is a bridging header that allows calling the Objective-C functions declared in Audio IO in the Swift ViewController file.

2.2 Noise Reduction Algorithm

- **NoiseReduction**: This is the main component of the native code. It initializes all the settings and then assigns how each module should be called in order to process audio frames. The following modules are used for the noise reduction:
    - **FIRFilter**: This module is used to bandlimit the incoming and outgoing audio signal.
    - **NLMS**: This module boosts the SNR of the incoming audio frames.
    - **filterCoefficients**: These are the filter coefficients for the FIRFilter module.

2.3 AudioIOCode

The Audio IO section of the code handles the initialization of the audio setup of the app and the synchronous callbacks that collect and transmit audio from the smartphone:

- **MovingAverageBuffer**: This is a multi-purpose Objective-C class that creates an object to calculate the moving and total average of a data stream. In this app, it is used to average the processing time to display on the GUI.
- **IOSAudioController**: This is the main component of the Audio IO. This class creates the audio i/o setup and initiates the input and output synchronous callbacks. The audio setup is designed to collect and play audio at the rate of 64 samples at 48 kHz in order to have the lowest latency.
- **TPCircularBuffer**: This is a data structure that synchronizes the audio i/o with the audio processing framework. The noise reduction is a multi-rate signal processing application and the circular buffer used enables synchronization of the modules. The input circular buffer collects 64 samples from the microphone till 12.5 ms or 600 samples of audio data are collected. These samples are then processed and fed into the output circular buffer and played back via the speaker/headphones at the rate of 64 samples.

2.4 MATLAB Coder

The MATLAB Coder section contains the MATLAB code converted to C using the MATLAB Coder. This conversion is covered in the Noise Reduction User's Guide which is provided on the project website.