Introduction

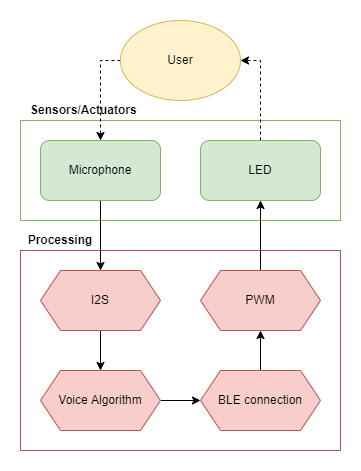
Two ESE students wrote a voice feedback program during their 4th semester project “Activating Parkinson Patients with Sensors”. This document contains relevant information intended for those that wish to understand how this program is structured.

Detailed information is given on the following components integral to our program:

1. BLE
2. INMP441 Microphone
3. Voice Algorithm

There are other components which are not mentioned because of their lack of complexity.

Here is a diagram that shows the high-level interconnection of the components we just mentioned:

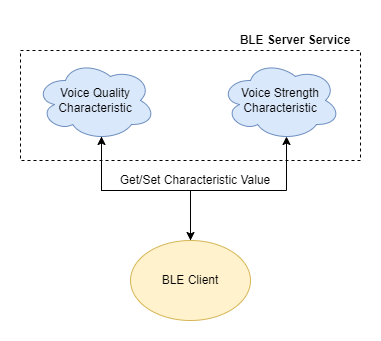


BLE

BLE was chosen for this project because of its edge in power consumption and development simplicity.

For this project we use just one service, which contains two characteristics representing the quality of the voice and the strength of the voice. There are two extra characteristics in the same service that are unused. Further there is one more service which is not in use and which contains four characteristics, this is purely to give you an example on how to add more services and/or more characteristics.

Any BLE client can set or get these characteristics on/from the server. The following diagram shows this simple interaction:



The UUIDs of the main service and its characteristics can be found either in the BLE.cpp or in the UUIDs.h file. As of 28th of June 2022, the UUIDs are:

**Service UUID**: 19B10000-E8F2-537E-4F6C-000000000000

**Voice Quality Characteristic UUID**: 19B10000-E8F2-537E-4F6C-111111111111

**Voice Strength Characteristic UUID**: 19B10000-E8F2-537E-4F6C-222222222222

INMP441 Microphone

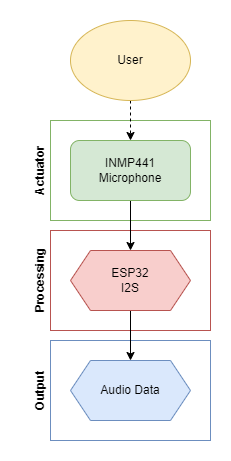
The INMP441 module is a simple I2S microphone breakout board, in this project the microphone is ran at 16KHz sampling rate and any pins may be used for I2S. As of 28th of June 2022, ESP I2S0 channel is used and the pins are configured as the following:

**BCK**: I/O 26

**WS**: I/O 25

**DATA**: I/O 33

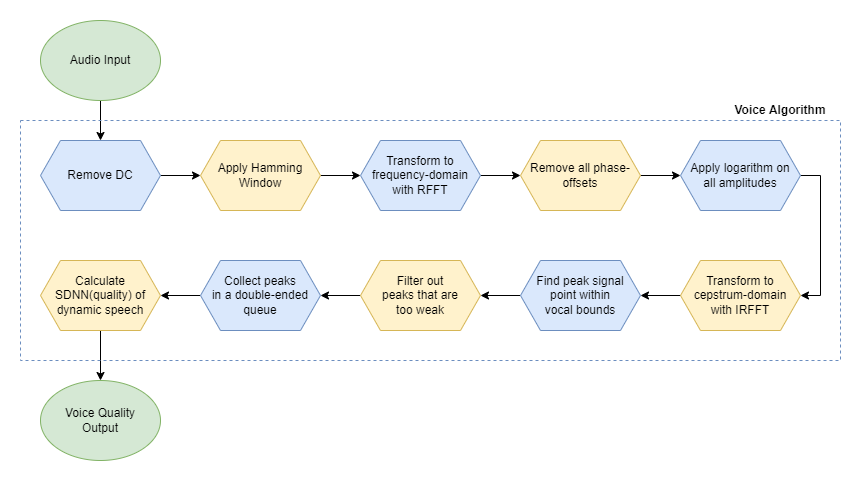
The microphone component I/O is shown in the following diagram:



Voice Algorithm

Jeroen Veen, an ESE teacher at the HAN, has been working for a while now on a speech algorithm for this project. We have successfully finished and tested his prototype algorithm, which remains to be further optimized and advanced.

The I/O of this component is shown in the following diagram:



A short explanation of each sub-component is given:

**Remove DC**: In the real world, microphones are not perfect and produce some type of DC noise in audio data. Here we remove this DC component so our audio becomes pure AC.

**Apply Hamming Window**: To analyze the input audio we must cut it into same-length “windows”. Since we intend to go into the frequency-domain using a Fourier-transform, we must ensure that the beginning amplitudes and the ending amplitudes of a “window” meet each-other smoothly at approximately the same height. This prevents various abnormalities in the frequency-domain.

**Transform to frequency-domain with RFFT**: Here we transform from time-domain into the frequency-domain where we can get the amplitude and phase of each sinusoidal frequency present in our input audio.

**Remove all phase-offsets**: Sinusoids may be delayed or ahead relative to each-other or relative to t = 0s. Here we simply remove all phase-offsets thus making each sinusoid start from t = 0s.

**Apply logarithm on all amplitudes**: Here we simply apply logarithm on the amplitude of each sinusoidal frequency present in our signal. Essentially what we are doing here is making the differences between our amplitudes smaller.

**Transform to cepstrum-domain with IRFFT**: Using IRFFT we transform back into the time-domain, but because we removed all the phase-offsets and applied logarithm on all the amplitudes, we are now in the cepstrum-domain. Here we can find the relevant quefrequencies(fundamental frequencies) that we need further in our algorithm.

**Find peak signal point within vocal bounds**: The fundamental frequency of the human voice has limits to it, which is especially apparent when comparing male and female gender. In this project we used vocal bounds for men: 50Hz minimum and 175Hz maximum. In this step we simply search through 50Hz to 175Hz and find which base frequency is strongest.

**Filter out peaks that are too weak**: Sometimes the signal is over-all too weak, so even though we just got the strongest vocal fundamental frequency peak of our audio signal, it may still be part of noise, thus we must filter it out. For our project we filter out any fundamental frequency peaks having an amplitude less than 0.15f (f = floating point notation).

**Collect peaks in a double-ended queue**: Here we collect all ours peaks in a deque(double-ended queue), which has a size limit to it. In our project the size limit for this deque is set at 2 seconds of fundamental frequency peaks. If you lower this limit then the feedback becomes more responsive but less accurate. On the other hand, if you increase this limit, then the feedback becomes less responsive but more accurate.

**Calculate SNN(voice quality) of dynamic speech**: Finally, we look at the variation in the fundamental frequency peaks that we have collected and calculate the SDNN(standard deviation) of those peaks which we use as a proxy for dynamic speech quality.

Miscellaneous

**Logging**: You can place ESP\_LOG functions directly in the source code to log what is happening in the program in real-time.

**Usage without BLE**: If you want to use the speech analysis module with BLE, then simply comment out “setup\_ble();” in **main.cpp** and “try\_send\_voice\_values("0", quality\_value);” in **main\_tasks.cpp**.

**Compilation and IDE**:

**The BLE server and the LED client**: Programmed without any extra options in Arduino IDE.

**The speech analysis client**: Programmed in Espressif IDE plugin for Eclipse with extra CMake options. ESP-IDF 4.4.1 was used in this project, though we do not expect future updates to break compatibility with our program.

The CMake options in the source folder are:

Graphical user interface, text, application

Description automatically generated

The CMake options in the root folder are:

Graphical user interface, text, application

Description automatically generated

The ESP32 menuconfig(“sdkconfig”) is present with the source code.

An extra Arduino component is required to compile the speech analysis client source code:

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

This component may be found here: <https://github.com/espressif/arduino-esp32>