Boston University
Electrical & Computer Engineering
EC463 Capstone Senior Design Project

Second 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. Ensure all of the wires are connected correctly
2. Connect ESP32 to the computer using USB-C
3. Record audio by tapping on or around the microphone.
4. (Optional) If the firmware needs adjustment, or to visualize the monitor output, use the Express Integrated Development Framework with the esp32 plugged in. Type *idf.py flash monitor* to build any new code, flash, and monitor the firmware.

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:

- The microphone was able to record data in analogue
- Analogue data was successfully converted to digital data by the ESP32
- Data was shown recording real-time data in digital
- Switch was able to turn the recording on and off successfully

b. Software Results:

- Good performance. Setting up the project on a local machine with a Github repo was no issue. All dependencies were easily installed and configured on testing machines running on MacOS.
- 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 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 4 audio channels; output in terminal indicates correct parsing of data into JSON format to be used
- Changes to the React app successfully displayed the lung visualization diagram and the corresponding data table showing the exact data recorded from the hardware. Data is correctly presented with up to 2 decimal places, which show data values for Channel, Delay, and Transmission Coefficient

2. Specific Results Regarding Data Analysis Script:

- Successfully interprets Audio file into an array of samples and identifies samples with amplitude exceeding a typical range.

- It assigns a cluster ID to samples within a desired range of each other and filters out clusters that likely represent false positives.
- It successfully identifies the time index of the highest amplitude for every cluster and every channel in the cluster, which is used to identify each channel and cluster.
- For every cluster, it successfully identifies the leading mother crackle and calculates a cross-correlation with all the daughters, which calculates a delay and transmission coefficient. Then, this is returned to the server to be interpreted for a visualization.
- The test successfully executed the data processing and returned a result at nearly instant speed.

III. Conclusion

Regarding hardware, the printed circuit board was prototyped successfully using breadboard, and the final product will be designed with PCBs in mind. This was done by connecting a microphone to a breadboard consisting of an ESP32 microcontroller, a pre-amp circuit made out of an Op-Amp (LM386) with a set of capacitors and resistors, and an ADC (ADC0808CCN). Ideally, these components would make up the PCB. The PCB connects to a Raspberry Pi Pico that receives SPI data, and converts it into a .wav file. The PCB design aims to minimize the analog line travel distance and encapsulate the single channel functionality to achieve a more modular approach. This is so that noise can be reduced as much as physically possible. Ideally, one PCB would be assigned to each microphone. In total, there should be eight microphones embedded within the foam. The circuit will also be confined within laser-cut acrylic housing in order to optimize space, increase durability, and minimize distance. According to the second prototype's results, live-recording from a single microphone connected to the breadboard was successfully observed and recorded onto a computer with a fully functional switch button.

Regarding software, both the front-end and back-end (including the Express Js and Python script) were quickly and correctly configured and set up. No bugs were recorded or encountered during the testing demo. The app requires minimal time to configure and set up and is available in less than 2 minutes.

As for 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 natively supported by a file format like '.wav. ' The script features two tunable parameters that allow it to be adjusted for different input data shapes. These parameters are the time threshold for a cluster and the length of slices that are being cross-correlated. For example, it is assumed that in a real-world use-case, where sound travels at approximately 1500 m/s inside the human body, the delay between a unique sound source reaching one mic versus another will be in

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, one cannot assume that they are picking up the exact source of sound. However, with the test audio file sourced online, all 8 channels have a single crackle about a second apart, so this cluster threshold was increased to 9 seconds to verify that the python script works. Similarly, in a real-world scenario, the length of a crackle will be very short, so when a slice of audio is isolated for cross-correlation, the slice must be long enough to encompass the full crackle but not so long that noise from previous or successive breaths gets included. However, again, since the crackles are so spread out, this slice length also had to be increased for the test audio. In conclusion, the audio processing script works as desired, and the adjustability of these parameters for fine-tuning will get the device ready to work in the real world soon as the team continues developing the device to generate real-world data.