Auto-determination of proper usage of Acapella by detecting pitch in audio input

Sondre Skatter, Smiths Group Digital Forge, Dec 21 2017

BACKGROUND

Being able to provide training on proper usage of the Smiths Medical Acapella PEP device is critical when this product goes on sale over-the-counter in the near future. Only when used correctly does the instrument provide the intended health benefits, but without the feedback from doctors or other health workers, the patient may not obtain these benefits, and may also lose motivation.

A patient using the Acapella correctly, will exhale through the instrument to produce a a positive internal pressure of 10-20 cm water.

Preliminary investigations revealed a strong correlation between the audible frequency, or pitch, of the fluttering action when the devise is used, and the internal pressure, which we would like to predict. Thus, by recording and analyzing the audio during usage, one can envision providing real-time feedback to the user via a phone app or other computer device, which can direct and motivate the user towards correct usage.

This document describes briefly the algorithm for determining the pitch from the audio signal recorded via a phone devise. It also describes methods and results for a study where a diverse group of volunteers followed some specific guidelines on exhalation patterns and device settings to provide a dataset to develop and validate a pitch/pressure model. The results confirms the strong connection between the fluttering pitch and the internal pressure seen earlier, and paves the way for a prototype implementation and further testing aimed towards FDA approval.

METHODS AND MEASUREMENTS

INSTRUMENTATION

For the initial testing the Acapella was instrumented to allow for measurements of both flow and pressure. However, it was found that some of the instrumentation would interfere with natural functioning of the PEP device, and so, in the current instrumentation only pressure was measured. Also, there was no need for the flow data in the algorithm verification at this stage.

The pressure was measured by inserting a T-Piece between the mouthpiece and the Acapella device and connect this to a TSI Certifier FA Plus to measure the pressure. This instrumentation is exactly the same as is used in the Pressure Indicator devise that can be purchased along with an Acapella device (see Figure 1).

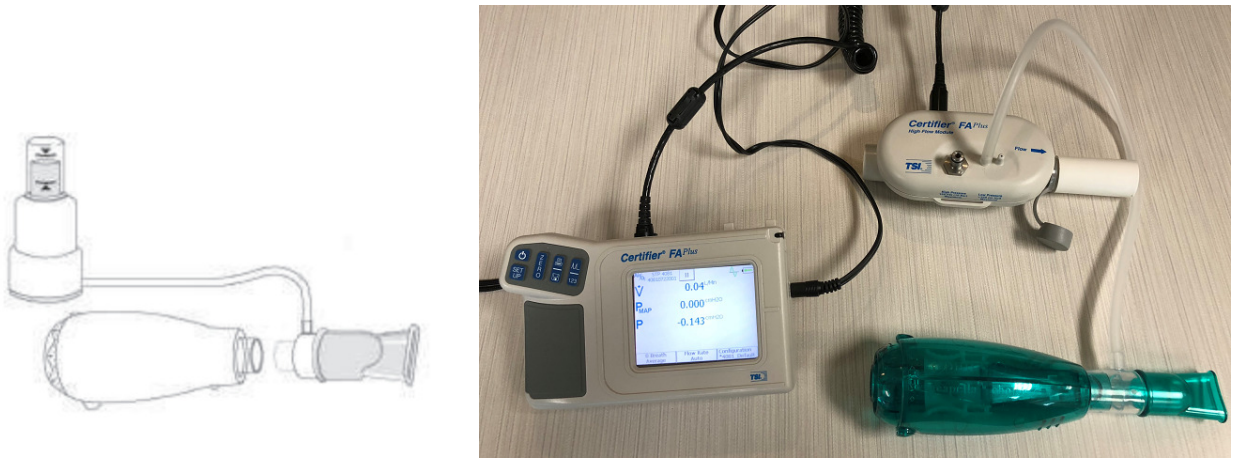

Figure 1. Left: Setup for attaching Acapella to a pressure indicator, which is the same coupling that was used in the present study (Right) to measure pressure with a TSI Certifier FA Plus.

As the test subjects exhaled multiple times through the Acapella devise, the pressure readings were logged at a 1kHz sampling rate and in parallel the sound was recorded by an iPhone SE devise.

The phone recording was done with the phone sitting at the table in the vicinity of the test subject but with no particular care being made to place it close to the Acapella device. This was done to mimic normal operating conditions and ensure robustness of the pitch detection algorithm.

TEST PROTOCOL

Nineteen people participated in the present study. Since the goal was to investigate whether there is patient-to-patient variation of the pitch/pressure relationship, great care was made to ensure diversity in the sample population, in particular to factors thought to may have an impact such as lung size/shape, general anatomy, and health. Figure 2 graphs the demographics for the test population.

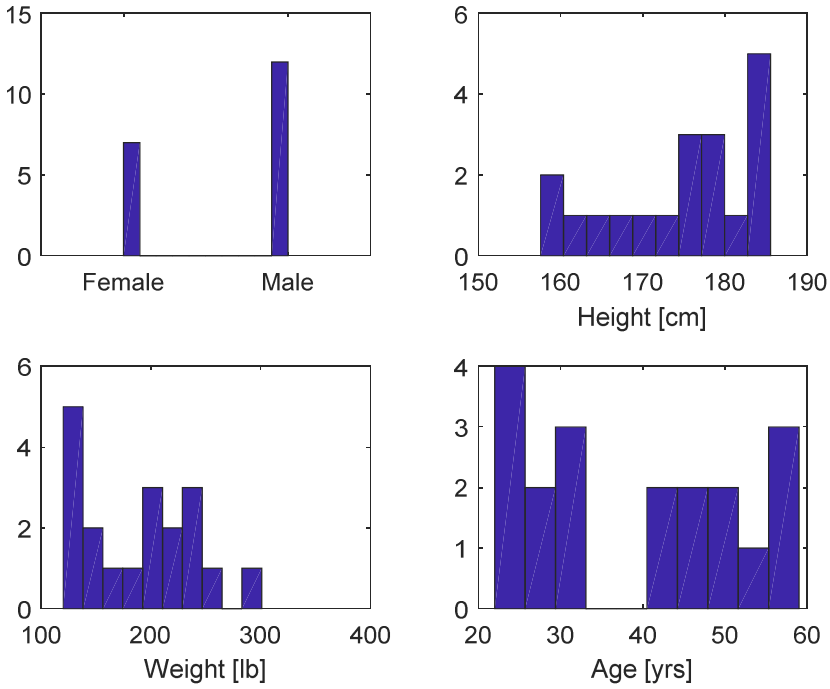

Figure 2. Demographics for the 19 people in the testing group. To make sure the pitch/pressure model holds up across the population it was important to achieve a large level of diversity despite the small sample size.

Each test person recorded 4 sessions, each lasting 15 seconds for the pressure measurements and a little longer for the sound recording. Typically, the tester would get 2-3 exhalation cycles done within these 15 seconds.

In the first session the resistance dial of the device was set to '1' and the test person was asked to exhale in accordance with the normal user guide. For the second session the user received the same instructions, but the resistance dial was switched to the '5' position. For the last two sessions the test person was asked to sweep from low to high pressure for each exhalation, first at resistance of '5' then at '1'. This allowed for creating a data sets where extremes along with normal usage were included.

TEST DATA PROCESSING

Time registration

In order to develop and validate the model to predict pressure from the fluttering frequency inferred from the audio, the two data streams – pressure data sampled at 1kHz and audio data recorded at 44.1kHz with an iPhone – needed to be registered in time (since it is impractical to start the data collection and the recording at the exact same time). The data streams were registered by correlating pressure with the square root of the audio energy. An algorithm was developed for this purpose and visually validated to perform as needed (bad registration results occurred for only one session, and this session was excluded from the data).

Time discretization

For the purpose of comparing paired observations of pressure with estimated pitch, the two signals were organized into discrete data points at every 0.05sec interval. These data points were used to develop the model and also to quantify and visualize the performance. For operational use, there is no need for this time discretization, as the only data to display should be the latest reading.

PITCH DETECTION ALGORITHM

The pitch detection algorithm developed to measure the fluttering frequency of the Acapella device during operation is based on autocorrelation in the time domain, operating upon a filtered signal that embodies the sound *energy* rather than the sound *amplitude*. Equal emphasis were placed on accuracy and computational speed. To illustrate the latter, the current C++ implementation can process 20 sec of audio feed in only 50msec, which provides a factor of 400 of margin for real time processing. Noting that the algorithm will run on a mobile device rather than the HP laptop used for this benchmark, this still makes a healthy margin.

The algorithm operates upon incremental chunks of audio data, typically a new chunk of the latest audio capture every 0.1 or 0.2 msec.

The operating range of the algorithm is configured to 10-40Hz, i.e. it aims at detecting a flutter frequency in this range.

For more details on the pitch detection algorithm refer to Appendix A.

RESULTS

In all 73 sessions with 1-3 exhalation cycles each was obtained from the 19 people in the study group. An example of such a session is shown in Figure 3.

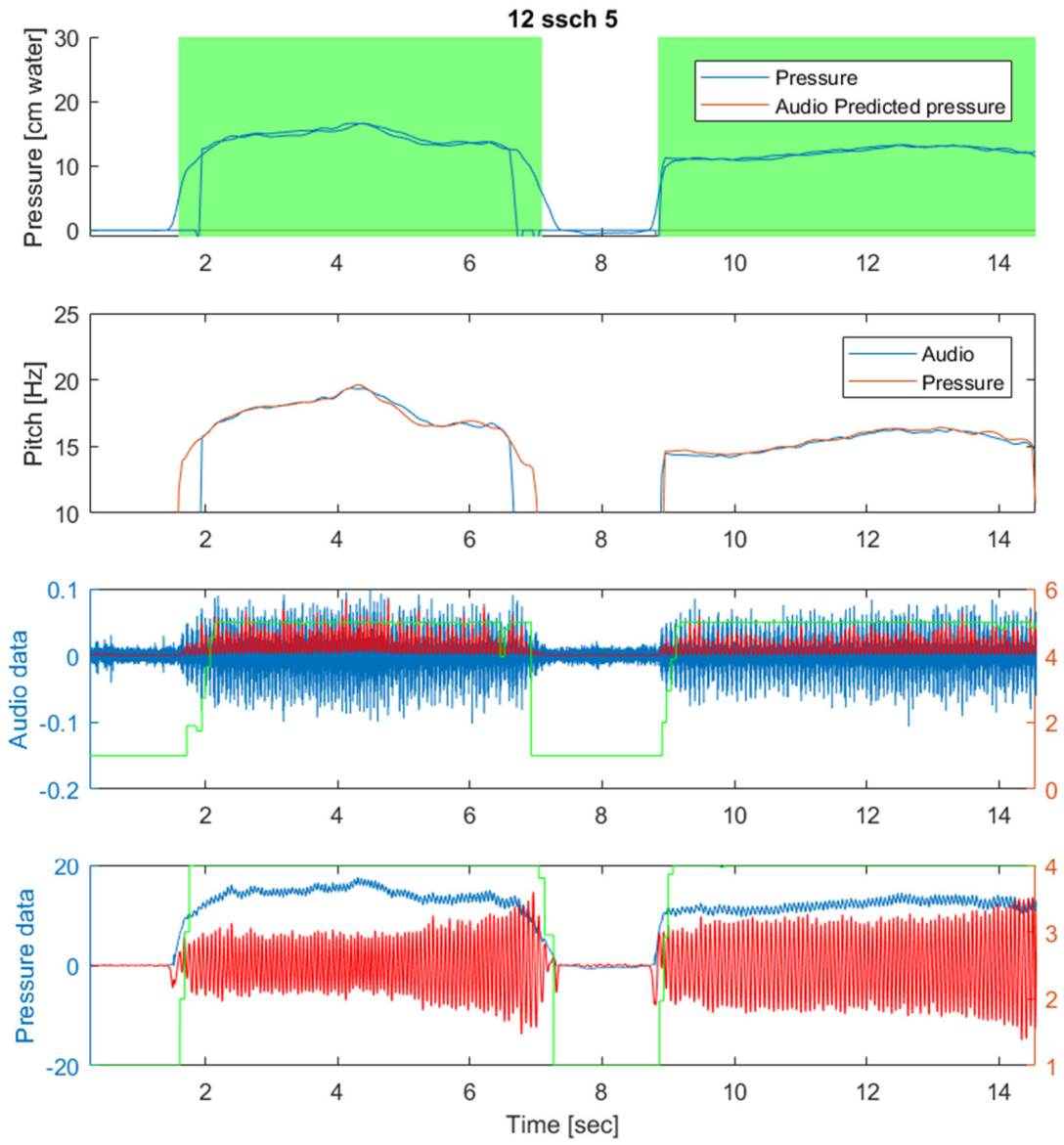

Figure 3. Example of a session: Top: green zones illustrates the exhalation periods. The graphs show the actual pressure along with the pressure predicted from the pitch detected in the audio signal. The second from top: Detected pitch from audio signal and from pressure signal (yes, you can run the same pitch detection algorithm on the 1kHz data too). Two bottom graphs shows raw and filtered versions of audio and pressure data respectively.

To establish the model, the data was filtered in the following way: Only data points where the pressure was in the range 6-30 cm water, and the detected pitch at 10Hz or higher. Note that the actual operating range is wider than this, but when establishing the model it makes sense to not go too far out of the correct usage operating range, which is 10-20 cm water.

The main results along with the established linear model is shown in Figure 4. Despite the diversity of the test group, the different Acapella resistance settings (1-5), and the different exhalation instructions, the data follows an extremely tight correlation between the pitch determined from the audio data and

the measured pressure. Using a linear model produces an r^2 of 0.886 over the 9,993 data points, which is extremely good.

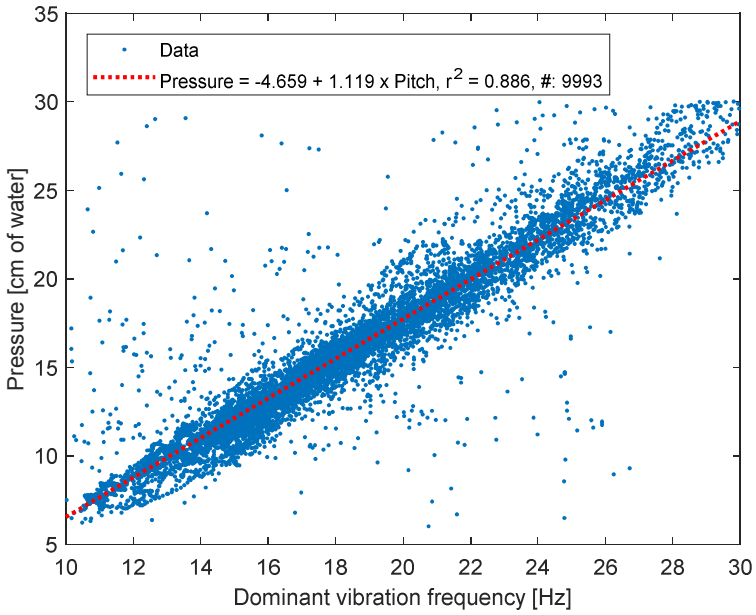

Figure 4. Pitch/Pressure data points and the model. The x-axis is the pitch detected from the audio data and the y-axis is the measured pressure. With an r^2 of almost 90% across the entire diverse sample and different test sequences this is extremely good.

Figure 5 shows the same data as in Figure 4 but with color coding for resistance setting and test person. Just from looking at the graphs, it is apparent that one model will be sufficient for all resistance settings and all test users. This greatly simplifies the use cases for the smart phone app.

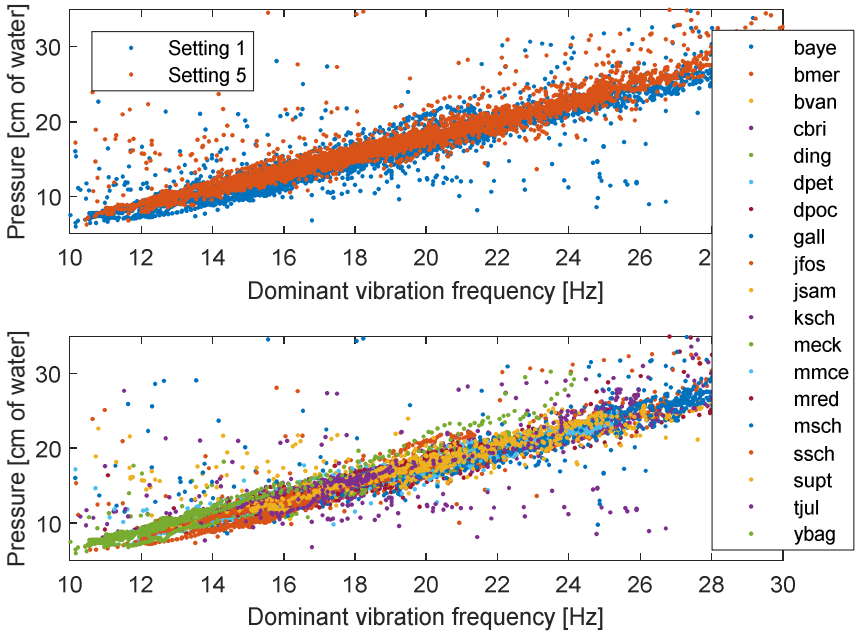

Figure 5. Data points using different coloring for (above) resistance setting, and, below, test person.

APPENDIX A: THE PITCH DETECTION ALGORITHM

The detailed steps of the pitch algorithm are listed below:

- A target range of the pitch is defined as: [f_{lower} f_{upper}] = [10Hz 40Hz]
- The audio signal, which comes at F_S = 48kHz or 44.1kHz, is downsampled by a factor 45 (to 980Hz). The downsampling rate is arrived at by setting the desired pitch accuracy at 2.5% in the middle of the frequency range
    - From the raw audio signal we subtract a smoothed version
        - The smoothed version is computed by averaging over a neighborhood equal to the length of the smallest formant frequency (this is the typical natural frequency generated wihin the Acapella, starting at 250Hz and up) *176* audio sample points
    - After the subtraction, compute the square
    - Downsample the signal by averaging contiguous sets of 45 samples (no overlap)
    - We refer to this downsampled signal as the audio energy
- A smoothed version of the audio energy is computed by convolving with a filter, which is a Gaussian with sigma = 0.2*45*f_{upper}/F_s. This means the FWHM of the Gaussian is 0.4 times the period of the highest pitch frequency we're looking for. If we go much larger, we will start to blur out the pitch signature in the upper range (40Hz).
- As chunks of audio becomes available, they are processed in a pipeline that keeps track of previous moving averages
- In the smoothed signal the autocorrelation is computed in an economical fashion:
    - Once a pitch has been detected, the next autocorrelation is computed at a location one wavelength (of the last pitch detected) ahead of the previous point
    - A target range for the pitch is maintained so that autocorrelation computation is focused in the most likely locations
    - The time resolution of the autocorrelation is rough in the first calculation, then finer resolution is done around the maxima
- Pick the argmax of the autocorrelation function, provided that the correlation is greater than 0.6
    - Update the moving average value for the pitch with the pitch corresponding to the argmax value
    - Update the derivative of the moving average, in order to refine the search window at the next iteration
- Update a run length for pitch detection:
    - The run length is initialized to 0
    - Every time a pitch has been positively identified, the run length is incremented by 1, <u>unless:</u>
        - The run length has reached a maximum, which is given by a decay rate as round(1 – 1/decayRate). In our case, decay rate is set to 0.8, which gives a max run rate of 5
    - If from one detected autocorrelation (>0.6) value to the next, the leap exceeds the expected amount, as given by the running average value for pitch, the run rate is reduced by the equivalent number

- Compute the mix value a: mix = min([decayRate 1 - 1/runlength])
- Update the moving average values for, period (1/pitch), amplitude, derivative
    - MovingAve = movingAve*mix + newValue*(1-mix)
- Compute pitch as: Pitch = 1/(movingAvePeriod)

APPENDIX B: IMPLEMENTATIONS OF ALGORITHM

The pitch detection algorithm and the pitch/pressure model was prototyped in Matlab, which serves as a reference implementation.

In addition, the pitch detection algorithm was implemented in C++, using only core C++ libraries such as vector and string. This implementation was validated against the reference implementation to produce the same results. It was also checked for memory leaks, and executed on all 73 sound samples to safeguard against any unexpected bugs. This algorithm, when processing the audio in incremental chunks of 0.2sec, executes a 20 second audio sample in about 50milliseconds. This provides a performance headroom factor of 400, which should provide ample headroom when migrating the code to a mobile platform rather than a PC.

APPENDIX B: INTERFACE OF ALGORITHM

The algorithm, implemented by a class, pitchTracker, operates incrementally on chunks of data. It has two main interfaces:

CONSTRUCTOR:

```
pitch_tracker(float Fs, int minFreq, int maxFreq, float freqAccuracy, int
lowerFormantFreq, float decayRate, float minAmp, bool saveResults);
```

Where the suggested parameter values for the acapella application are:

`Fs` = 44,100 (the audio sampling rate). Could also be 48,000 depending on the native app/os environment

`minFreq` = 10, `maxFreq` = 40, `freqAccuracy` = 0.025f , `lowerFormantFreq` = 250, `decayRate` = 0.8f, `minAmp` = 2.0E-4f

Note that the constructor should be called whenever a new session is created, i.e. when the user enters the view where sound is being monitored.

SUBMITTING NEW AUDIO DATA FOR PROCESSING

```
float processChunk(float* audioSignal, int thisFirstNativeInd, int numElements);
```

Returns the computed value for Pitch as a float.

Takes a pointer to an audio buffer in the `audioSignal` input parameter. In addition, it takes a number `thisFirstNativeInd` that indexes the first value in the audio Buffer, counting from when the first time

processChunk was called in the current session. Lastly it needs `numElements,` the number of elements currently in the audio buffer.

SUGGESTED BUFFER SIZES

There is a slight performance drop (very small) when choosing smaller buffer sizes, but the advantage is shortened latency. Suggested buffer size is 0.1sec, which will give a latency of about the same amount.