Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

Final 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. Make sure there are proper connections with all the cables and dongles.
2. Open n-Track and open the audio device settings.
3. Connect each microphone to the USB splitter hub, one-by-one, monitoring the device name of each microphone.
4. Set each microphone as an MME (Microsoft Multimedia Environment, or alternatively, any viable audio driver in the user's PC) input and output device, based on the order that they were plugged into the hub. After setting the input devices, n-Track will automatically set the buffer settings (buffer size: 4800, # of buffers: 4)
5. Open the input VU meter to check the microphones and ensure that all 6 are picking up signals.
6. Apply these equalization effects to each microphone: high pass filter at 24 Hz

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, mainly Node.js, NPM, and all Python libraries are correctly configured using either a global or Anaconda environment. Set up based on the machine OS (Mac, Linux, Windows, etc)
3. Ensure all React front-end web app and Express backend dependencies are installed and correctly configured. This is done by running the command ‘npm install’
4. Start the server hosting the React front-end app on a local machine with the ‘npm start’ command on the command line. By default, the server is hosted at the address localhost:3000.
5. Start the server hosting Express backend on the local machine with the ‘node server’ command on the command line. By default, the server is hosted at localhost:5000. Ensure the command is run in the appropriate folder/directory in 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 it is started up correctly. If it boots up correctly, the backend app should return the output ‘Server listening on port 5000’ on the terminal of the IDE or command line.
7. Ensure there is enough space for uploaded files to be stored on ‘backend/public/data/upload/’, as this is 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 and a data table displaying the data picked up by the hardware.

II. Measurements and Test Data

1. a. Hardware Results:

- All six microphones were recording sound simultaneously.
- Labelled microphones correspond with the software
- Data was shown in real-time recording in digital format on the computer
- Consistent results were obtained while recording multiple single-channel files versus a singular multichannel file
- The sound waves on each recorded track were visually similar to each other in n-Track's visualization.

b. Software Results:

- Software processing speed < 2 min
- Quick installation and configuration of project dependencies (done in less than 2 minutes, excluding time taken to clone the repo from the Internet). No major or minor bugs were encountered.
- The visualization result was returned with no manual input or command from the user, as expected.
- Correctly receives the uploaded .wav file and produces a satisfactory result and speed, as noted by professors observing the demo. The total time from file upload to displaying the results is less than 2 minutes.
- Correctly handle .wav data consisting of 6 audio channels; output in terminal indicates correct parsing of data into JSON format to be used
- Script correctly processed and displayed data on lung visualization and data table correctly, but correctly used the unprocessed data as value for sound playback functionality
- The app can correctly playback audio from each channel when the corresponding button is pressed
- Correctly identify the channel location where the sound is detected, clearly indicate the diagram and data table result for each crackle family, and the data from each channel and each crackle family.

III. Conclusion

Regarding the hardware, the device was able to successfully allow for simultaneous recording across all 6 microphones. What was especially important for this project was making sure that the microphone indexing stayed consistent across both the hardware and software. When stimulated by tapping, each microphone responded to its assigned number when performing real-time monitoring on n-Track. Before recording, n-Track is adjusted for the correct order of

channels, which is done by individually plugging in the microphones one at a time. Each microphone is assigned to a corresponding track that will be numerically assigned to its physical position when exported. Post-processing effects are also applied to minimize interference from noise via a band pass filter and compression**.** Afterwards, the audio is easily exported into .wav files to be uploaded onto the software interface.

During the demonstration, there were two different test files used. The first was recorded by using the device hardware. This was done to test the calibration process. One of the group members, Hilario, tapped on the microphones in order of track number. The recorded taps were exported as 6 separate .wav files (each representing a separate channel/microphone) that were to be uploaded onto the interface. The interface is coded to be able to take in either one singular multi-channel file or up to 6 single-channel files. In this case, each .wav file was compiled into one six-channel .wav file. After uploading this file, a visualization was successfully outputted that showed all the taps as crackle families, as well as an indicator of which microphone was closest to the source of the noise. Each crackle family was in order of microphone location, given that Hilario was tapping the microphones in that order. This entire process was repeated with test files obtained from an online lung database (ICBHI 2017 Challenge). Using this database, a single two-channel audio file was created via recordings from the posterior left and right of a pneumonia patient's back (106_2b1_Pr_mc_LittC2SE, 106_2b1_Pr_mc_LittC2SE). Once this .wav file was uploaded, a visualization was uploaded (showcasing up to 30 crackle families), showcasing the closest microphones, the delays between each, as well as their transmission coefficients.

Major changes were also made to the software since the last test that we conducted. The changes happen in all three different parts of the app: the frontend/UI, Node.js backend server, and the Python data processing script.

The frontend built with React receives a big update on the visual side as we incorporated frameworks such as Bootstrap and third party libraries such as Bootswatch to give the app a modern and more visually appealing look. The app are also better formatted with a header bar, tabs, and utility buttons to improve the user experience and makes the app more intuitive to use. Major updates are also made to how the data visualizations are displayed, moving from a simple lung image to a more comprehensive diagram completed with data tables clearly displaying the data at every channel and crackle families. As a result the UI now meets the standard and expectation of users similar to what is seen in modern web apps. The app also now features the audio playback capability which is implemented on the frontend using the Web Audio API. This helps the frontend app able to execute playback efficiently using data passed to it from the backend.

The backend Node.js server also saw a change in its capabilities, particularly with the addition of scripts aimed at handling different types of file upload and utility script. In particular it can now be accessed not only on the local machine that is hosting but also on the local network providing easy access to the application. This prevents the difficulties that we encountered where only one machine can access the same iteration of the app in earlier test runs. Secondly, the addition of scripts is also reflected in the Node.js which is able to correctly execute the script when needed, which is a big update compared to the old version of the app where we only have 1 processing script to be executed by the Node.js server.

We overhauled the audio analysis system completely for this test. The Python data-processing script ingests a multichannel WAV file, computes a rolling median + MAD noise threshold per channel and flags high-energy "spikes". It groups temporally proximate spikes across channels into crackle families, retains the loudest spike per channel, and extracts short waveform windows around each event. Pairwise cross-correlations of those windows identify the channel with the strongest summed correlations as the family's acoustic "leader." Final per-channel metrics are then calculated: the raw delay is simply each spike's timestamp minus the leader's, while a transmission-coefficient is the ratio of cross-correlation peak energy to the leader's autocorrelation peak. These delays are finally corrected with per-microphone offsets supplied by the calibration routine, which was another significant addition for this test demo, and all results are returned to the Node.js backend as clean JSON for display.

The calibration script derives those per-microphone timing offsets. Starting from a fixed 6×6 distance matrix of the microphone array, it converts distances to theoretical propagation times using the speed of sound. A second 6×6 matrix of measured delays from a tap-test recording is loaded; each entry equals the true propagation time plus an unknown electronic/placement offset from the two microphones involved. Stacking these relationships produces an over-determined linear system $A\delta=b$, which is solved by least-squares to recover the vector of offsets δ. The solution is re-referenced so microphone 0's offset is zero, and the resulting array is returned to the main processing script, enabling hardware-induced skew to be removed from all subsequent delay estimates.

One of the python scripts that was added was to accommodate the fact that users can choose to upload either a single multi-channel .wav file or multiple single-channel .wav files. For the first option the backend server will trigger the normal data processing on the uploaded file directly, whereas in the second option a separate 'combine .wav files' script is triggered first to combine all incoming files into a single containing all the channels before it is passed onto the data processing script, effectively allowing the user to choose how to upload their data.

Overall, the test is a successful demonstration of the device capability to capture, process and display results in a short amount of time (< 5 minutes overall including collecting data to receive the visualization result). The app makes use of a combination of both third party softwares as recommended by the client and our own app implementation, combined with hardware design to produce a successful lung noise detection product.