Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

Final Testing Plan

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

Required Materials

Hardware:

1. Foam Enclosure
2. Six piezo contact microphones
3. Computer with USB-C slot
4. 3.5mm male plug to bare wire TRRS cord cable
5. Six female 3.5mm jack to male USB-C dongles
6. 6-in-1 USB-C splitter hub
7. n-Track Studio 10 for recording and exporting audio files using the microphones.

Software:

1. Front-end:
    - React app:
        - The app automatically starts running at the network address localhost:3000 on the device or at a specified IP address on the local network also at port 3000
        - Receives .wav data file input through upload (or from detection).
        - User Interface built and customized with JavaScript, HTML, and CSS.
        - Backend link using Axios
        - Frontend is also capable of executing sound playback picked up from microphones at specific channels
2. Back-end:
    - Express server:
        - Listens on port 5000 with 'app.post('/compute' ...') running in both localhost:5000 on the local machine and likewise at a specified IP address on the local network also at port 5000
        - Uses Multer to receive a .wav input from user upload (or in-app recording) and stores it in the application directory at './public/data/uploads/.'
        - Spawns a process running the data analysis script and passes the file name.
        - Sends formatted output from data analysis script back to front end for visualization.
    - Python data analysis script:
        - Interprets the .wav file into a matrix. Rows = number of channels, Cols = number of samples.
        - Stores all instances of audio spikes using a sliding window convolution that identifies spikes in local context.
        - Identifies spike families based on temporal proximity across channels.
        - Filters out families that don't have the appropriate number of channels represented.
        - It selects only the channel instance with the highest amplitude to index that spike for that channel for every remaining family.
        - The channel with the highest cross-correlation transmission is selected as the mother, and every other channel is a daughter. Every channel is

cross-correlated with the mother to find how much the sound lagged on the daughter channel and to calculate a transmission coefficient.
    - All of these results are returned in an array.
- Calibrate script
    - Has a distance adjacency array for every mic in our device.
    - The speed of sound is set as a parameter. For now, we have set it to 343 m/s, the speed of sound through air, however, it would be ideal to be able to measure the speed of sound through the foam to get a more accurate calibration measure.
    - We calculate an adjacency matrix for the amount of time it would take a sound to reach from any one microphone to any other microphone, called calculated delay.
    - We make the assumption that the observed delay, output by our audio processing script, for a sound originating at microphone *a* and being heard by microphone *b* is a sum of the true delay due to the distance between the microphones and the artificial delay due to lack of synchronization on microphones *a* and *b*.
    - We record a calibration test where we play a sound starting at every mic that is picked up by every other mic. The output delays for this file are input as an adjacency matrix of observed delay.
    - Taking the difference of the calculated delay and the observed delay gives a matrix of sums of the two component artificial delays
    - These artificial delays are the unknowns we want to solve for in order to calibrate the output.
    - We set up a system of linear equations and solve for these unknowns.
    - The audio processing script receives these values and adjusts the channel time indices accordingly.
- Python .wav files combine script:
    - Takes in multiple .wav files corresponding to a single channel and combines them into a multi-channel single .wav file
    - Automatically passes this combined file on to the data analyzing script to be processed without further input by the user
    - Preserve the information of each .wav file and combine them without overlaps or corrupting the data

<u>Set Up and Pre-Testing Setup Procedure</u>

Server/Software Side:

1. Ensure all React front-end web apps and Express backend dependencies are installed and correctly configured.
2. Start server hosting React front-end app on the local machine with the ‘npm start’ command on the command line terminal, by default hosting at the address: localhost:3000. The terminal will output
   ‘ Local: http://localhost:3000

On Your Network: {IP ADDRESS OF HOSTING MACHINE}:3000'

3. Start server hosting Express backend-end on the local machine with the '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 in the project's folder on the local machine.
4. 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 running on port 5000
   Server is accessible at:
   - Local: http://localhost:5000
   - Network: {IP ADDRESS OF HOSTING MACHINE}::5000'
   on the terminal of the IDE or command line if it boots up correctly.
5. 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.

Hardware Side:

1. Make sure there are proper connections with all the cables and dongles.
2. Open N-Track and assign each microphone to its corresponding channel.
3. Test every microphone and ensure they can pick up a signal.

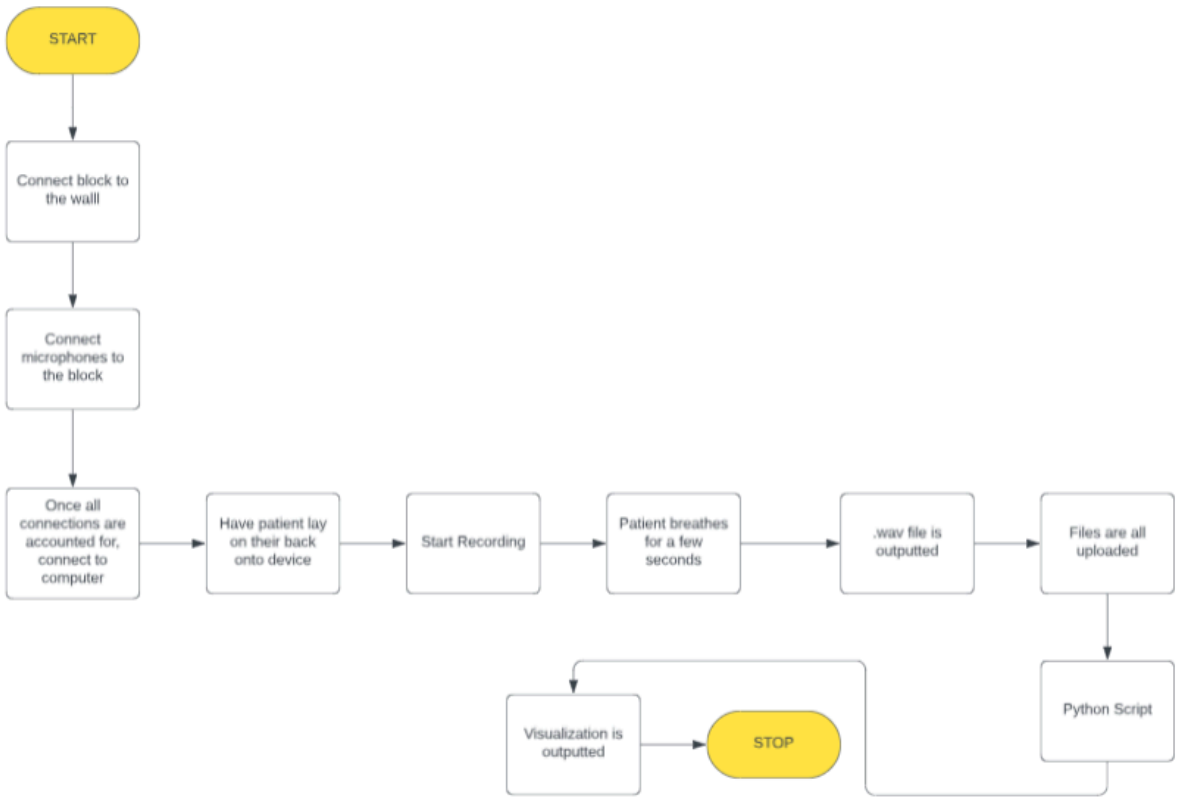

Figure 1: *Illustration of Process Flow*

Testing Procedure:

Hardware Side:

1. Place the device behind a testing subject's back, ensuring all microphones are in contact with the subject's back.
2. Briefly test the device's microphones by having the patient casually breathe in and out while recording.
3. Prepare the subject with a countdown before recording the data, ensuring they are ready.
4. Start recording on N-Track for a couple of minutes.
5. Stop the recording and export the recorded audio as a .wav file.

Software Side:

1. User uploads a single multichannel .wav audio file or multiple single-channel .wav audio files via the user interface in the corresponding tab
2. Press the 'Process File' button to initiate the analysis process
3. Identify crackle instances and calculate the delay on every channel.
4. The server sends results back to the front end.
5. The front end produces visualization.
6. Repeat the process with lung database .wav file for comparison

Measurable Criteria

Hardware Side:

1. The device must record a 6-channel sound file to be taken in by the PC for data processing.
2. Each microphone should record on a separate channel from the other in a stereo audio track.
3. Microphones should be able to detect sound while being embedded within the foam enclosure.
4. The sound waves on each recorded track should be visually similar in N-Track's visualization.

Software Side:

1. Software should be able to start both the frontend and backend servers running simultaneously.
2. React App capable of allowing users to choose and upload .wav file(s), then commence computation when the 'compute' button is pressed (no computation before or after this action should happen).
3. The software can receive and store .wav files in the project's directory on a local machine in the path 'backend/public/data/uploads/.'
4. The uploaded files should be able to be recognized and located by the backend Express app, which will call the Python scripts to run with the file. All of this will happen very quickly (almost in real time). Errors will be logged on the console terminal, and successful runs will print the data received on the terminal.

5. Computation and result visualization should happen nearly instantaneously, precisely no more than 2 minutes maximum. Also, the process should happen automatically with no further manual input from the user after the .wav file has been submitted.
6. 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.
7. App should be able to execute playback functionality when a button is pressed corresponding to the location of the microphones.

SCORE SHEET:

OBJECT	Correct? (Y/N)
Software processing speed < 2 min?	Y
Microphones can pick up sound?	Y
Expected consistent order of microphone recording vs. actual order?	Y
Are there consistent results in uploading multiple single-channel files versus a single multiple-channel file?	Y
Is delay between peaks < 3 ms?	Y
App able to perform audio playback of sounds detected by channels on demand?	Y