A Commercial Nocturnal Asthma Monitor

Final Report

Group Number: Group 26

Group Members:

William Padovano

David Kim

Chris Beyer

Course: BME 401

Date: 12/1/2014

Need:

Nocturnal Asthma (NA), or a nighttime exacerbation of asthma symptoms, affects an estimated 47-75% of the several hundred million asthmatics worldwide[1,2]. Like sleep apnea, the disorder causes frequent nighttime arousals and lower-quality sleep, and it has been linked to depression, anxiety, and "developmental, emotional, and behavioral problems" in children[3,4,5]. NA is indicative of improper management of asthma symptoms, and its prevalence is largely due to a lack of awareness that the sufferer is even affected. Indeed, there is currently no objective, home-based monitoring system for nocturnal asthma.

Scope:

This report describes a prototype for a commercial, home-based device capable of continuously monitoring symptoms and alerting parents or caregivers if intervention may be required (i.e. during an asthma exacerbation). The device monitors nocturnal cough frequency and is not diagnostic for asthma. Instead, it is intended for children who have already been diagnosed with asthma and who may suffer from NA. The device can be trained to the child's cough and will silently run in the background every night.

Design Specifications:

Hardware specifications	**Metrics**
Sampling rate	44,100 samples/second
Recorded audio frequency range	Up to 20 kHz
Minimum single core CPU speed*	400 MHz
Minimum SDRAM size*	512 MB
Minimum SDRAM read/write speed*	400 MHz
Power supply requirements	3W - 10 W at 5 V
Transmitter open field range	300 m
Operating noise	Below 30 dB (inaudible)
Software specifications	**Metrics**
Read rate	44,100 samples/sec
Allowable processing delay	Less than 50 ms
Computations	Must perform Fast Fourier Transforms
Enclosure specifications	**Metrics**
Length x width x depth	113 mm x 97 mm x 58 mm
Weight (prototype)	240 g

***** Abbreviations explained in the Details of Key Requirements section

Details of Key Requirements:

The minimum audio sampling rate of 44,100 kHz was chosen because it enables the software to accurately monitor frequencies of up to 22,050 kHz (44,100 / 2). This relatively large upper frequency bound is necessary because coughs can have significant power in frequencies as high as 18 kHz. As is described in greater detail in the Software Description section, the monitoring software processes sounds with an 11.6 ms time delay. Considering that coughs typically last more than ten times longer than this, the monitor can detect coughs in real-time.

The Raspberry Pi B+ (RPi) used in the prototype has an ARM V7 32-bit CPU (central processing unit), a VideoCore IV GPU (graphical processing unit), and 512 MB of SDRAM (standard dynamic random access memory) that is shared between the CPU and GPU.[6,7] The standard operating frequencies are 700 MHz for the CPU, 250 MHz for the GPU, and 400 MHz for the SDRAM. However, the performance of the Raspberry Pi can be raised, or "overclocked" to 4 different levels: "Modest," "Medium," "High," and "Turbo." The CPU, GPU, and SDRAM speeds at these levels are shown in **Table 1**. The Raspberry Pi can also be "underclocked," or its processing speed can be lowered. To find the minimum processing speed for this cough monitoring software, the CPU, GPU, and SDRAM speeds were altered to 5 different levels as is shown in **Table 1**. The minimum possible CPU, GPU, and SDRAM speeds appear to be around 400 MHz CPU, 200 MHz GPU, and 400 MHz SDRAM. The differences in run loop iteration times (minimum, maximum, and mean times) for the cough monitoring software at this minimum speed as well as the 5 other levels that could successfully run the software are shown in **Supplementary Figure 1**. From this plot, there is a trend towards shorter iteration times for increased clockspeeds. While increased performance comes at the cost of longevity, power consumption, and RPi system stability, the significantly shorter iteration times lead to greater software reliability as is discussed in the Performance and Limitations section. Consequently, the RPi was set to the "Turbo" level in this report.

Table 1: CPU, GPU, and SDRAM speeds at different settings and their associated successes. In Underclock 4, the program ran for a few iterations but always errored after a few seconds.

Name	CPU (MHz)	GPU (MHz)	SDRAM (MHz)	Run success
Underclock 1	200	200	200	No
Underclock 2	200	200	400	No
Underclock 3	400	200	200	No
Underclock 4	300	200	300	Error during run
Underclock 5	400	200	400	Yes
None	700	250	400	Yes
Modest	800	250	400	Yes
Medium	900	250	450	Yes
High	950	250	450	Yes
Turbo	10,000	500	600	Yes

While the GPU speed was included in the discussion above because the software was run from the RPi desktop, the it was not listed in the requirements because the final product will not have an associated LCD screen or complex graphical user interface. Instead, as in the prototype, it will have a small LED display. This choice was made both to lower cost and to increase the amount of memory available to the CPU. Indeed, the GPU in the raspberry pi uses a full 64 mb of the available 512 mb. At least this amount of memory is required for the prototype because, as is discussed later in the report, the software appears to be using most of the current memory to run. Indeed, the memory size currently precludes several useful features in the software, including continuous updating of the cough template. However, the memory requirements could probably be lowered if the audio signal were initially passed through analog filters as is mentioned in the Future Directions section.

The RPi is powered by a 5V power supply and the current it draws is dependent on the number of USB devices connected and the overclocking level. However, the possible current range is 600 mA to 2000 mA, and so the prototype uses between 3 and 10 Watts. The 10 W upper range is likely greater than the energy consumption of the finished cough monitor design because it would not need to power the up to 4 USB devices and displays that the RPi can.

The microphone and soundcard used in this prototype is from a Microsoft LifeCam Cinema webcam. According to the datasheet for this product, it only has an audio frequency

response of 200 Hz to 8,000 Hz +- 4 dB[8]. Because this cutoff is lower than the upper frequency range for coughs, these higher frequencies are attenuated in our recorded signal. However, it is worth noting that the high sensitivity and high sound quality of this microphone appear to help remedy this issue. This point can be seen in **Figure 1**, where audio spectrograms are produced from recordings performed with a less expensive microphone and soundcard (Audiotechnica ATR4650 microphone Sabrent 3D Audio sound card). The Audiotechnica microphone actually purports to have a frequency response of 50 Hz to 13,000 Hz, though the spectrograms appear noisier due to decreased sensitivity and recording quality. Ideally, the finished product will contain a microphone with a broad frequency response and high sensitivity, though this also comes with greater cost.

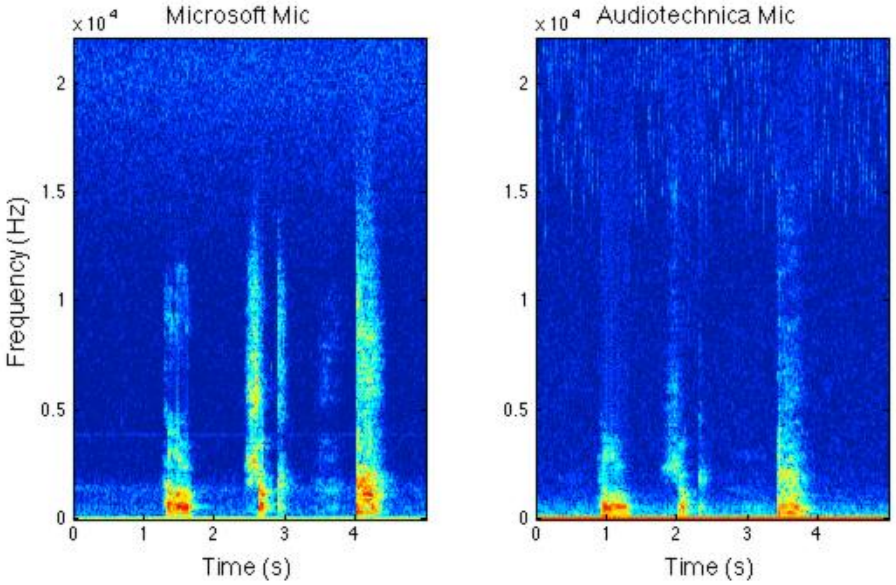


Figure 1: Comparison of spectrograms from two different microphones using the words “Hello” and “shook,” and a cough. Color bar ranges from -3 to 5 arbitrary units.

Details of chosen design:

Physical embodiment:

A 512 MB Raspberry Pi B+ single-board computer was used to run the Python software described in subsequent sections. The RPi is connected to a 5 V power supply and receives audio input from a Microsoft Lifecam Cinema USB webcam. The GPIO (general purpose

input/ouput) pins of the RPi are connected to two LEDs and to a 4 digit display. One LED flashes every time a cough is detected and the other turns on when a high frequency of coughs is detected and serves as an alarm. The LED screen displays the number of coughs. The design of a 3D printed enclosure for these components is shown in **Figure 2**.

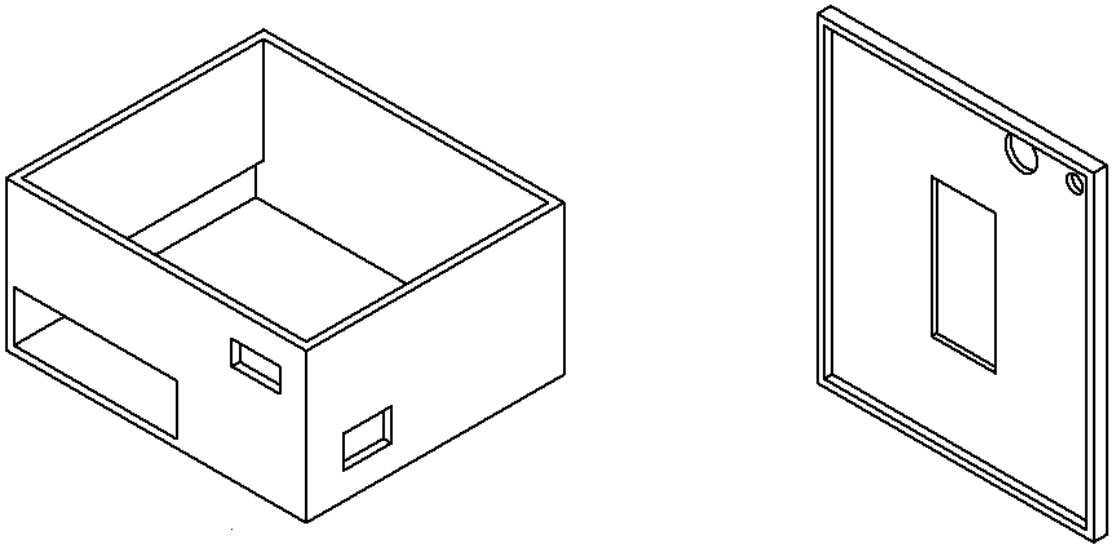


Figure 2: On left, the design for an open box houses the Raspberry Pi and camera. The windows cut into the sides of the box are for input cables to the RPi or help expose the camera's microphone. On right, the lid for this box. The rectangular window is for an LED display and the two circular holes are for the LEDs.

Software overview:

A stream of audio data is continuously analyzed in 11.6 ms time segments and a Fast Fourier Transform (FFT) is performed on each segment. If there is significant power in three selected frequency bands, then that segment may be part of a cough and the FFT is appended as a column to a growing spectrogram (rows are frequencies and columns are time bins). Once the power in the frequency bands drops down below specific thresholds, the audio event is over and the collected spectrogram is analyzed. It is compared to a template cough from the user, and if the two spectrograms are similar, then the sound event is registered as a cough. An alarm is triggered if several coughs are detected per minute. Detailed flowcharts of the cough detection and template creation programs are shown in **Figures 3 and 4** below.

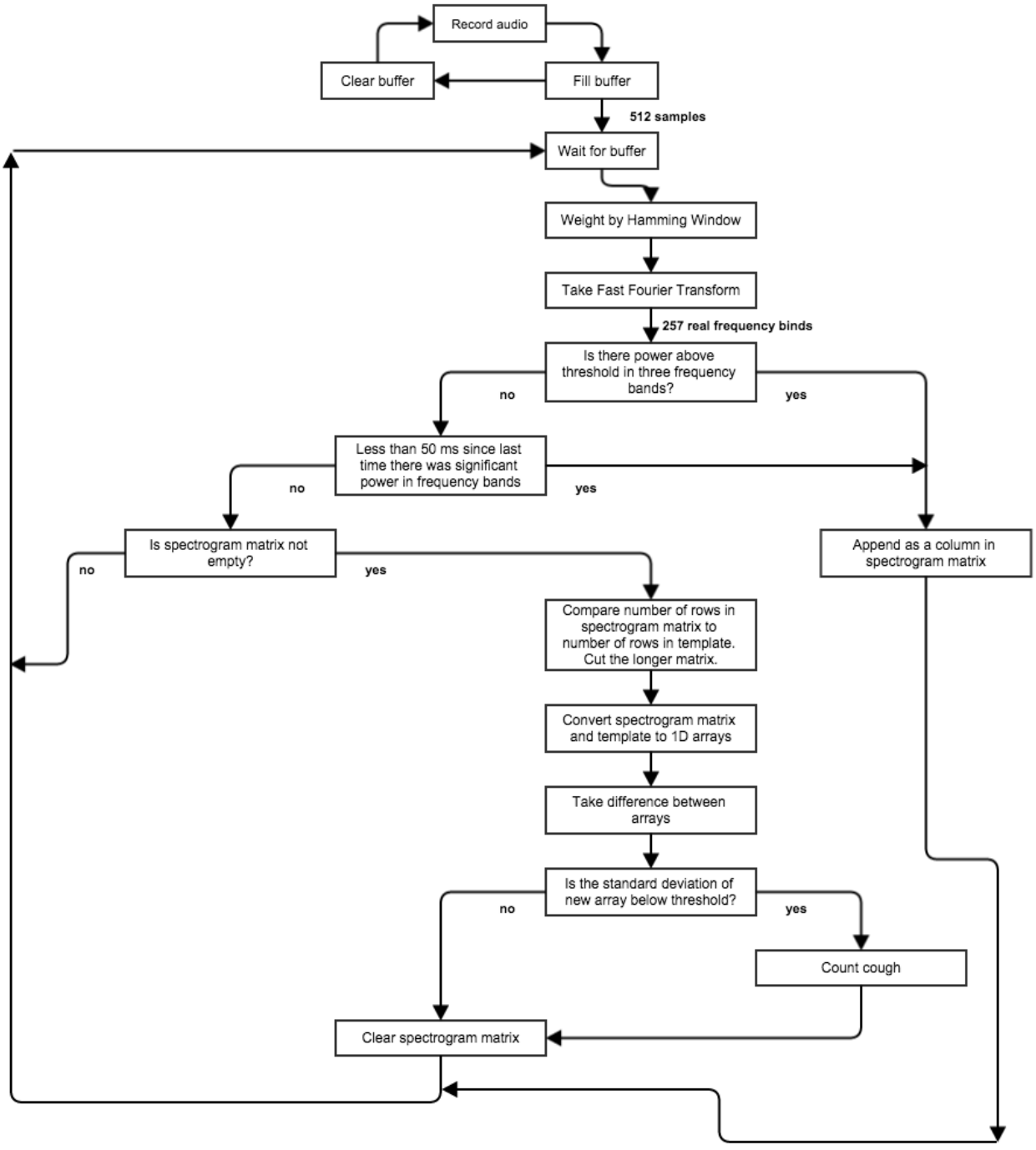


Figure 3: The flowchart for the cough detection program. Note that the software is composed of two loops—for recording and processing—that run simultaneously.

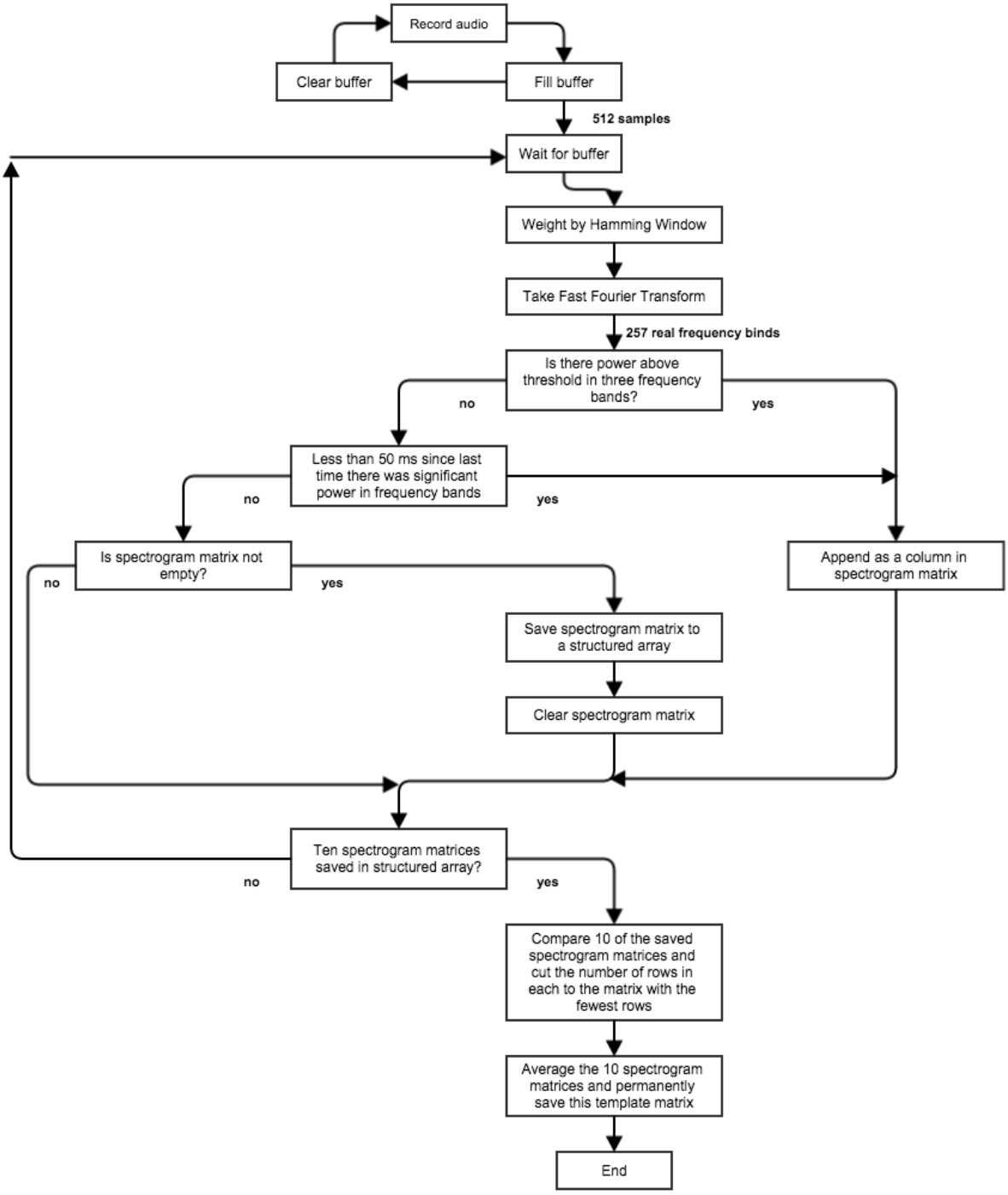


Figure 4: The flowchart to produce the template that is used in the cough detection program. While the cough detection program is an infinite loop, the template creator ends once 10 coughs have been collected.

Detailed Software Description:

The software was written in Python because it is a convenient language for use in the Raspberry Pi and because it offers a large number of useful and free packages for audio signal collection and manipulation. This software uses the PyAudio, NumPy, RPi.GPIO, and Time packages.

As was discussed in the design specifications section, the audio sampling rate was chosen to be 44,100 samples/second. However, the software's processing loop does not iterate 44,100 times per second. This makes practical sense because each iteration simply cannot be run in the under .000022 seconds required. Further, several data points are needed to compute FFTs. Instead of reading in samples immediately, chunks of several samples are read at a time. The size of the audio chunks depends on the method used to create sound spectrograms. In Python, spectrograms can be formed using the "specgram" function or iteratively by piecing together several Fast Fourier Transforms produced with the "fft" function. The "specgram" command analyzes a large chunk of audio data and then splits it up into a number of smaller windows. The windows can be overlapped, which reduces noise and improves accuracy but also significantly increases computation time. However, as is shown in **Supplementary Figure 2** there is little difference between cough spectrograms produced with 50% window overlap and zero overlap. The iterative FFT method produces a matrix that is equivalent to this zero overlap case, where the window size is the length of one audio chunk, as is confirmed in **Supplementary Figure 3**. However, the "specgram" function does not appear to be feasible for real-time recording with relatively low computational power devices like the Raspberry Pi. Indeed, even with zero window overlap, it takes an average of 11 ms for the RPi to split 1024 samples into two nonoverlapping 512 sample windows and compute the spectrogram, while it takes under 3 ms for two 512 sample iterations of the FFT method to do the same. This may be because less data must be produced, manipulated, and stored at once in the FFT method than the "specgram" method. The spectrogram matrix can also be initialized before the start of the

loop in the FFT method, which further decreases the computational burden. This is crucial because the device's memory seems to be a major constraint in our design.

Consequently, the FFT method was chosen. However, due to the algorithm used, the FFT of a data set with length n has n frequency bins (where n/2+1 are real frequencies). When sampling continuously, there is therefore a tradeoff between the number of frequency bins and the number of time bins in the spectrogram of a detected sound event. For example, with a chunk size of 1024 samples, the spectrogram has an excellent frequency resolution of 513 real frequency bins. However, each column of the spectrogram contains 23 ms of data (1024 samples / 44,100 Hz). A smaller chunk size of 256 samples offers 4 times more time bins at the expense of 4 times fewer frequency bins. Cough spectrograms produced with 256 sample, 512 sample, and 1024 sample chunk sizes are shown in **Figure 5**.

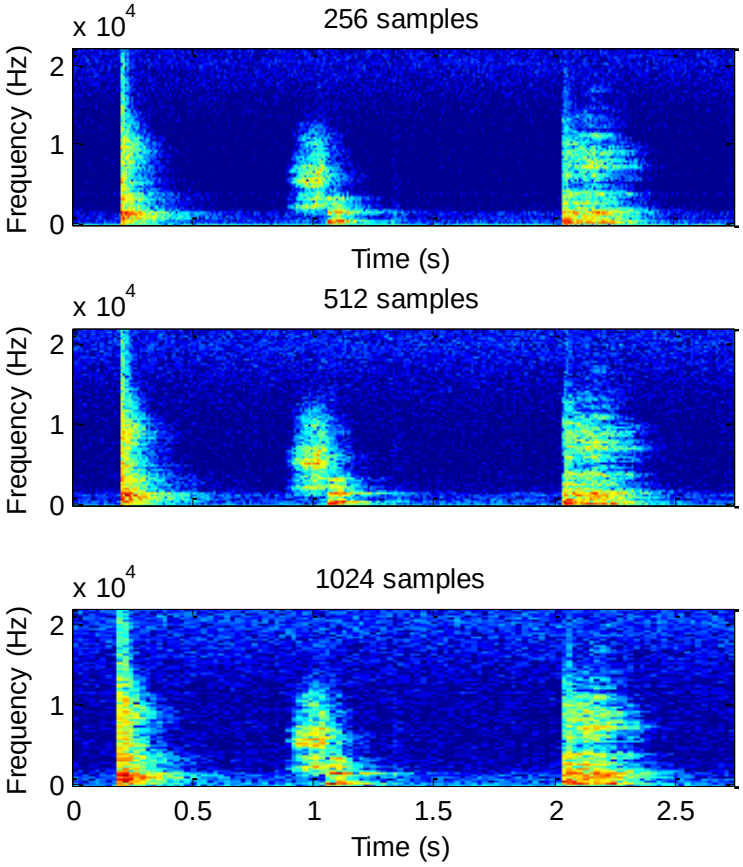


Figure 5: Spectrograms of a clap, the world "ship," and a cough formed with different sample window sizes. Color bar ranges from -3 to 5 arbitrary units.

A chunk size of 512 samples was chosen because, visually, it appeared to provide the greatest consistency between coughs and the greatest difference between coughs and other sounds.

And so, the audio samples fill a buffer at 44,100 samples per second and, once the 512 sample chunk size is reached, the buffer is cleared and the samples are analyzed in a loop iteration. During this analysis time, the buffer fills again with the samples for the next loop iteration. Before the FFT is taken, the 512 samples are weighted by a Hamming window function that is shown in **Supplementary Figure 4**. This is necessary because the FFT algorithm assumes that the signal is periodic in the chunk. A non-periodic signal, such as a cough, results in leakage, or an inaccurately wide distribution of power in the FFT spectrum. A Hamming window was chosen to diminish the effect of leakage because it works well with random signals, is frequently used in sound analysis, and is the default window for Matlab spectrograms.[9] Spectrograms produced with and without the use of a Hamming window are shown in **Figure 6**.

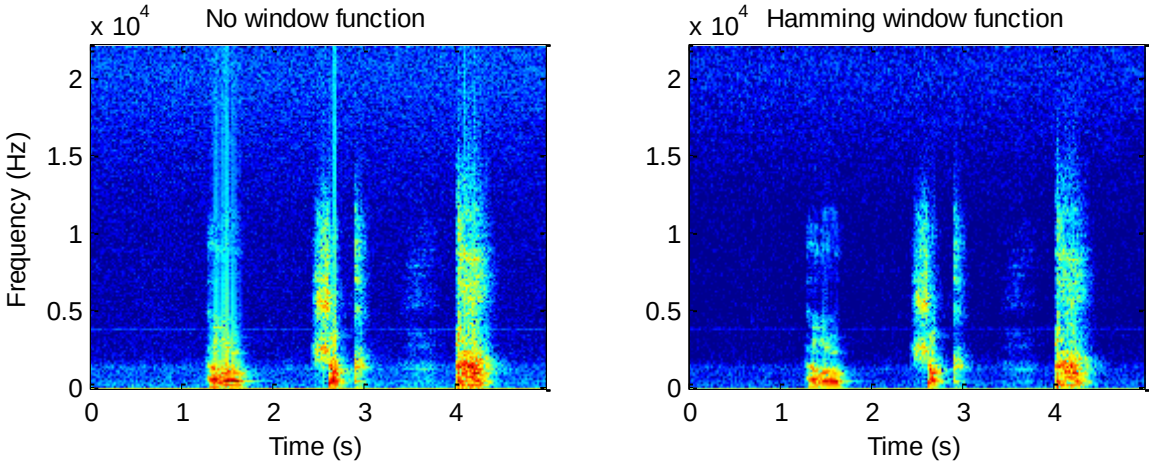


Figure 6: Spectrograms of the words “hello” and “shook” and a cough that were formed without (on left) and with (on right) weighting by the Hamming window function. Color bar ranges from -3 to 5 arbitrary units.

Following the FFT, the magnitudes of the complex values in the real frequency bins (257 bins) are taken and they are scaled logarithmically. The signal power in three frequency ranges of the resulting 257 element array (1.1 kHz - 1.3 kHz, 6.7 kHz - 6.9 kHz, and 8.4 kHz - 8.6 kHz)

are then examined to see if they are above experimentally determined thresholds. Coughs spread a large frequency range of 200 Hz - 18 kHz that help to distinguish them from normal speech. From spectrograms recorded for this report, voiced sounds tend to have power between 200 Hz - 3 kHz and fricatives and non-voiced stops are usually in the range of 1.5 kHz - 14 kHz. However, most of the power for speech sounds tends to lie near the lower frequencies of these ranges. Many other sounds, such as claps and snores, span similar frequencies as coughs and so power in these three frequency bands alone cannot be used to robustly detect coughs. **Figure 7** shows the frequency content of coughs, speech sounds, and a clap in the three frequency bands. Note from this figure that the soft cough has much less power in frequencies above 10 kHz than the loud cough, which is why the highest monitored frequency band (8.4 kHz - 8.6 kHz) is not at higher frequency.

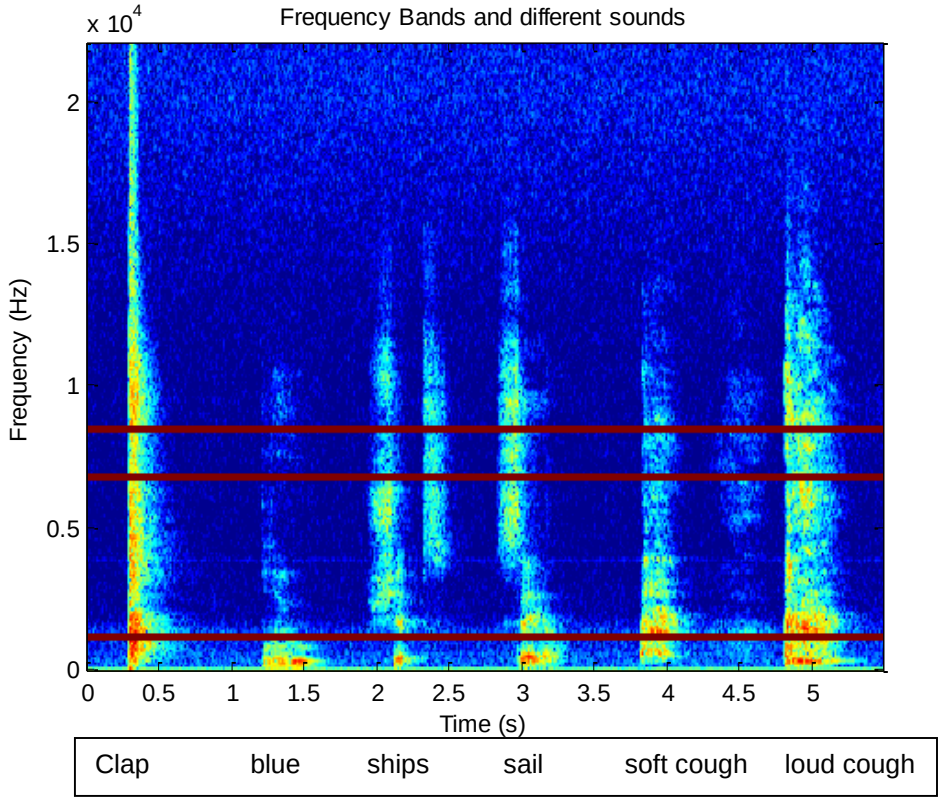


Figure 7: The words “blue,” “ships,” and “sail; a clap; and two coughs (soft and loud) are shown in relationship to the three frequency bands monitored by the software (shown in red). Note that the clap has significant power in the same bands as the coughs. Color bar ranges from -3 to 5 arbitrary units.

If the FFT array has significant power in the three frequency bands, then it is appended as a column in growing matrix, which is initially empty. Because a few chunks within a cough might fail the frequency screening, the 3 chunks of audio data (35 ms) following a chunk that passes this test are appended to the matrix regardless of their frequency content. If 46 ms of data (4 chunks) is recorded without a chunk that passes the frequency screening, then the audio event is over. This number of chunks was chosen because smaller numbers sometimes resulted in truncated coughs. Before the collected matrix of FFTs, or the spectrogram, is analyzed, its width is compared to a minimum threshold of 12 chunks. This represents 8 chunks of actual signal, or about 100 ms, because signals shorter than this are very likely not coughs. The spectrogram is now compared to a template cough as a final, rigorous test.

The cough template is created during a training period, in which the spectrograms of 10 coughs from a specific user are averaged into a single matrix. This number of samples, rather than a larger value, was chosen due to memory constraints in the Raspberry Pi. When creating the template, there is the issue of matching the coughs for averaging, since coughs seldom have the same exact duration. One solution to this problem is clipping the columns of all cough spectrograms to the number of columns of the shortest spectrogram, such that the resulting template has the same length as the shortest recorded cough. Another method is to rescale all the longer coughs to the shortest cough duration by interpolating values with a bicubic interpolation method. **Figure 8a** provides side-by-side spectrograms of the clipped portion of a long cough, which had an original duration of around 0.80 seconds, and shorter cough that lasted 0.35 seconds**. Figure 8b** scales the longer cough to the duration of the shorter cough.

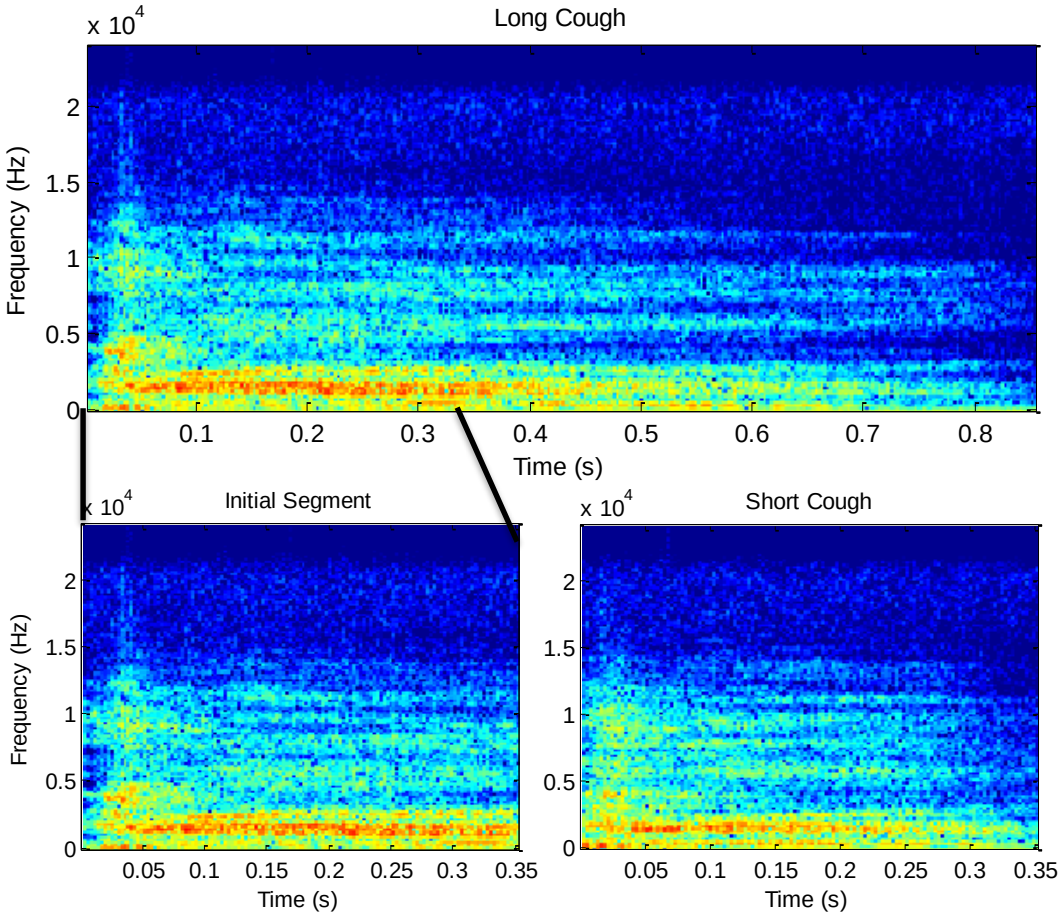


Figure 8a: A longer cough is cut to the width of a short cough, and the initial segment of the long cough is displayed next to the short cough. Note the similarity between the resulting spectrograms. Color bar ranges from -3 to 5 arbitrary units

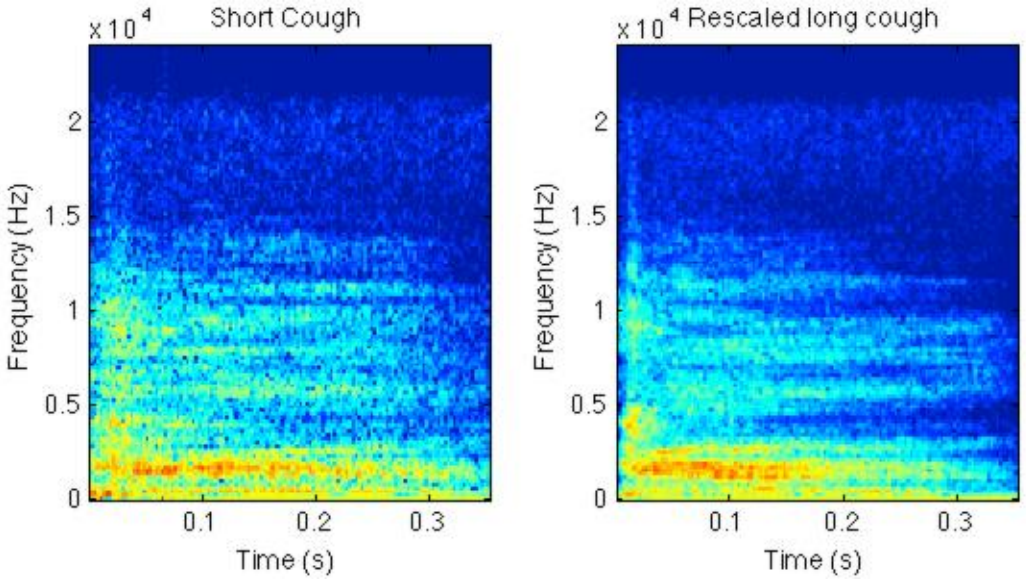


Figure 8b Side-by-side comparison of a short cough and a rescaled long cough. Color bar ranges from -3 to 5 arbitrary units.

Visually, the two spectrograms in both figures appear very similar, and so the methods were compared by a more quantitative approach. In each case, the difference was taken between the size-matched spectrograms, and the standard deviation of the resulting values was computed. The clipped and short coughs had an average difference of 0.21 a.u. and a standard deviation of 1.00 a.u., and the rescaled and short coughs had an average difference of 0.25 a.u. ad a standard deviation of 0.84 a.u. Although the clipping method produced a smaller average difference than did the rescaling method, the rescaling method had a much smaller standard deviation. This is because the clipped spectrograms diverged greatly in the last few columns.

While the scaling method seemed to contribute to a better cough template in Matlab, it has limited efficacy in the Raspberry Pi because it is much more computationally expensive. Other interpolation methods could have been applied as well, but still pose the same issue. Fortunately, however, the most unique portion of a cough appears to be the initial segment, and so the simple clipping method still produces a useful cough template.

Once the cough template is loaded by the detection program, it is compared to each sound event that passes the preliminary frequency screening. Several comparison methods were examined to find the scheme that could most easily distinguish between coughs and other sounds with little processing power. Among them were: principal component analysis (PCA), cross-correlation, dot product, and standard deviation.

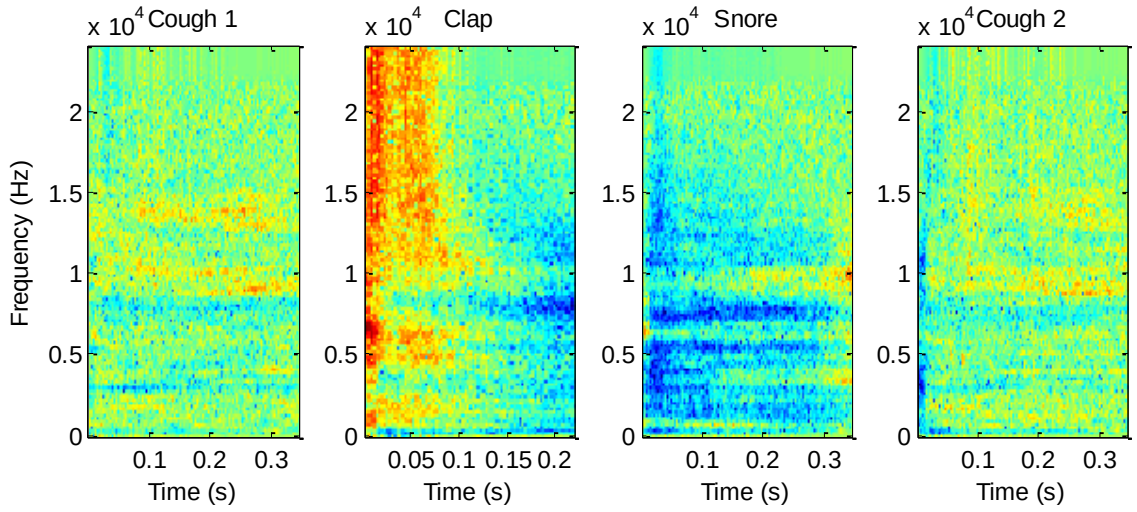
For the PCA method, each frequency bin of the time matrix was considered to be a dimensions and each time chunk had a measurement in each dimension. The first three principal components (PCs) contained 75.9%, 8.9%, and 3.8% of the variance, though only the first two principal components were considered for the analysis to prevent excessive computational complexity. As is shown in **Supplementary Figure 5**, the spectrograms of two new coughs, a clap, and a snore were projected onto the eigenspace generated by the first and second principal components of the template cough. The spectrograms of the original sounds and template are found in **Supplementary Figure 6**. While there does appear to be some

separation between the coughs and other sounds, the events do not seem separate enough to robustly distinguish coughs. Indeed, there is very significant overlap for measurements around the coordinates (PC 1, PC 2) = (-7, 10).

Two dimensional cross-correlation was also examined to measure similarity between the template and other sounds. For these matrices, the highest possible cross-correlation would be obtained when the template and sound event were lined up in frequency and time. However, this is equivalent to a simple dot product between the two matrices. **Supplementary Figure 7** illustrates this point, where the intensity is highest at the center of the plot, or at the point where the spectrograms are lined up in frequency and time. To simplify calculations, this maximum value could be used to determine an appropriate threshold for cough classification. This value can be determined more easily by simply flattening the audio event and the template spectrograms into 1D arrays and finding the scalar projection of event on the template. **Supplementary Table 1** shows the normalized scalar projections of different audio events on the cough template. The relative error between two coughs is 0.58%, between the coughs and a clap around 11%, and between the coughs and a snore is about 19%. Alternatively, some method could be developed to analyze the intensity distribution of the cross-correlation as a function of the x- and y-shift. However, this would be a very difficult process for the Raspberry Pi or any small computer to perform.

Finally, the simple difference was taken between a sound event and the template. This matrix was then flattened into a 1D array and the standard deviation was calculated. **Figure 9** illustrates this process as the two coughs, clap, and snore spectrograms are compared to the cough template. The total sums of the subtracted spectrograms seem fairly similar for each sound event. However, there was only a 3.9% difference between the standard deviations of the two coughs. When compared to the 35% relative error between the coughs and the snore and the 75% relative error between the coughs and the clap, this detection scheme appears to be very robust. The process can easily be performed by the Raspberry Pi prototype, and it has the

additional advantage that the information in a two-dimensional array containing around 10,000 data points can be reduced to a single scalar value. The scalar projection method has similar advantages, and, while it had very low relative error between coughs, it tended to have lower relative error values between coughs and other sounds than the standard deviation method. Consequently, the standard deviation method was chosen for the cough detection program.



Sound event	Total sum (a.u.)	Standard Deviation (a.u.)
Cough 1	9.6196e3	0.7687
Clap	1.1354e4	1.3723
Snore	1.6972e4	1.0629
Cough 2	9.9372e3	0.7987

Figure 9: The spectrogram for each sound event is subtracted from the template matrix. These difference matrices are shown for two coughs, a clap, and a snore. The total sum and standard deviation for each difference matrix is tabulated. Color bar ranges from -3 to 5 arbitrary units.

And so, once an audio event with sufficient power in all three chosen frequency bands is detected, the difference is taken between its spectrogram and a template cough. If the standard deviation of the resulting of resulting array is below a given preset threshold, then the sound event is counted as cough. If several coughs are counted during a particular time interval, then an alarm signal is triggered. Though there is no standard cough frequency that signals an asthma exacerbation, several physicians stated that around 20 coughs/minute was a good threshold.

Performance and Limitations:

Performance:

To gauge the selectivity and specificity of the cough monitor, the software was run in four, thirty-minute tests. The first trial was performed with a silent background and the participant coughed periodically. Another team member manually counted the coughs and observed if and when a cough was registered by the Raspberry Pi. Under these conditions, the software correctly detected all 38 coughs with zero false positives. In the second trial, two team members had a conversation as one member coughed sporadically and, encouragingly, there was 100% cough detection and 1 false positive cough. The false positive occurred when the observing member said the word “sure” with a prolonged “sh” sound. The two-team members then watched a television show and talked freely in the third trial as one of the members coughed. While the monitor correctly detected all coughs, there were 4 false positives detected. These were triggered by some sound effects in the loud TV show and during a laugh. While the rate of 8 false coughs per hour is relatively high, it is important to note that this trial does not reflect the device’s intended operating conditions. Lastly, the participant coughed periodically with his face in a pillow, which muffled and reduced the sound intensity. While the monitor had no false positives, it missed 6 of the 24 recorded coughs. These misses seemed to be very closely related to sound intensity, and very soft, short coughs were not detected.

As a result of this trial, the minimum sound intensity was determined. A participant coughed at different intensities, which were monitored in decibels by a smartphone sound meter application. The resulting cough spectrograms are shown in **Figure 10** and the monitor was able to detect all coughs at and above around 50 dB, which is around the sound intensity of a quiet conversation monitored by that same application. This minimum intensity is reasonable because most coughs under normal conditions had an intensity of around 65 dB. It is important to note that the spectrogram for the 47 dB cough looks very different from the louder coughs. The cough template was trained with louder coughs and this likely contributed significantly to

the minimum cough intensity for detection. Beyond sound intensity, the coughs used to train a template naturally have a very large impact on cough detection. On average, coughs from the cougher that trained the template have significantly lower standard deviation values than other coughs as is shown in **Supplementary Figure 8**, though both coughs in this figure have lower standard deviations than the clap. This specificity could potentially allow the monitor to focus on a single child's cough even if the child shares a room with other siblings.

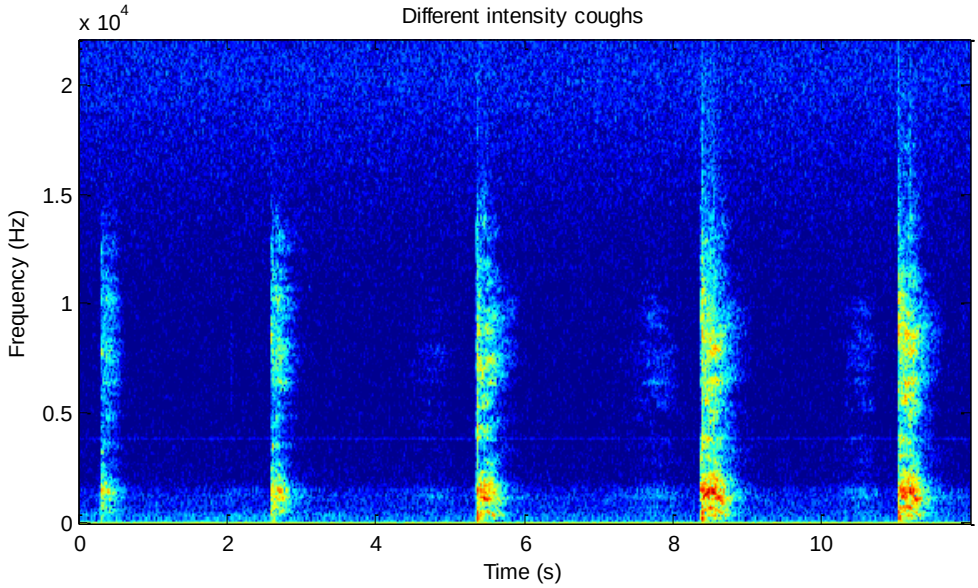


Figure 10: Spectrograms of coughs with intensities of 47, 51, 59, 62, 66 dB, respectively. Color bar ranges from -3 to 5 arbitrary units.

Limitations:

The software does not work properly if there is loud drum- or electronic bass-intensive music playing in the background. As can be seen in **Supplementary Figure 9** the spectrograms for a cough and a beat that registered as a cough are fairly similar. The software also occasionally incorrectly detects a cough during laughter, especially during raspy laughter. This is because the prolonged "hhhh" that sometimes precedes voiced sections in a laugh are very similar to the "hhhh" sound that that follows the start of a cough. However, it is important to note that this monitoring device is intended for use in a reasonably quiet environment and so these mistakes are likely not a significant issue.

Additionally, the software requires there to be low power in the three selected frequency bands for 50 ms following the end of a cough before it can register another cough. If a second cough starts within this time window, then the software does not detect it. This means that streams of several coughs might be underrepresented by the software, and some methods have been attempted to fix this issue. The most promising of these is discussed in the Future Directions section.

The greatest limitation of the current design, however, appears to be the hardware. The Raspberry Pi's memory appears to fill relatively quickly and this prevents the software from saving cough spectrograms over time to periodically update the cough template. This learning would likely allow the software to improve its specificity. The limited memory also prohibits the saving of nighttime recordings, which would allow parents to better assess the accuracy of the cough monitor or to get a better sense of their child's asthma symptoms. Another issue is that each loop iteration must be completed in less than 11.6 ms to avoid an overflow error, where the audio buffer fills to capacity before the current loop iteration is completed. This constraint prohibits data manipulations that could improve cough detection, such as using more than one detection method to identify coughs. An additional, troubling limitation in the current prototype is that the software occasionally and seemingly randomly returns an overflow error. This error does not seem to be related to the number or loudness of coughs or to the volume of background noise. In some cases, the software runs for hours with zero issues, though in others (and much less frequently) it errors out in a number of minutes. The continuous cough detection code has been placed inside of a try-except structure so audio monitoring simply restarts after this error, though this is not an ideal patch because it takes several seconds for monitoring to resume. Further, it seems that once software errors out, it is much more likely to present the error again during that trial.

There are two possible sources for this error: the software or the soundcard in the microphone. From **Supplementary Figure 1** it seems that, while the average iteration time is

less than 2 ms, it occasionally takes much longer to run. These high loop iteration times, inexplicably, do not seem to be dependent on the number of coughs detected. Regardless, a large delay in processing could foreseeably lead to the overflow error. The error might also come from incompatibilities between the RPi and the soundcard. After all, the Microsoft Lifecam Cinema is intended for computers with at least an Intel Dual-Core 1.6 GHz CPU and 1 GB of RAM that have a Windows operating system. The Sabrent 3D audio soundcard has lower system requirements of a 133 MHz single core CPU and 32 MB of RAM (minimum requirements for Windows 2000), though the prototype experienced the overflow error with a similar frequency. However, the soundcard is cheaper and might not be able to faithfully sample at 44,100 Hz in the first place.

Conclusions:

Future directions:

The software currently cannot distinguish between two coughs if they are closer than 50 ms apart. As can be seen from the several cough spectrograms included in this report, the power in several high frequencies appears to be greatest at the start of a cough. Consequently, large changes in power within a spectrogram that inappropriately contains several coughs might be used to identify individual coughs from a stream. While this did seem to work reasonably well when attempted in Matlab, longer coughs (lasting longer than 500 ms) were often inappropriately segmented into 2 or more, smaller cough sections. Because these long coughs tended to happen more frequently than cough intervals of less than 50 ms, the software does not currently use this feature.

Additionally, some parts of the code may be replaced with separate hardware components to diminish the burden on the CPU. Currently the software produces an FFT every 512 samples regardless of their frequency content. Instead, three parallel band-pass filters could be connected to the microphone output and could monitor power in these frequency

bands without taxing the CPU. The filters could be connected to two AND logic gates that could indicate when the device should start performing FFTs on the audio data.

Another issue stems from how the audio signals are clipped to match the cough template. As discussed earlier, the present method uses a simple truncation to match the size of the coughs and fails to include information from the end of the sounds. Currently it seems that most of the information is contained in the beginning of the cough, but it may be possible to use the behavior near the end of the sound event as well to help distinguish coughs from other sounds, such as musical beats and laughs. Perhaps characterizing both the behaviors in the beginning and end of the cough may also help in identifying multiple coughs within a single audio stream.

Currently, the algorithm finds the standard deviation of the difference between an audio event and template spectrogram. However, some regions of the cough spectrograms are more highly conserved than others. These more consistent areas could be weighted higher than the others to improve cough detection. A simple scheme for accomplishing this would be to find the standard deviation between coughs at every point in the template spectrogram, and to then weight each point by the inverse of its standard deviation. However, this would give high weight to areas in the spectrogram of a cough where background noise dominates, since these regions are highly conserved. So, this weight could also be multiplied by the intensity of the point to offset the high weight of background noise.

Price and Marketing:

The Raspberry Pi B+ unit used retails for $40, but buying whole units for every monitoring device is inefficient. For this design, only the Broadcom BCM2835 SoC chip (containing the CPU, GPU, and display outputs) and the 512 MB SDRAM (<$10) are required. Unfortunately, a sales representative for Broadcom has informed us that the BCM 2835 chips are not for individual sale. Alternatively, a wholesale price could be negotiated with the Raspberry Pi Foundation to retain the benefits of using a premade, no-assembly-required

device. It is safe to assume that a cost reduction could be arranged, especially if the device would advertise its use of the RPi and thus provide marketing value to the Foundation.

The other primary cost driver of the device is the microphone. The prototype uses a webcam microphone that, though high-quality and accurate, has an upper range of only 8 kHz. Furthermore, this webcam costs around $70, likely due to the irrelevant video functionality. Accordingly, other microphones were researched that would be more appropriate for a refined future design. The best option discovered so far is the Blue Snowflake, a simple, high-quality USB microphone with an upper range of 20 kHz, a sample rate of 44.1 kHz, and a retail price of only $41[10]. As in the case of the RPi, it would likely be possible to negotiate a significantly lower wholesale price with the manufacturer (likely around $30).

There are other significant costs involved in the hypothetical final device, such as: internal wiring, LED screen, switches, a power supply, and the costs of manufacturing and assembly into a single consolidated unit. These should not exceed a sum total of $30, contributing to a final production cost estimate of $90 per unit. This estimate is designed to be high, as economies of scale and wholesale parts prices would surely have a greater effect than assumed here. Nonetheless, this estimate provides a valuable tool in assessing the device's marketability, and also falls well within the maximum price of $200 specified in the Progress Report. Based on this value, a sale price of at least $100 should be set in order to incur reasonable profits. Theoretically, the device could come to be subsidized by patient health insurance since it prevents doctor's office visits and hospitalizations.

From 2010-2012, 9.4% of Americans under the age of 18 had asthma[11] . According to the US Census Bureau, the US population on July 4th, 2011 was 311,602,811. In that year, 12.86% of males and 12.27% of females were under 18[12]. Assuming an equal amount of men and women among the population, this corresponds to a total population of American asthmatics under 18 of approximately 3.7 million in that year. Estimates for the prevalence of nocturnal asthma within the asthmatic population range from 47-75% [2], with the lower bound

found in a specific study on children.[13] This proportion results in a reduced population of 1.74 million American children with nocturnal asthma. In market sizing, one must also take into account the willingness and/or ability of potential consumers to purchase the device based on its price and value. Accordingly, data from a study on the correlation of asthma prevalence and severity with socioeconomic status was analyzed to reveal that 46.2% of children with nocturnal asthma in the survey came from homes that would be unlikely to be able or willing to pay for the device.[14] Specifically, this value is the percentage of the study's nocturnal asthmatics that came from homes in the two lowest Townsend Deprivation Index quintiles of the survey population. The Townsend Deprivation Index is a widely used indicator of a household's material deprivation based upon census data.[15] This brings the final market size down to approximately 936,000 American children with nocturnal asthma whose parents would potentially buy the device.

Intellectual Property and Ethics Considerations:

The design does contain intellectual property, specifically in the software process it uses to detect and count coughs. Extensive literature searches conducted throughout the project have proven that the algorithm developed here is novel, as even the few existing asthma monitors which do analyze cough sounds use different, less robust processes to identify cough events. The asthma attack alarm functionality is also a novel feature, and provides critically important information that enhances not only the understanding of a patient's condition but also her/his safety and quality of life. For these reasons, the design could qualify for a utility patent as a new and useful process and machine. The primary claim would be the invention's ability to use audio data alone to accurately and specifically count coughs, calculate cough frequency throughout each night, and trigger an alarm if this frequency exceeds a dangerous threshold.

Whether or not this group will pursue protection of this IP has yet to be decided, and will depend heavily upon the members' personal plans regarding future goals and time commitments. There is an altruistic and ethical argument to be made for releasing the process

without protection for the sake of pushing the field of asthma monitoring forward. Any patent filed would likely be a Provisional Patent, due to the need for further hardware development before the design could be considered a commercial product. Another potential protection would be applying for a registered copyright on the Python code used to analyze the cough signal. This software process is the core functionality of the design, and embodies its primary contribution to the field of asthma monitoring.

There are other areas in which ethics must be considered in the design as well. As in all cases in which patient data is recorded, patient privacy and data security must be secured. Since the device continuously monitors audio throughout the course of each night, layperson patients may have fears that it is recording speech as well as cough frequency. While such misconceptions can be objectively refuted by explaining the device's capabilities, there are potentially legitimate ethical concerns. The most evident of these is that, since the device monitors asthma symptoms that have been linked to poor daytime performance, data from the device could be used to discriminate against patients in employment or other situations. Importantly, however, the device does not report to any external devices or databases. Therefore, the information is wholly under the control of the user, who thus has no reason to fear its improper dissemination.

FDA Regulation

Since the device could loosely be thought of as a medical device, the potential requirement of FDA approval must be considered. This seems increasingly likely since a Premarket Notification [501(k)] filing and approval of the VitaloJAK device—an ambulatory cough monitor investigated as an existing solution in the Preliminary Report—was found.[16] In this filing, the VitaloJAK is classified as a Class II device, specifically a "Recorder, magnetic tape, medical," or 21 CFR 870.2800. While our device is very similar to the VitaloJAK in concept, this classification is obviously not applicable. Instead, 21 CFR 870.2050, or "Biopotential amplifier and signal conditioner" is a more suitable and intuitive designation for our

design.[17] As such, the device would require the filing of a 501(k), as it is not exempted, and would be subject to “other General Controls such as registration and listing, labeling, and good manufacturing processes….”[18] FDA approval is a time consuming process that should ideally be avoided given the nature of our device as a completely non-invasive and safe consumer product that claims no diagnostic or therapeutic abilities. However, in light of the VitaloJAK’s need for FDA approval—despite its inventors also using the “non-diagnostic” disclaimer—the possibility that the device may need to be approved must be considered.

Bibliography:

[1] Braman, S. S. (2006). The global burden of asthma. Chest Journal,130(1_suppl), 4S-12S.

[2] Ginsberg, D. (2009). An Unidentified Monster in the Bed–Assessing Nocturnal Asthma in Children. McGill Journal of Medicine: MJM, 12(1), 31.

[3] Chugh IM, Khanna P, Shah A. Nocturnal symptoms and sleep disturbances in clinically stable asthmatic children. Asian Pacific journal of allergy and immunology / launched by the Allergy and Immunology Society of Thailand. 2006 Jun–Sep;24:2–3. 135–42.

[4] Bentur L., Beck, R., et al. Wheeze monitoring in children for assessment of nocturnal asthma and response to therapy. European Respiratory Journal 2003; 21: 621-26.

[5] Stores G, Ellis AJ, Wiggs L, et al. Sleep and psychological disturbance in nocturnal asthma. Archives of disease in childhood. 1998 May;78(5):413–9.

[6] "Raspberry Pi B Specification Sheet." *Adafruit*. Web. 23 Nov. 2014. <https://www.adafruit.com/datasheets/pi-specs.pdf>.

[7] "Config.txt." *Raspberry Pi Documentation*. Web. 23 Nov. 2014. <http://www.raspberrypi.org/documentation/configuration/config-txt.md>.

[8] "LifeCam Cinema." Microsoft Technical Datasheet. Web. 25 Nov. 2014. < https://www.1000ordi.ch/microsoft-lifecam-cinema-h5d-00003-47338_en.pdf>

[9] "Understanding FFT Windows." LDS Group, 1 Jan. 2003. Web. 21 Nov. 2014. <http://www.physik.uni-wuerzburg.de/~praktiku/Anleitung/Fremde/ANO14.pdf>.

[10] "Blue Microphones | Snowflake - Head and Shoulders Above Any Comparable Portable USB on the Market!" Blue Microphones. 1 Jan. 2013. Web. 25 Nov. 2014. http://www.bluemic.com/snowflake/#/desc.

[11] "Morbidity and Mortality Report," NCHS, U.S. CDC, 2003. 25 Nov. 2014.

[12] "United States Census Bureau." Population Clock. Web. 24 Nov. 2014. http://www.census.gov/popclock.

[13] Meijer, Gerda G., et al. "Frequency of nocturnal symptoms in asthmatic children attending a hospital out-patient clinic." European Respiratory Journal 8.12 (1995): 2076-2080.

[14] Kwong, G. Ng Man, et al. "Diagnostic and treatment behaviour in children with chronic respiratory symptoms: relationship with socioeconomic factors." Thorax 57.8 (2002): 701-704.

[15] "GEO-REFER Learning Resources Repository." GEO-REFER Learning Resources Repository. 1 Jan. 2008. Web. 30 Nov. 2014.

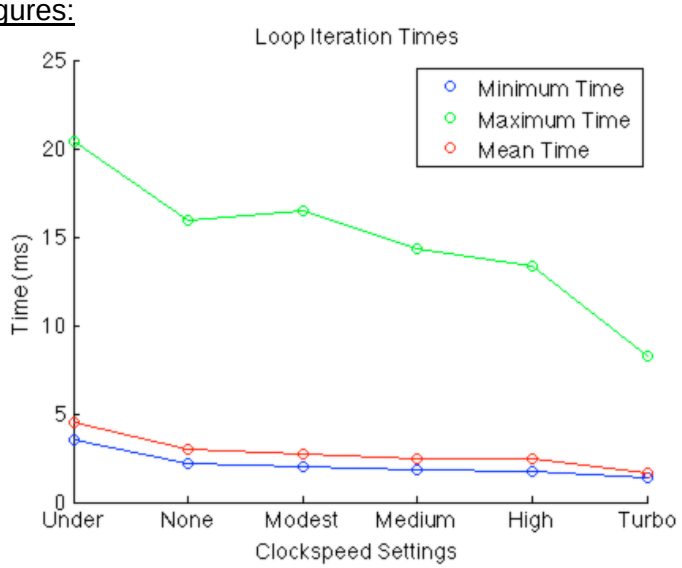
[16] “501k Summary: K110525” U.S. Food and Drug Administration Center for Devices and Radiological Health. 23 Nov. 2011. Web. 29 Nov. 2014. http://www.accessdata.fda.gov/cdrh_docs/pdf11/K110525.pdf

[17]"CFR - Code of Federal Regulations Title 21." CFR - Code of Federal Regulations Title 21. 1 Sept. 2014. Web. 29 Nov. 2014. http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/cfrsearch.cfm?fr=870.2050.

[18] "Device Classification Panels." U.S. Food and Drug Administration, 26 June 2014. Web. 29 Nov. 2014. http://www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/Overview/ClassifyYourDevice/ucm051530.htm.

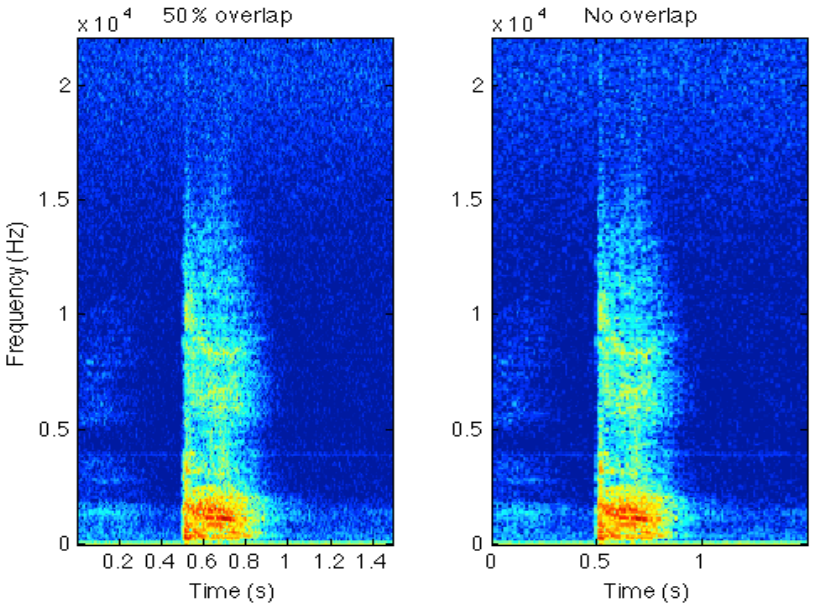
Appendix:

Supplementary Figures:

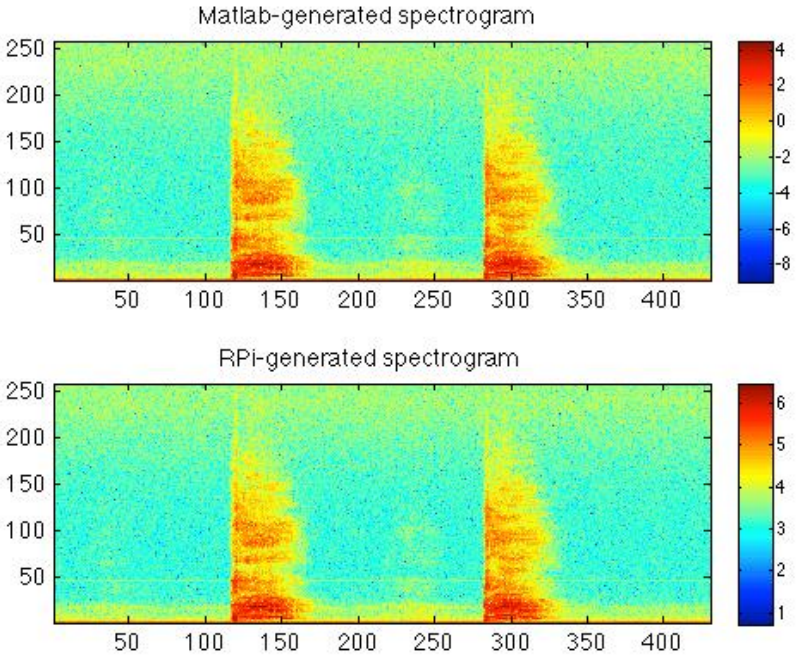


Clockspeed	Minimum Time (ms)	Maximum Time (ms)	Mean Time (ms)
Under	3.57	20.43	4.54
None	2.20	15.90	2.98
Modest	2.05	16.49	2.74
Medium	1.82	14.32	2.48
High	1.76	13.36	2.45
Turbo	1.35	8.23	1.70

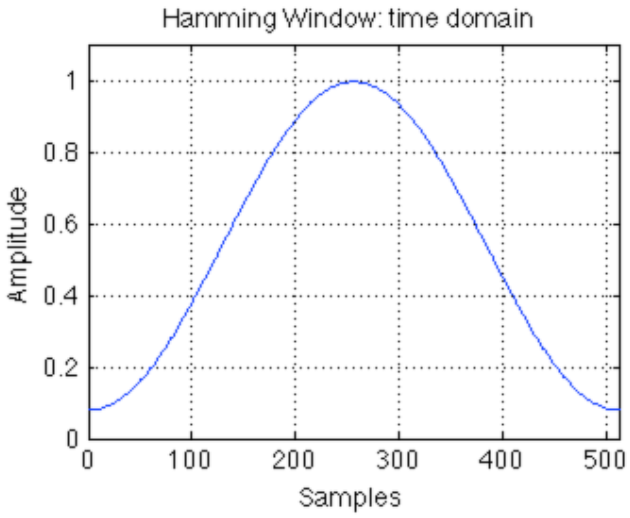
Supplementary Figure 1: Loop iteration speeds of all clockspeed settings that successfully ran the software.



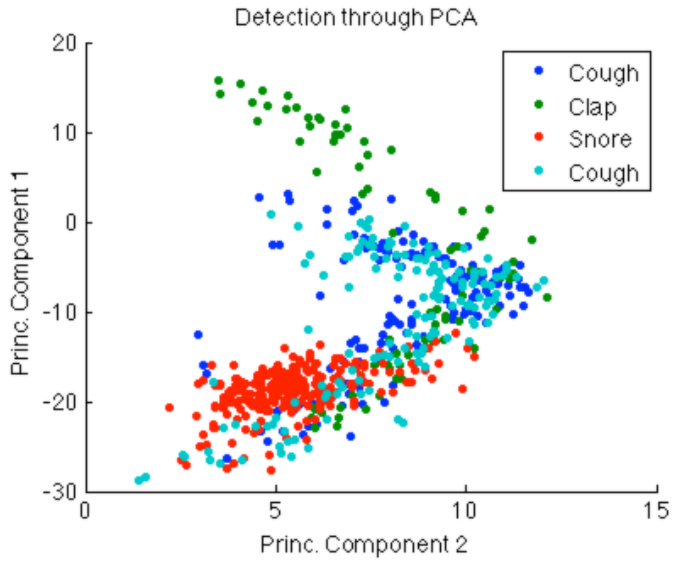
Supplementary Figure 2: Spectrograms of the same cough using 50% and 0% overlap windows. Color bar ranges from -3 to 5 arbitrary units.



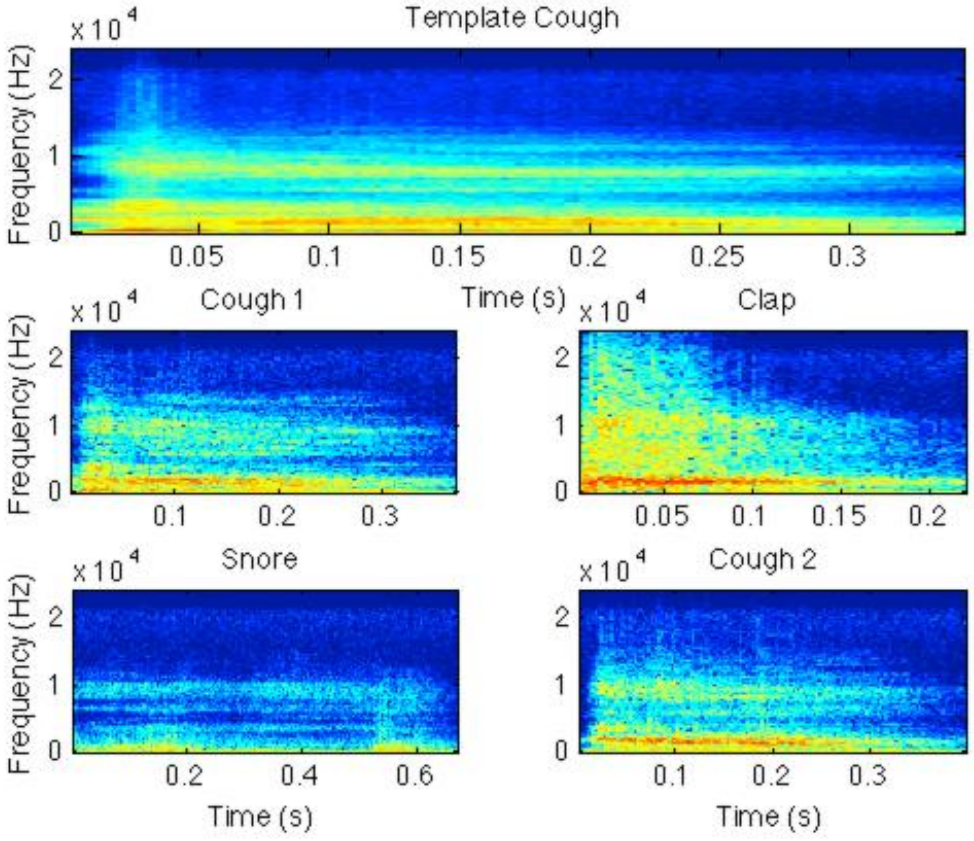
Supplementary Figure 3: The two coughs were recorded and saved as a .wav file, which was then read in by Matlab and processed by the "spectrogram" command (top) and by the Python code using the iterative FFT method. Note that, while the two plots are essentially identical, the color bars are different. This is because the spectrogram command simply scales the data differently



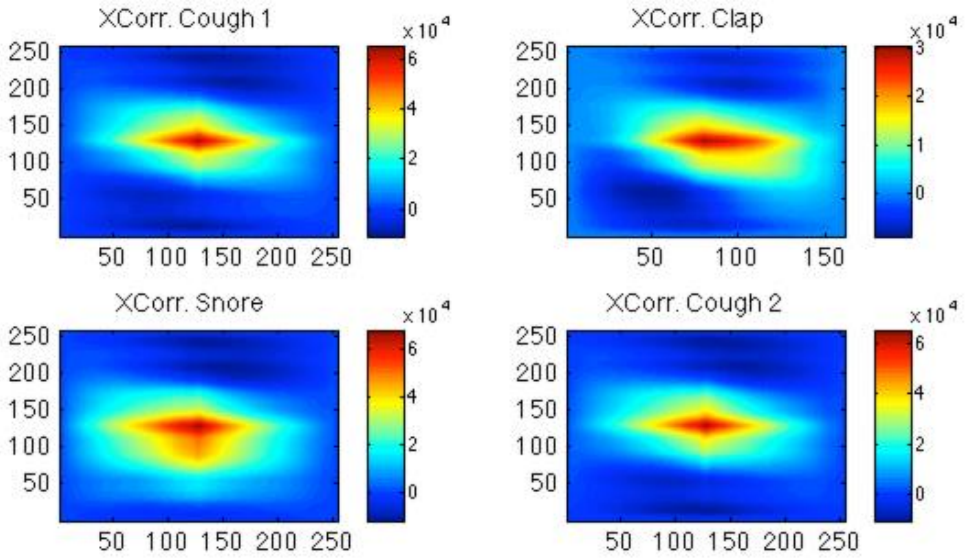
Supplementary Figure 4: The Hamming window function used to weight the 512 audio samples.



Supplementary Figure 5: The projection of four sound events onto the eigenspace of a template cough. Princ. Component 1 had 75.9% of the variance and Princ. Component 2 had 8.9% of the variance of the template cough spectrogram.



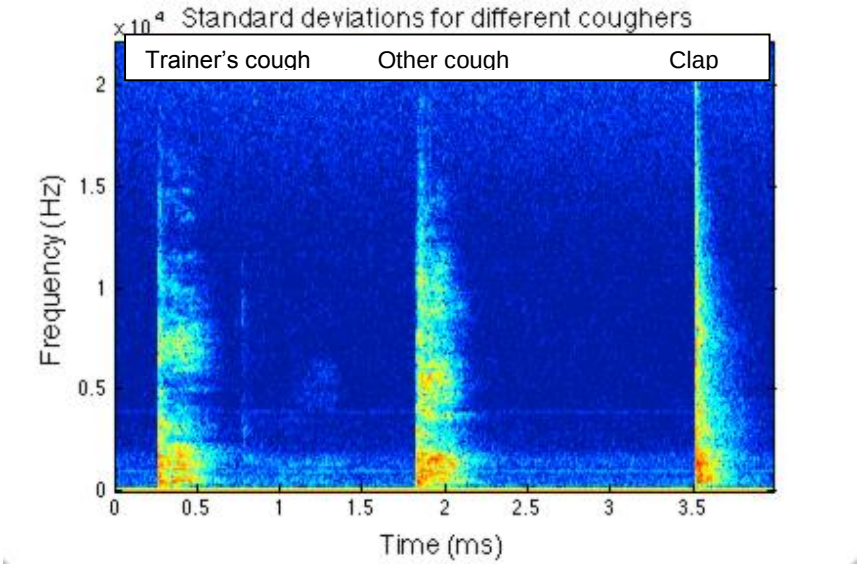
Supplementary Figure 6: The figure illustrates the spectrograms of the template cough and four sound events: two coughs, one clap, and one snore. Color bar ranges from -3 to 5 arbitrary units.



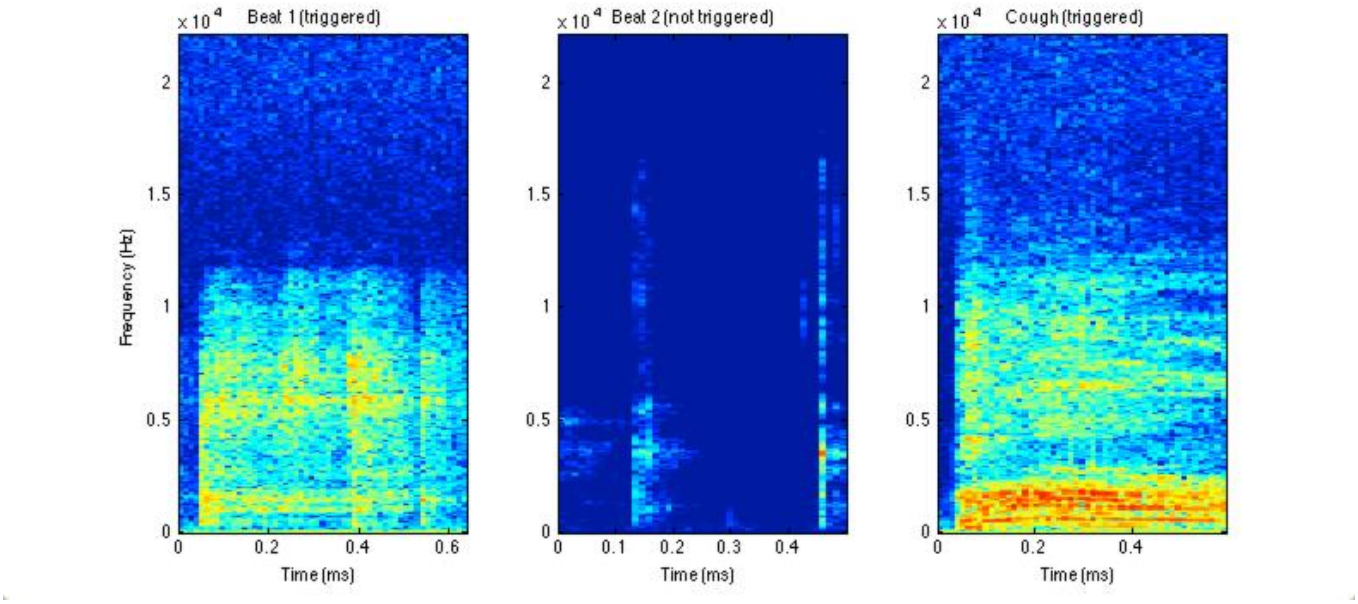
Supplementary Figure 7: A two-dimensional color map of cross-correlation between the template cough and four sound events. Note the highest intensity is at the center.

Supplementary Table 1: Normalized scalar projects of 4 sound events onto a template cough.

Sound event	Normalize scalar projection
Cough 1	0.9313
Clap	0.7551
Snore	0.8299
Cough 2	0.9259



Supplementary Figure 9: This figure illustrates the spectrogram of the template trainer's cough, another group member's cough, and a clap. These three audio events had standard deviations as follows: 0.4566, 0.4770, and 0.54032. These standard deviations are lower than others in this report because they were calculated by the Raspberry Pi code, whose spectrograms have a smaller range of values. Color bar ranges from -3 to 5 arbitrary units



Supplementary Figure 10: This figure shows a typical musical beat that triggered a false alarm in the detector, a beat that did not trigger the software, and an actual cough. Color bar ranges from -3 to 5 arbitrary units.

Python Software:

Cough Detection Program

```
################################## Cough Detection Program ###################################
#Import necessary modules:
import pyaudio, numpy as np, RPi.GPIO as GPIO, matplotlib.pylab as plt
from time import time/Users/William/Downloads/Continuous_2v6.py
from array import array

##############################################################################################
#Set up pins to shine LEDs
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT) #Shines when cough is detected
GPIO.setup(11, GPIO.OUT) #Shines warning signal

#variables intialized to zero and are discussed later in the code
ci=seg_start=z_count=coughs_per_time=previous_count=coughs_counted=light_on=l_on_passes=0
cough_freq_thresh=20 #if coughs per minute is greater than this, sound alarm

#stds=[] #give option to save standard deviations for audio events
min_width=12 #collected signal threshold width
min_width2=200 #Template max width
widths=np.array([0,0]) #Holds collected signal width and template width

template=np.loadtxt("template1") #import template from cougher 1
##template=np.loadtxt("template2") #import template from cougher 2

chunk = 512 #Number of audio samples per loop iteration
FORMAT = pyaudio.paInt16 #audio read in as 16 bit integer
CHANNELS = 1 #Just recording, so only need 1 channel
RATE = 44100 #Recording rate in samples/second

fft_mat=np.zeros(((chunk/2)+1,(RATE/chunk)*5)) #initialize an empty spectrogram matrix
window=np.hamming(chunk) #A Hamming window function of length=chunk (512 in this case)

p = pyaudio.PyAudio() #Initialize the recording object
print('Hello! Opening audio stream')
stream = p.open(format=FORMAT, #open a stream to continuously record audio
                channels=CHANNELS,
                rate=RATE,
                input=True,
                output=True,
                frames_per_buffer=chunk)
##############################################################################################
start_recording=time() #indicates when recording started
#for i in range(0,passes): #useful when testing code to not have infinite loop

while(True):

##  try: #a try statement to account for the random overflow error discussed in report
    data = stream.read(chunk) #A buffer that fills with samples and only procedes when
                              #number of samples = chunk
    #Turn off cough detection LED after it has been on for a few passes
    if light_on==1:
        l_on_passes+=1
        if l_on_passes>=3:
            GPIO.output(7, False)
            light_on=0
            l_on_passes=0
```

```
    data=np.fromstring(data,'Int16')*window #Convert from string to numeric and weight by Hamming window
    fftdata=np.log10(abs(np.fft.rfft(data))+1) #Find magnitude of complex values and scale logarithmically

    #If power in three frequency bands is above thesholds, add FFT as a column in spectrogram matrix
    if np.mean(fftdata[15-1:15+2])>=2 and np.mean(fftdata[80-1:80+2])>=1.5 and np.mean(fftdata[100-1:100+2])>=1.2:
        seg_start=1 #signals that the spectrogram matrix is being filled
        z_count=0 #the number of loop iterations since the FFT failed the above test reset to zero
        fft_mat[:,ci]=fftdata #spectrogram matrix filled at column indicated by ci
        ci+=1 #the next loop iteration will fill the next column in the spectrogram matrix
    elif z_count>=4: #it has been about 50 ms since FFT last passed the categorization test
        seg_start=0 #turn off spectrogram matrix filling
        z_count=0
        if fft_mat[:,0:ci].shape[1]>min_width: #if numbers of rows in spectrogram matrix above threshold,
                                               #save the spectrogram in the coughs structured array
            col_sig=fft_mat[:,0:ci] #collect event spectrogram from fft_mat
            widths[:]=[col_sig.shape[1], template.shape[1]]
            #Cut the event spectrogram and the template spectrogram to the shorter number of columns,
            #then, flatten each to 1D array and take the difference. Next, take standard deviation.
            st_dev=np.std(np.subtract(col_sig[:,0:widths.min()].flatten(), template[:,0:widths.min()].flatten()))
            #stds.append(st_dev) #useful to save standard deviations when determining detection threshold
            if st_dev<=.54: #if standard deviation below this threshold, event is a cough
                coughs_counted+=1 #count cough
                GPIO.output(7, True) #turn on cough detection LED
                light_on=1
                l_on_passes=0
        ci=0 #the next time an FFT passes the categorization test, it will fill first
             #column of spectrogram matrix

    elif seg_start==1: #spectrogram matrix was filled less than 50 ms ago,
                       #though FFT failed categorization test this iteration
        z_count+=1
        fft_mat[:,ci]=fftdata
        ci+=1

    end=time()
    if end-start_recording >= 60: #find the coughs that take place in every minute bin
        coughs_per_time=(coughs_counted-previous_count)
        previous_count=coughs_counted
        start_recording=time()
    if coughs_per_time >= cough_freq_thresh: #if cough frequency above threshold, sound alarm
        GPIO.output(11, True) #turn on alarm LED

#stop audio recording
stream.stop_stream()
stream.close()
p.terminate()

print('DONE!')
#Give option to see total coughs
#print(coughs_counted)
```

Cough Template Creation:

```
################################## Template Creation ######################################
#Import necessary modules:
import pyaudio, numpy as np, matplotlib.pylab as plt
from array import array

#############################################################################################
ci=seg_start=z_count=0 #variables intialized to zero and are discussed later in code
coughs=[] #List of coughs
min_width=12 #threshold number of rows in spectrogram matrix for it to be a cough
min_width2=200 #Starting number of rows in empty cough template

chunk = 512 #Number of audio samples per loop iteration
FORMAT = pyaudio.paInt16 #audio read in as 16 bit integer
CHANNELS = 1 #Just recording, so only need 1 channel
RATE = 44100 #Recording rate in samples/second
max_cough_length = min_width2*5 #The maximum number of rows a spectrogram matrix can have.
                        #5 times more than the template width as a safety factor
fft_mat=np.zeros(((chunk/2)+1,max_cough_length)) #Initialize an empty spectrogram matrix
window=np.hamming(chunk) #A Hamming window function of length=chunk (512 in this case)
p = pyaudio.PyAudio() #Initialize the recording object
print('Hello! Opening audio stream')
stream = p.open(format=FORMAT, #Open a stream to continuously record audio
            channels=CHANNELS,
            rate=RATE,
            input=True,
            output=True,
            frames_per_buffer=chunk)

#############################################################################################
#Continuous recording loop.
while(len(coughs)<10): #Records continuously until 10 coughs have been detected
    data = stream.read(chunk) #A buffer that fills with samples and only procedes when
                        #number of samples = chunk
    data=np.fromstring(data,'Int16')*window #Convert from string to numeric and weight by Hamming window
    fftdata=np.log10(abs(np.fft.rfft(data))+1) #Find magnitude of complex values and scale logarithmically

    #If power in three frequency bands is above thesholds, add FFT as a column in spectrogram matrix
    if np.mean(fftdata[15-1:15+2])>=2 and np.mean(fftdata[80-1:80+2])>=1.5 and np.mean(fftdata[100-1:100+2])>=1.2:
        seg_start=1 #signals that the spectrogram matrix is being filled
        z_count=0 #the number of loop iterations since the FFT failed the above test reset to zero
        fft_mat[:,ci]=fftdata #spectrogram matrix filled at column indicated by ci
        ci+=1 #the next loop iteration will fill the next column in the spectrogram matrix
    elif z_count>=4: #it has been about 50 ms since FFT last passed the categorization test
        seg_start=0  #turn off spectrogram matrix filling
        z_count=0
        if fft_mat[:,0:ci].shape[1]>min_width:  #if numbers of rows in spectrogram matrix above threshold,
                                    #save the spectrogram in the coughs structured array
            coughs.append(fft_mat[:,0:ci])
        ci=0 #the next time an FFT passes the categorization test,
            #it will fill the first column of the spectrogram matrix

    elif seg_start==1: #spectrogram matrix was filled less than 50 ms ago,
                    #though FFT failed categorization test this iteration
        z_count+=1
        fft_mat[:,ci]=fftdata #spectrogram matrix filled at column indicated by ci
        ci+=1

#Stop audio recording
```

```
stream.stop_stream()
stream.close()
p.terminate()

##########################################################################################################
#
#Average saved coughs and permanently save the resulting template
print('Computing template')
for kk in range(10):
    if min_width2>coughs[kk].shape[1]:
        min_width2=coughs[kk].shape[1]
    mat_sums=np.zeros(((chunk/2)+1,min_width2))
for kk in range(10):
    mat_sums=np.add(mat_sums, coughs[kk][:,0:min_width2])
template=mat_sums/10
np.savetxt("template4", template)



#Give the option to visualize the new template
#imshow(template, interpolation='bilinear', origin='lower')
#show()
```

Matlab Software (used to easily test different detection schemes):

```
function cough_demo_new
% Set Defaults
set(0,'DefaultUicontrolunits','pixels');
set(0,'defaultuicontrolbackgroundcolor',[.9 .9 .9]);
bc = [.8 .8 .8];
%====================Button Sizes & Spacing====================&
% Set Button Sizes
bw=110; bh=20;
fBx=0; fBy=300; % Move Folders
%============================Create Buttons============================%
% Figure
h.fig = figure('position', [550 300 800 420],'color',[0.9 0.9 0.9]); % [550 300 700/800 550]
% Template Axis
h.axes_temp=axes('units','pixels','Position', [40 50 350 200]);title('Template'); % [300 325 350 200]
% Spectrogram Axis
h.axes_spec=axes('units','pixels','Position', [430 50 350 200]);title('Spectrogram'); % [300 50 350 200]

% Template Panel
h.template_panel = uipanel('units','pixels','backgroundcolor','white','title','Template',...
    'position',[fBx+40 fBy-20 1.1*bw 3*bh+70]);
% Create Template Button
h.create_template = uicontrol('parent',h.template_panel,'Style','pushbutton',...
    'String', 'Create Template','Position',[5 90 bw bh],'backgroundcolor',bc);
% Create Template Recording Time
h.time_template = uicontrol('parent',h.template_panel,'Style','edit','String', 'Template_Rec_Time',...
    'Position',[5 70 bw bh],'backgroundcolor','white');
% Load Template Button
h.template = uicontrol('parent',h.template_panel,'Style','pushbutton','String', 'Load Template',...
    'Position',[5 50 bw bh],'backgroundcolor',bc);
% Folder Name Text Box
h.template_disp = uicontrol('parent',h.template_panel,'style','text','backgroundcolor',[.95 .95 .95],...
    'position',[5 5 bw 45]);

% Audio Monitoring Panel
h.audio_panel = uipanel('units','pixels','backgroundcolor','white','title','Audio Monitor',...
    'position',[bw+60 fBy-20 2*bw+5 6*bh+10]);
% Monitoring Time
h.mon_time = uicontrol('parent',h.audio_panel,'style','edit','backgroundcolor','white',...
    'Position',[0 80 bw bh]);
h.mon_time_label = uicontrol('parent',h.audio_panel,'style','text','string','Monitoring Time',...
    'Position',[0 100 bw-20 bh-5],'backgroundcolor','white');
% Threshold
h.threshold = uicontrol('parent',h.audio_panel,'style','edit','backgroundcolor','white',...
    'Position',[0 50 bw bh],'string','1.5');
h.threshold_label = uicontrol('parent',h.audio_panel,'style','text','string','Threshold for std',...
    'Position',[0 70 bw-20 bh-10],'backgroundcolor','white');
h.thresh_dot = uicontrol('parent',h.audio_panel,'style','edit','backgroundcolor','white',...
    'Position',[bw 50 bw bh],'string','2e4');
h.thresh_dot_label = uicontrol('parent',h.audio_panel,'style','text','string','Threshold for dot',...
    'Position',[bw 70 bw-20 bh-10],'backgroundcolor','white');
h.thresh_choose = uicontrol('parent',h.audio_panel,'style','toggle','string','Dot Product',...
    'Position',[bw 80 bw bh],'backgroundcolor','white');
% Measured Standard Deviation & Dot Product
h.sound_std = uicontrol('parent',h.audio_panel,'style','text','backgroundcolor','white',...
    'Position',[0 20 bw-5 bh],'backgroundcolor',[.95 .95 .95]); % 'white'
h.sound_std_label = uicontrol('parent',h.audio_panel,'style','text','string','Sound std',...
    'Position',[0 40 bw-50 bh-10],'backgroundcolor','white');
h.sound_dot = uicontrol('parent',h.audio_panel,'style','text','backgroundcolor','white',...
    'Position',[bw 20 bw-5 bh],'backgroundcolor',[.95 .95 .95]); % 'white'
h.sound_dot_label = uicontrol('parent',h.audio_panel,'style','text','string','Sound dot',...
```

```
    'Position',[bw 40 bw-50 bh-10],'backgroundcolor','white');

% Message Boxes
h.message_gui = uicontrol('style','text','backgroundcolor','white',...
    'Position',[430 fBy+70 1.1*bw bh]);
h.message_gui_label = uicontrol('style','text','backgroundcolor',[.9 .9 .9],...
    'Position',[430-5 fBy+90 .5*bw bh],'string','Message:');
h.message_cough = uicontrol('style','text','backgroundcolor','white',...
    'Position',[430 fBy+27 1.1*bw bh]);
h.message_cough_label = uicontrol('style','text','backgroundcolor',[.9 .9 .9],...
    'Position',[430-10 fBy+55 bw .5*bh],'string','Number of Coughs:');
h.message_cough_r = uicontrol('style','text','backgroundcolor','white',...
    'Position',[430 fBy-15 1.1*bw bh]);
h.message_cough_r_label = uicontrol('style','text','backgroundcolor',[.9 .9 .9],...
    'Position',[430-2 fBy+10 .6*bw .6*bh],'string','Cough Rate:');
% Continuous Audio Record
h.monitor = uicontrol('Style','pushbutton','String', 'Monitor',...
    'Position',[650 fBy-10 bw+20 6*bh],'backgroundcolor',bc);

%============================Callbacks============================%
% Callback Function Calls
set(h.create_template,'callback',{@create_template,h});
set(h.template,'callback',{@load_template,h});
set(h.monitor,'callback',{@monitor,h});
set(h.thresh_choose,'callback',{@thresh_choose,h});

function h = create_template(hObject,eventdata,h)
% Clear Message Labels
set(h.message_gui,'string','');
set(h.message_cough,'string','');
set(h.message_cough_r,'string','');
% Get Recording Time
rec_time = str2double(get(h.time_template,'string'));
% Record Template
cough_record(hObject,eventdata,h,rec_time);
% Get Cough files
coughs_name = getappdata(0,'coughs_name');
% Create Weight Template
cough_weight(coughs_name);

function h = cough_record(hObject,eventdata,h,rec_time)
% Template & Coughs filenames
template_name = 'template_ttt.mat';setappdata(0,'template_name',template_name);
coughs_name = 'cough_ttt.mat';setappdata(0,'coughs_name',coughs_name);
% Recording Object
recObj=audiorecorder(44100, 16, 1);
record(recObj,rec_time);
pause(.5);
% Set counters & Initialize Collected Signal Matrix
prev_end=1;
jj=0;
col_sig=[];
coughs_counted=0;
previous_count=0;
% Display Start, Start 1st Timer
first_time=tic;
set(h.message_gui,'string','Start!');
% Loop to Record Continuously
while 1;
% Set 2nd Timer
second_time=tic;
% Retrieve Audio Data of most recent segment
```

```
y = getaudiodata(recObj);
y2=y(prev_end:end);
prev_end=length(y);
% Try-Catch for Spectrogram of recent audio data
try
   [S, F, T]=spectrogram(y2, 512, 0, 512, 44100, 'yaxis');assignin('base','F',F);assignin('base','T',T);
catch
   set(h.message_gui,'string','Catch');
   break
end
% Log of Spectrogram
S2=log(abs(S));
% S2(S2<=-6)=-6; % Cutoff values below certain intensity
%%%%%%%Visualize spectrogram snapshot%%%%%%%%%%%%%
axes(h.axes_temp);imagesc(T,F,S2);
set(gca,'YDir','normal');colorbar;
%caxis([-15 1]);
set(gca, 'YTickLabel',[]);set(gca, 'XTickLabel',[]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initial Frequency Test: check if there is power in bands characteristic
% of cough
c_range=find(mean(S2(14:16,:))>-2. & mean(S2(79:81,:))>-2 & mean(S2(99:101,:))>-3);
if ~isempty(c_range);
      set(h.message_gui,'string','');pause(.01);
      set(h.message_gui,'string','Something');
      col_sig=[col_sig S2(:,c_range(1):c_range(end))];
   if size(col_sig,2)>3
         coughs_counted=coughs_counted+1;
         set(h.message_cough,'string',sprintf('%d coughs counted!', coughs_counted));
         jj=jj+1;
         coughs{jj}=col_sig;
         set(h.message_gui,'string','Here');

%%%%%%% Update template %%%%%%%%%%%%%%%
      if jj==10;
         min_width=200;
         for kk=1:10
            if min_width>size(coughs{kk},2)
               min_width=size(coughs{kk},2);
            end
         end
         pause(.001);
         mat_sums=zeros(257,min_width);
         for kk=1:10
            mat_sums=mat_sums+coughs{kk}(:,1:min_width);
         end
         template=mat_sums./10;
         save(template_name,'template');
         save(coughs_name,'coughs');
         clearvars coughs
      end
   end
   col_sig=[];
end

% Break Recording Loop if recording object is no longer recording
if ~recObj.isrecording;
   break
end
% Calculate Time & Prnt Cough Rate
if toc(first_time)>10
```

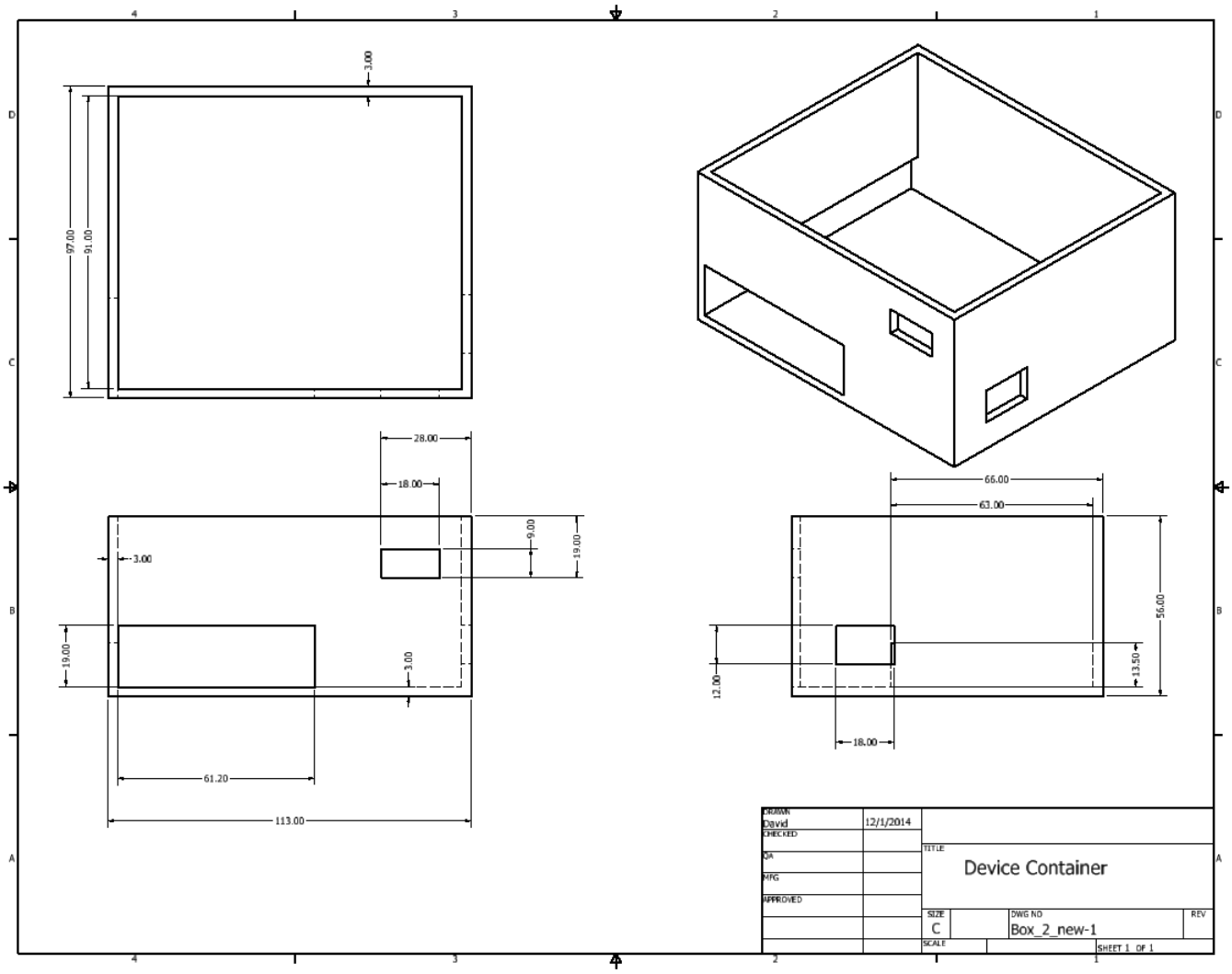
```
    set(h.message_cough_r,'string',sprintf('%d coughs in 10 seconds', coughs_counted-previous_count));
    previous_count=coughs_counted;
    first_time=tic; % Reset counter
end

n=toc(second_time); % Total Time for loop
pause(0.1-toc(second_time));
end

set(h.message_gui,'string','Done!');

function h = load_template(hObject,eventdata,h)
% Brings up pop menu to select template folder
[filename,folder_name] = uigetfile('*.*','C:\Users\David\Desktop\Senior Design Recordings');
set(h.template_disp,'string',[folder_name filename]);
% Load Template Data & share to GUI
template_cough=load(filename);
template_cough=template_cough.template;
% Plot Template
axes(h.axes_temp);
imagesc(template_cough);colorbar;set(gca,'YDir','normal');
xlabel('Time');ylabel('Frequency');title('Spectrogram of Template Cough');
% Set Data
setappdata(0,'template_cough',template_cough);
assignin('base','template_cough',template_cough);

function h = monitor(hObject,evetdata,h)
% Clear Message Labels
set(h.message_gui,'string','');
set(h.message_cough,'string','');
set(h.message_cough_r,'string','');
% Get Parameters
thresh = str2double(get(h.threshold,'string')); % get threshold value for std
thresh_dot = str2double(get(h.thresh_dot,'string')); % get threshold value for dot
rec_time = str2double(get(h.mon_time,'string')); % get monitoring/recording time
% load('template2.mat');
template_cough=getappdata(0,'template_cough');

% Set up recording object
recObj=audiorecorder(44100, 16, 1);
record(recObj,rec_time);
pause(.5);
% Set Counter & Initialize Collected Signal Matrix
prev_end=1;
jj=0;
col_sig=[];
coughs_counted=0;
previous_count=0;
first_time=tic;
set(h.message_gui,'string','Start');
% Sound, Std Dev, and Dot Product Vectors
snd_mat = [];
st_dev_mat = [];
dot_prod_mat = [];
while 1;
second_time=tic;
y = getaudiodata(recObj);
y2=y(prev_end:end);
prev_end=length(y);
try
    [S, F, T]=spectrogram(y2, 512, 0, 512, 44100, 'yaxis');
catch
```

```
    set(h.message_gui,'string','Catch');
    break
end
S2=log(abs(S));
S2(S2<=-6)=-6; % Intensity Cutoff
snd_mat = [snd_mat, S2];

%%%%%%%Visualize spectrogram snapshot%%%%%%%%%%%%
axes(h.axes_spec);imagesc(T,F,S2);colorbar;
set(gca,'YDir','normal');title('Spectrogram of Monitored Sound');
caxis([-15 1]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test Signal power in specific frequency bands
c_range=find(mean(S2(14:16,:))>-2. & mean(S2(79:81,:))>-2 & mean(S2(99:101,:))>-3);
if ~isempty(c_range);
    set(h.message_gui,'string','');pause(.1);
    set(h.message_gui,'string','Something');
    col_sig=[col_sig S2(:,c_range(1):c_range(end))];
elseif ~isempty(col_sig)
    if size(col_sig,2)>25
        small_width=sort([size(col_sig,2),size(template_cough,2)-1]);
        % Standard Deviation
%       w = load('weight_mat.mat'); % Testing weights
%       weight_mat = w.weight_mat;
        sig_diff = (col_sig(:,1:small_width(1))-template_cough(:,1:small_width(1)));%.*weight_mat(:,1:small_width(1));
        total_diff=sum(sum(abs(sig_diff)));
        st_dev = std(reshape(sig_diff,[1 numel(sig_diff)]));
        st_dev_mat = [st_dev_mat, st_dev];
        % Dot Product
        cough_test = reshape(col_sig(:,1:small_width(1)),[1 numel(col_sig(:,1:small_width(1)))]);
        cough_template = reshape(template_cough(:,1:small_width(1)),[1 numel(template_cough(:,1:small_width(1)))]);
        dot_prod = dot(cough_test,cough_template);
        dot_prod_mat = [dot_prod_mat, dot_prod];
        % Set Values in Gui
        set(h.sound_std,'string',num2str(st_dev)); % Show standard deviation of sound
        set(h.sound_dot,'string',num2str(dot_prod)); % Show dot product of sound
        % Test sound as cough
         if get(h.thresh_choose,'value')
            if st_dev<thresh;
                coughs_counted=coughs_counted+1;
                set(h.message_cough,'string',sprintf('%d coughs counted!', coughs_counted));
                jj=jj+1;
                coughs{jj}=col_sig;

            else
                set(h.message_gui,'string','');pause(.1);
                set(h.message_gui,'string','Almost!');
            end
         elseif get(h.thresh_choose,'value')==0
            if dot_prod>thresh_dot
                coughs_counted=coughs_counted+1;
                set(h.message_cough,'string',sprintf('%d coughs counted!', coughs_counted));
                jj=jj+1;
                coughs{jj}=col_sig;
            else
                set(h.message_gui,'string','');pause(.1);
                set(h.message_gui,'string','Almost!');
            end
         end
%%%%%%% Update template %%%%%%%%%%%%%%%
        if jj==10;
            min_width=200;
```

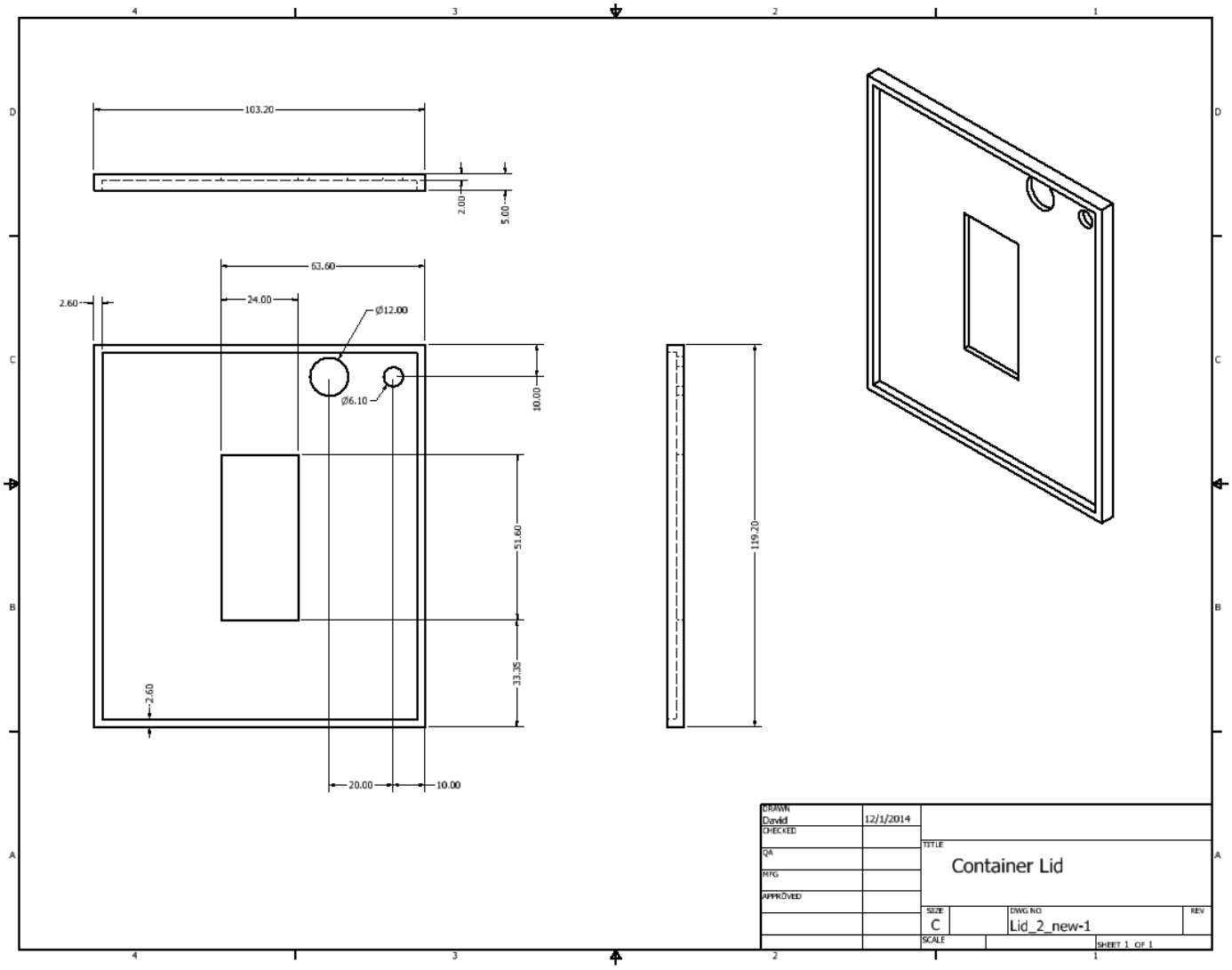
```
            for kk=1:10
                if min_width>size(coughs{kk},2)
                    min_width=size(coughs{kk},2);
                end
            end
            mat_sums=zeros(129,min_width);
            for kk=1:10
                mat_sums=mat_sums+coughs{kk}(:,1:min_width);
            end
            template_cough=mat_sums./10;
            save('template_monitor.mat','template_cough');
            clearvars coughs
        end

    end

    col_sig=[];
end
% Break Loop
if ~recObj.isrecording;
    break
end

if toc(first_time)>10
    set(h.message_cough_r,'string',sprintf('%d coughs in 10 seconds', coughs_counted-previous_count));
    previous_count=coughs_counted;
    first_time=tic;
end

n=toc(second_time);
pause(0.1-toc(second_time));
end
% Assign matrices to workspace
set(h.message_gui,'string','Done!');
assignin('base','snd_mat',snd_mat);
assignin('base','st_dev_mat',st_dev_mat);
assignin('base','dot_prod_mat',dot_prod_mat);
assignin('base','S',S);
figure;imagesc(snd_mat);colorbar;set(gca,'ydir','normal');

function thresh_choose(hObject,eventdata,h)
% Choose test to be either standard deviation or dot product
if get(h.thresh_choose,'value')
    set(h.thresh_choose,'string','Standard Deviation');
else
    set(h.thresh_choose,'string','Dot Product');
end
```

Enclosure Design:

Box (all measurements in mm):



Lid (all measurements in mm):



Intentionally Blank.

designsafe Report

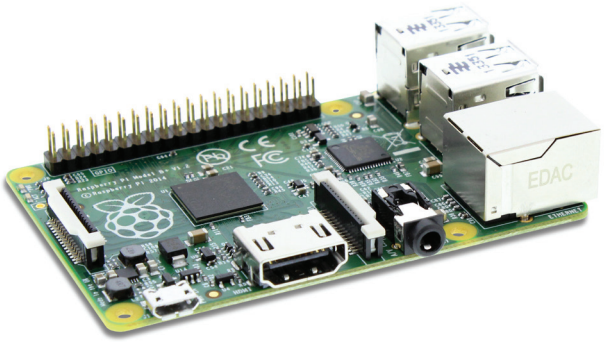
Application: A Commercial Nocturnal Asthma Monitor
Analyst Name(s):
Description:
Company:
Product Identifier:
Facility Location:
Assessment Type: Detailed
Limits:
Sources:
Risk Scoring System: ANSI B11.0 (TR3) Two Factor

Guide sentence: When doing [task], the [user] could be injured by the [hazard] due to the [failure mode].

Item Id	User / Task	Hazard / Failure Mode	Initial Assessment Severity Probability	Risk Level	Risk Reduction Methods /Comments	Final Assessment Severity Probability	Risk Level	Status / Responsible /Reference
1-1-1	operator normal operation	electrical / electronic : water / wet locations Spilled glass of water, etc.	Moderate Unlikely	Low	Warn against keeping water near the device	Moderate Unlikely	Low	TBD Operator
1-1-2	operator normal operation	slips / trips / falls : falling material / object Falls off of nightstand	Minor Unlikely	Negligible	Advise keeping the device in the middle of its stand	Minor Unlikely	Negligible	TBD Operator
1-1-3	operator normal operation	heat / temperature : radiant heat Device may generate significant heat	Minor Unlikely	Negligible	Ensure power consumption won't cause undue heating and quality check the device	Minor Remote	Negligible	On-going [Daily] Designers
1-2-1	operator clean up	electrical / electronic : improper wiring May cause a shock	Moderate Unlikely	Low	Quality-check the device	Moderate Unlikely	Low	On-going [Daily] Designers
1-2-2	operator clean up	electrical / electronic : water / wet locations Excessive liquid cleaners/water could damage the circuitry or cause a short	Moderate Likely	Medium	Provide a safety warning on the hazards of wetting the device	Moderate Unlikely	Low	TBD Operator
1-2-3	operator clean up	slips / trips / falls : trip Power cord on the ground	Minor Unlikely	Negligible	Safety warning about keeping cords out of foot traffic areas	Minor Unlikely	Negligible	TBD Operator
1-3-1	operator basic trouble shooting / problem solving	electrical / electronic : energized equipment / live parts Opening up and trouble shooting the device without unplugging it	Moderate Unlikely	Low	Warn against opening up the device and tampering with it / advise to unplug it before you do so	Moderate Remote	Negligible	TBD Operator

Item Id	User / Task	Hazard / Failure Mode	Initial Assessment Severity Probability	Risk Level	Risk Reduction Methods /Comments	Final Assessment Severity Probability	Risk Level	Status / Responsible /Reference
1-3-2	operator basic trouble shooting / problem solving	electrical / electronic : lack of grounding (earthing or neutral) Wiring mistake	Moderate Unlikely	Low	Quality-check the device	Moderate Unlikely	Low	On-going [Daily] Designers
1-3-3	operator basic trouble shooting / problem solving	electrical / electronic : shorts / arcing / sparking Wiring mistake, liquid, physical damage	Moderate Remote	Negligible	Quality-check the device, see above for water warning	Moderate Remote	Negligible	On-going [Daily] Designers
1-3-4	operator basic trouble shooting / problem solving	electrical / electronic : improper wiring Mistake in assembly	Moderate Unlikely	Low	Quality-check the device	Moderate Unlikely	Low	On-going [Daily] Designers
1-3-5	operator basic trouble shooting / problem solving	electrical / electronic : water / wet locations Water spilled nearby or used to clean	Minor Remote	Negligible	Provide a safety warning on the hazards of wetting the device	Minor Remote	Negligible	TBD Operator
1-3-6	operator basic trouble shooting / problem solving	electrical / electronic : overvoltage /overcurrent Electrical surge	Moderate Remote	Negligible	Quality-check the device, advice the use of a surge protector	Moderate Remote	Negligible	TBD Operator
1-3-7	operator basic trouble shooting / problem solving	slips / trips / falls : falling material / object Falls off of nightstand	Minor Unlikely	Negligible	Advise keeping the device in the middle of its stand	Minor Unlikely	Negligible	TBD Operator
2-1-1	passer by / non-user work next to / near machinery	slips / trips / falls : trip Power cord on the ground	Minor Remote	Negligible	Safety warning about keeping cords out of foot traffic areas	Minor Remote	Negligible	TBD Passerby
2-1-2	passer by / non-user work next to / near machinery	slips / trips / falls : falling material / object Falls off of nightstand	Minor Remote	Negligible	Advise keeping the device in the middle of its stand	Minor Remote	Negligible	TBD Passerby
2-1-3	passer by / non-user work next to / near machinery	heat / temperature : radiant heat Excessive power consumption	Minor Remote	Negligible	Ensure power consumption won't cause undue heating and quality check the device	Minor Remote	Negligible	On-going [Daily] Designers

Raspberry Pi



MODEL B+

Product Name	**Raspberry Pi Model B+**
Product Description	The Raspberry Pi Model B+ incorporates a number of enhancements and new features. Improved power consumption, increased connectivity and greater IO are among the improvements to this powerful, small and lightweight ARM based computer.

Specifications

Chip	Broadcom BCM2835 SoC
Core architecture	ARM11
CPU	700 MHz Low Power ARM1176JZFS Applications Processor
GPU	Dual Core VideoCore IV® Multimedia Co-Processor
	Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode
	Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	512MB SDRAM
Operating System	Boots from Micro SD card, running a version of the Linux operating system
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V, 2A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4)
	Composite RCA (PAL and NTSC)
Audio Output	3.5mm jack, HDMI
USB	4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip
	Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
JTAG	Not populated
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	SDIO

Microsoft®
LifeCam Studio™

Version Information	
Product Name	Microsoft® LifeCam Studio™
Product Version	Microsoft LifeCam Studio
Webcam Version	Microsoft LifeCam Studio
Product Dimensions	
Webcam Length	4.48 inches (114 millimeters)
Webcam Width	2.36 inches (60.0 millimeters)
Webcam Depth/Height	1.77 inches (45.0 millimeters)
Webcam Weight	4.52 ounces (128 grams)
Webcam Cable Length	72.0 inches +6/-0 inches (1829 millimeters +152/-0 millimeters)
Compatibility and Localization	
Interface	Hi-speed USB compatible with the USB 2.0 specification
Operating Systems[1]	• Microsoft Windows® 8.1, Windows 8, Windows RT 8.1, Windows RT 8, and Windows 7 • Macintosh OS X v10.7-10.9 • Android 3.2 and 4.2 [1]Advanced functionality not available with all devices and/or operating systems. See compatibility information at: www.microsoft.com/hardware/compatibility.
Top-line System Requirements	Requires a PC that meets the requirements for and has installed one of these operating systems: • Microsoft Windows 8.1, Windows 8, or Windows 7 • **VGA video calling:** • Intel Dual-Core 1.6 GHz or higher • 1 GB of RAM • **720p HD recording:** • Intel Dual-Core 3.0 GHz or higher • 2 GB of RAM • 1.5 GB hard drive space • Display adapter capable of 16-bit color depth or higher • 2 MB or higher video memory • Windows-compatible speakers or headphones • USB 2.0 You must accept License Terms for software download. Please download the latest available software version for your OS/Hardware combination. Internet access may be required for certain features. Local and/or long-distance telephone toll charges may apply. Software download required for full functionality of all features. Internet functions (post to Windows Live™ Spaces, send in e-mail, video calls), also require: Internet Explorer® 6/7 browser software required for installation; 25 MB hard drive space typically required (users can maintain other default Web browsers after installation) The Microsoft LifeCam Studio has basic Video & Audio Functionality with Windows Live Messenger, AOL® Instant Messenger™, Yahoo!® Messenger, Skype, and Microsoft Office Communicator
Compatibility Logos	• Compatible with Microsoft Windows 8 and Windows RT • Hi-Speed USB Logo
Software Localization	Microsoft LifeCam software version 3.5 may be installed in Simplified Chinese, Traditional Chinese, English, French, German, Italian, Japanese, Korean, Brazilian Portuguese, Iberian Portuguese, or Spanish. If available, standard setup will install the software in the default OS language. Otherwise, the English language version will be installed.
Windows Live™ Integration Features	
Video Conversation Feature	Windows Live call button delivers one touch access to video conversation
Call Button Life	10,000 actuations
Webcam Controls & Effects	LifeCam Dashboard provides access to animated video special effect features and webcam controls
Windows Live Integration Features	• Windows Live Photo Gallery integration - Take a photo with LifeCam Software, then with one click open Photo Gallery to edit, tag and share it online • Windows Live Movie Maker integration - Record a video with LifeCam Software and start a movie project on Movie Maker with just one click to then upload it to your favorite networking site
Imaging Features	
Sensor	CMOS sensor technology
Resolution	• Sensor Resolution: 1920 X 1080 • Still Image: 5 megapixel (2560 x 2048 pixel, interpolated) photos*
Field of View	75° diagonal field of view
Imaging Features	• Automatic face tracking** • Digital pan, digital tilt, and 3x digital zoom** • Auto focus from 0.1m to ≥ 10m • Automatic image adjustment with manual override • Up to 30 frames per second
Product Feature Performance	
Audio Features	Integrated omni-directional super wideband microphone
Frequency Response	100 Hz – 18 kHz
Mounting Features	• Desktop and CRT universal attachment base • Notebook and LCD universal attachment base • Tripod universal attachment base
Storage Temperature & Humidity	-40 °F (-40 °C) to 140 °F (60 °C) at <5% to 65% relative humidity (non-condensing)
Operating Temperature & Humidity	32° F (0° C) to 104° F (40° C) at <5% to 80% relative humidity (non-condensing)

Certification Information		
Country of Manufacture	People's Republic of China (PRC)	
ISO 9001 Qualified Manufacturer	Yes	
ISO 14001 Qualified Manufacturer	Yes	
Restriction on Hazardous Substances	This device complies with all applicable worldwide regulations and restrictions including, but not limited to: EU directive 2002/95/EC on the Restriction of the Use of Certain Hazardous Substances in Electrical and Electronic Equipment and EU Registration Evaluation and Authorization of Chemicals (REACH) regulation regarding Substances of Very High Concern.	
FCC ID	This device complies with Part 15 of the FCC Rules and Industry Canada ICES-003. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation. Tested to comply with FCC standards. For home and office use. Model number: 1425, LifeCam Studio.	
Agency and Regulatory Marks	• ACMA Declaration of Conformity (Australia and New Zealand) • ICES-003 report on file (Canada) • EIP Pollution Control Mark, EPUP (China) • WEEE (European Union) • CE Declaration of Conformity (European Union) • VCCI Certificate (Japan) • MIC Certificate (Korea)	• GOST Certificate (Russia) • CITC Letter (Kingdom of Saudi Arabia) • UkrSEPRO Certificate (Ukraine) • FCC Declaration of Conformity (USA) • UL and cUL Notice of Approval (USA and Canada) • CB Scheme Certificate (International)
Windows Certification Kit (WCK)	ID: 1609318 (32-bit) and 1604420 (64-bit) Microsoft Windows 8.1	

snowflake

Portable Professional USB mic

Welcome to the Small Chill.

Thank you for your purchase of the Snowflake, Blue's cool USB mic designed for use on the go. Whether you're recording your newest podcast, talking business on the web, or just narrating your latest great family movie, the Snowflake can capture it with amazing clarity and depth that's head and shoulders above any comparable portable USB on the market.

The Snowflake is a high-quality condenser microphone designed to give you professional results with very little effort. Simply connect it to your computer's USB port, follow the prompts and you'll have high-fidelity sound in no time! The Snowflake has its own digital converter and specially-designed preamplifier, both optimized to work in conjunction with Blue's precision-tuned capsule to make sure that you always get the best sound possible into your computer.

Suggested applications

The Snowflake works on both PC and Mac with no complicated drivers to install: just connect it to your computer's USB port, follow the prompts and you'll have high-fidelity sound in no time. It's perfect for using with instant messaging, video conferencing, and social networking programs like iChat, Skype, ooVoo, Google Talk, Windows Live Messenger, AOL Instant Messenger, Yahoo Messenger, Vonage and more. Or use it for voice recognition, dictation, field recording, lecture recording, podcasting, or narration for slideshows and PowerPoint presentations. The sky's the limit.

Where do I position it?

The Snowflake's unique design allows you to place it on a desk or flat surface near your computer, or mounted to the screen of most laptops. To open your Snowflake, gently grasp the top and bottom halves of the Snowflake mounting box and slide them away from each other **(figure 1)**. You'll find the USB cable snugly inside. The metal base of the Snowflake can then be positioned on any flat surface **(figure 2)**, or hung over the back of most laptops **(figure 3)**.

NOTE: Forceful positioning of the head can result in damage not covered by the warranty.

When not in use, just disconnect the USB cable and store it in the mounting box, sliding both halves together to close. When traveling with the Snowflake, gently rotate the capsule head so that the metal grille faces down and into the box **(figure 4)**. This will help protect the precision capsule from damage.

What next?

Simply connect the Snowflake to the USB port on your Windows or Macintosh computer and follow the setup instructions below. The Snowflake head pivots back and forth and rotates side-to-side for optimal positioning. Experiment with the positioning of the Snowflake to find the sound that best suits your application: normal conversational volumes are best captured at 1-2 feet (0.3-0.6M), with the Snowflake head pointing directly at the sound source.

Snowflake Setup

Macintosh Setup Procedure:

1. For OSX users: in the Apple menu, open *System Preferences*.
2. Double-click *Sound* preference file.
3. Click on the *Input* tab.
4. Double click *Blue USB Snowflake Mic* under choose a device for *sound input* dialog box.
5. Set input volume to the appropriate level.
6. Exit *System Preferences*.

Windows Setup Procedure (Win98/XP/NT):

1. Under *Start Menu*, open the *Sounds and Audio Devices* in the control panel.
2. Select *Audio* tab; insure that *Blue Snowflake USB Mic* is selected as default device.
3. Click on *Volume*; select appropriate volume level.
4. Exit control panel.

Windows Setup Procedure (Vista):

1. Under *Start Menu*, open the *Control Panel*, then select *Sound*.
2. Select *Recording* tab; insure that *Blue Snowflake* is selected as *Working* with check mark next to icon (disable alternate mic if necessary).
3. Click on *Properties*; select the *Levels* tab and set your input level, click *Apply*, then *OK*.
4. Exit control panel.

Technical Specifications

Transducer Type:	Condenser, Pressure Gradient
Polar Pattern:	Cardioid
Sample Rate/Word Length:	44.1 kHz/16 bit
Frequency Response:	35Hz – 20kHz
Maximum SPL (THD 0.5%):	120 dB SPL

Blue Microphones
5706 Corsa Ave., Suite 102, Westlake Village, CA 91362
www.bluemic.com

2-Year Limited Warranty.
Designed in USA. Made in China.