Boston University
Electrical & Computer Engineering
EC463 Capstone Senior Design Project

First Prototype Test Report

LungDetect:
Acoustic Imaging Device for Diagnosing Pneumonia

by

Team 21

Team Members:

Thinh Nguyen tnguy19@bu.edu
Matthew Pipko mattpi23@bu.edu
Hilario Gonzalez hilario@bu.edu
Astrid Mihalopoulos astridm@bu.edu
Shadin Almaiman shadin@bu.edu

I. Testing Summary

a. Hardware setup:

The steps to set up the hardware for prototype testing are as follows:

1. Download and initialize Audacity and VoiceMeeter (for Windows). Ensure that they can recognize input and output audio devices.
2. Connect both soldered, piezo contact microphones to the male-to-female, USB-C to 3.5mm aux jack dongles.
3. Connect the male USB-Cs of the microphones to a USB-C splitter hub, which is then plugged into the computer responsible for recording the audio. Ensure the OS reads both microphones as input devices, not output devices. The hardware was tested on Windows.
4. Open VoiceMeeter and assign the left microphone to output A1, and then assign the right microphone to output A2. Pan A1 and A2 through the left channel and the right channel of the virtual output B1 respectively.
5. Open Audacity, and select the recording device to be VoiceMeeter's B1 virtual output. Ensure that B1 is a stereo (2-channel) audio input device
6. Start recording on a stereo track, and tap the microphones and/or the surface the microphones are on.
7. When finished recording, export the recorded audio as a stereo .wav file for software analysis.

b. Software setup:

The steps to set up the software for prototype testing are as follows:

1. Clone the repository and all the code files from Github from the appropriate branch.
2. Ensure all initial dependencies are installed, particularly Node Js, NPM, and all Python libraries are properly configured using either a global or anaconda environment. Set up based on the machine OS (Mac, Linux, Windows, etc)
3. Ensure all dependencies for the React front-end web app and Express backend are installed and correctly configured. This is done by running the command 'npm install'
4. Start server hosting React front-end app on local machine with 'npm start' command on the command line, by default hosting at the address: localhost:3000.
5. Start server hosting Express backend-end on local machine with 'node server' command on the command line, by default hosting at the address: localhost:5000. Ensure the command is run at the appropriate folder/directory on the project's folder on the local machine.
6. Ensure the front-end app correctly boots up as an automatic web page that appears if started up correctly. The backend app should return the output 'Server listening on port 5000' on the terminal of the IDE or command line if it boots up correctly.
7. Ensure there is enough space for uploaded files to be stored on 'backend/public/data/upload/' as this will be where uploaded sound files will be stored.

8. Upload the .wav files using the UI interface, the app will automatically process the uploaded file and produce a visualization of the data.

II. Measurements and Test Data

1. Hardware:

- The live prototype test .wav file was successfully recorded, exported, and interpreted as two separate channels on the software interface.
- Both microphones were initialized as 1-channel audio input devices.
- Both microphones were able to record sound individually and simultaneously while plugged into the same computer.

2. Software:

- Good performance, no issues with setting up the project on a local machine with Github repo. All dependencies are easily installed and configured on testing machines running on MacOS.
- Quick install and configuration of project dependencies (done in less than 2 minutes in total excluding time taken to clone the repo from the Internet). No major or minor bugs were encountered.
- Visualization result returned with no manual input or manual command from the user as expected.
- Correctly receives uploaded .wav file and produces the result, satisfactory result, and speed as noted by professors observing the demo. The total time taken from file upload to displaying the results is less than 1 minute.

3. Specific Results Regarding Data Analysis Script:

- Successfully interprets Audio file into an array of samples and identifies samples where the amplitude exceeds a typical range.
- Assigns a cluster ID to samples within a desired range of each other and filters and filters out clusters that likely represent false positives.
- For every cluster, and for every channel in the cluster, it successfully identifies the time index of the highest amplitude, and this time index is used to identify each channel in each cluster.
- For every cluster, it successfully identifies the leading mother crackle and calculates a cross-correlation with all the daughters, which is used to calculate a delay and transmission coefficient. Then, this is returned to the server to be interpreted for a visualization.
- The test demonstrated success in executing the data processing and returning a result at nearly instant speed.

III. Conclusion

Regarding hardware, both microphones were successfully soldered and working properly on the Windows laptop. The microphones still have issues being read as input devices, not output devices on Mac OS, so it limited hardware testing to Windows computers only. VoiceMeeter was able to pick up both microphones as input devices and was able to pan them separately into different channels on one virtual output. Audacity recorded the vibrations caused by tapping the microphones on the table, showing which microphone was picking up the tappings the loudest at a given point in time, then was able to successfully export the recorded sounds as a stereo .wav file.

One issue was that one microphone was picking up less sound than the other microphone, but that might be a result of several different causes such as inconsistent wire lengths, wire soldering, and damage from heavy use. These factors will be kept in mind when the next iteration of hardware is being produced. Another aspect that needs to be addressed is the potential lag between the microphones. To test for lag, plug in both microphones and record a 2-channel audio track while tapping on a surface equidistant from both microphones. Once the tapping is recorded, the time delay can be tested by looking at the waveforms of the channels, finding common peaks in sound, and then averaging the time delay between all of the common peaks. This delay is then applied to the faster microphone, which delays its recording start, but it ensures that any recorded peaks are concurrent on both channels.

Regarding software, both the frontend and backend (including the Express Js and Python script) were easily and correctly configured and set up, and there were no bugs recorded or encountered during the testing demo. Minor delays include time to clone the repo from the Internet or computation time being negligible, with total time from file upload to result visualization negligible and noted as satisfactory by professors observing the demo (less than 1 minute in total).

Regarding the data analysis script, it was successful in producing the desired results for a variety of audio inputs. The only requirement is that the input audio file is in a multi-channel format, which is natively supported by a file format like ‘.wav’. The script features two tunable parameters which allow it to be adjusted for different shapes of input data. These parameters are the time threshold for a cluster, and the length of slices that are being cross correlated. For example, we anticipate that in a real world scenario, where sound travels at approximately 1500 m/s inside the human body, the delay between a unique sound source reaching one mic versus another is going to be on the order of milliseconds. This means that the threshold of time to be considered a cluster must be very narrow, and if two channels spike outside of this narrow range, we can’t assume that they are picking up the same source of sound. However, with the test audio file found online, all 8 channels have a single crackle about a second apart, so this cluster

threshold was increased to 9 seconds for the sake of verifying that the python script works. Similarly, in a real world scenario the length of a crackle will be very short, so when we isolate a slice of audio to cross correlate, we want the slice to be long enough to encompass the full crackle, but not so long that noise from previous or successive breaths gets included. However, again for the test audio, since the crackles are so spread out, this slice length also had to be increased. In conclusion, the audio processing script works as desired, and the adjustability of these parameters will allow us to fine tune it to work best in the real world as we continue developing the device to generate real world data.