2022

# A Measurement System for Detection of Intestinal Motility in Neonates by Monitoring Slow Wave Activity

Garett Goodale
*University of Central Florida*

# A MEASUREMENT SYSTEM FOR DETECTION OF INTESTINAL MOTILITY IN NEONATES BY MONITORING SLOW WAVE ACTIVITY

BY

GARETT GOODALE
B.S. UNIVERSITY OF CENTRAL FLORIDA, 2020

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2022

# ABSTRACT

Similar to how electrocardiographic waves are the pace making signals of the heart, slow waves are the pace making signals of the intestines. Slow waves are electrical signals in the intestines that determine the speed at which food can move through the intestine ensuring proper digestion and uptake of nutrients. It has been shown that slow waves can be measured in adults using non-invasive, surface electrodes. However, no study has investigated the measurements of slow waves in neonates, specifically pre-term neonates. Around 7% of pre-term neonates suffer from necrotizing enterocolitis (NEC) which is a condition that causes damage to the intestinal tract and often death of intestinal tissue. NEC affects around 9,000 neonates each year with a survival rate estimated to be between 60%-80%. Currently, there are no non-invasive, early-stage indicators of NEC. This pilot study aims to create a non-invasive measurement setup to measure and characterize slow wave activity in neonates.

# ACKNOWLEDGMENTS

Thank you to Dr. Reza for all of his support and mentorship throughout my time at UCF, I would not be here without him. Thank you to Dr. Sreekanth Viswanathan at Nemours Children's hospital for suggesting this research project and for pushing the project forward at Nemours. Thank you to Dr. Javier Garcia-Casado at Universitat Politècnica de València in Spain for his work in the field of slow wave measurements and for his help in advising on the measurement setup and data analysis. I also have much appreciation for my lab mates at the Dynamic Microsystems Lab for their intriguing conversations and support in this project.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ACRONYMS

| | |
|---|---|
| AAMI | Association for the Advancement of Medical Instrumentation |
| AC | Alternating Current |
| ADC | Analog to Digital Converter |
| ANSI | American National Standards Institute |
| CMRR | Common Mode Rejection Ratio |
| CPM | Cycles Per Minute |
| CPU | Central Processing Unit |
| DAQ | Data Acquisition Unit |
| DC | Direct Current |
| EEnG | Electroenterogram |
| EGG | Electrogastrogram |
| EKG | Electrocardiogram |
| EMG | Electromyography |
| EMI | Electromagnetic Interference |
| EOG | Electrooculogram |
| FDA | Food and Drug Admistration |
| FFT | Fast Fourier Transform |
| GIL | Global Interpreter Lock |
| GUI | Graphical User Interface |
| IEC | International Electrotechnical Commission |

| | |
|---|---|
| IO | Input Output |
| IPS | Isolated Power Supply |
| IRB | Internal Review Board |
| NEC | Necrotizing Enterocolitis |
| NI | National Instruments |
| NICU | Neonatal Intensive Care Unit |
| OS | Operating System |
| PCB | Printed Circuit Board |
| PET | Polyethylene Terephthalate |
| RAM | Random Access Memory |
| USB | Universal Serial Bus |

# CHAPTER 1: INTRODUCTION

The use of surface electrodes to measure internal electrophysiological signals has been widely applied to common medical measurements like the electrocardiogram (EKG) and electroencephalogram (EEG). Surface electrode measurements are also used for some less common medical measurements like electrooculogram (EOG), electromyography (EMG) and electrogastrogram (EGG). Another, though less widely known, surface electrode measurement is the Electroenterogram (EEnG) which has been well understood in research but is not widely used in the medical field. EEnG can be measured invasively by surgically placing electrodes directly on the small intestine. This leads to high quality signals which has been used to evaluate motility typically in gastroparetic patients [1]. Of course, for most applications an invasive approach is not viable so surface electrodes must be used instead.

Surface EEnG is more difficult to measure than internal EEnG due to high electrode impedance as well as interference from EKG and respiratory signals. As a result, time domain analysis of EEnG waveforms typically has limited use for direct viewing and spectral content must be observed instead. Studies have found in humans [1] and in beagles [2] that the spectra observed by surface electrodes is correlated to the spectra observed by internal electrodes placed directly on the small intestine. In one study they found the correlation to be between 0.5 and 0.7 depending on the placement of the internal and external electrodes. In this study, surface electrodes will be used to measure slow waves in neonates and the frequency spectra will be used to determine the presence of slow waves.

# CHAPTER 2: MEASUREMENT SETUP

The measurement shown in Figure 1 is meant to acquire EEnG, EKG and respiratory. EKG and respiratory are taken in addition to EEnG since the artifacts of both signals tend to interference with EEnG and must be accounted for. The electrode measurements (EKG and EEnG) all run though Grass P511kk AC pre-amplifiers which provide variable low and high pass filtering as well as a notch filter at 60 Hz and amplification. Two EEnG measurements are taken both to investigate better electrode placement and to provide reduce the chances of a fluke measurement (both electrodes should measure similar frequencies). EKG is taken in the standard fashion across the chest and is typically taken in addition the EKG provided at the hospital due to incompatibilities between measurements systems. Respiratory is taken via an accelerometer which also allows for the monitoring and removal of motion artifacts in post processing. The accelerometer provides X, Y and Z components which are all filtered and amplified with a custom front end discussed further in the subsections. After all analog signals are filtered and amplified, they are fed into an NI-6008 data acquisition unit (DAQ) which provides the analog to digital conversion. A computer then pulls the data from the DAQ and displays it to the user via custom software along with the fast Fourier transform (FFT) on the EEnG for identification of frequency components. The software stores all data to file for post processing after the measurement is taken.

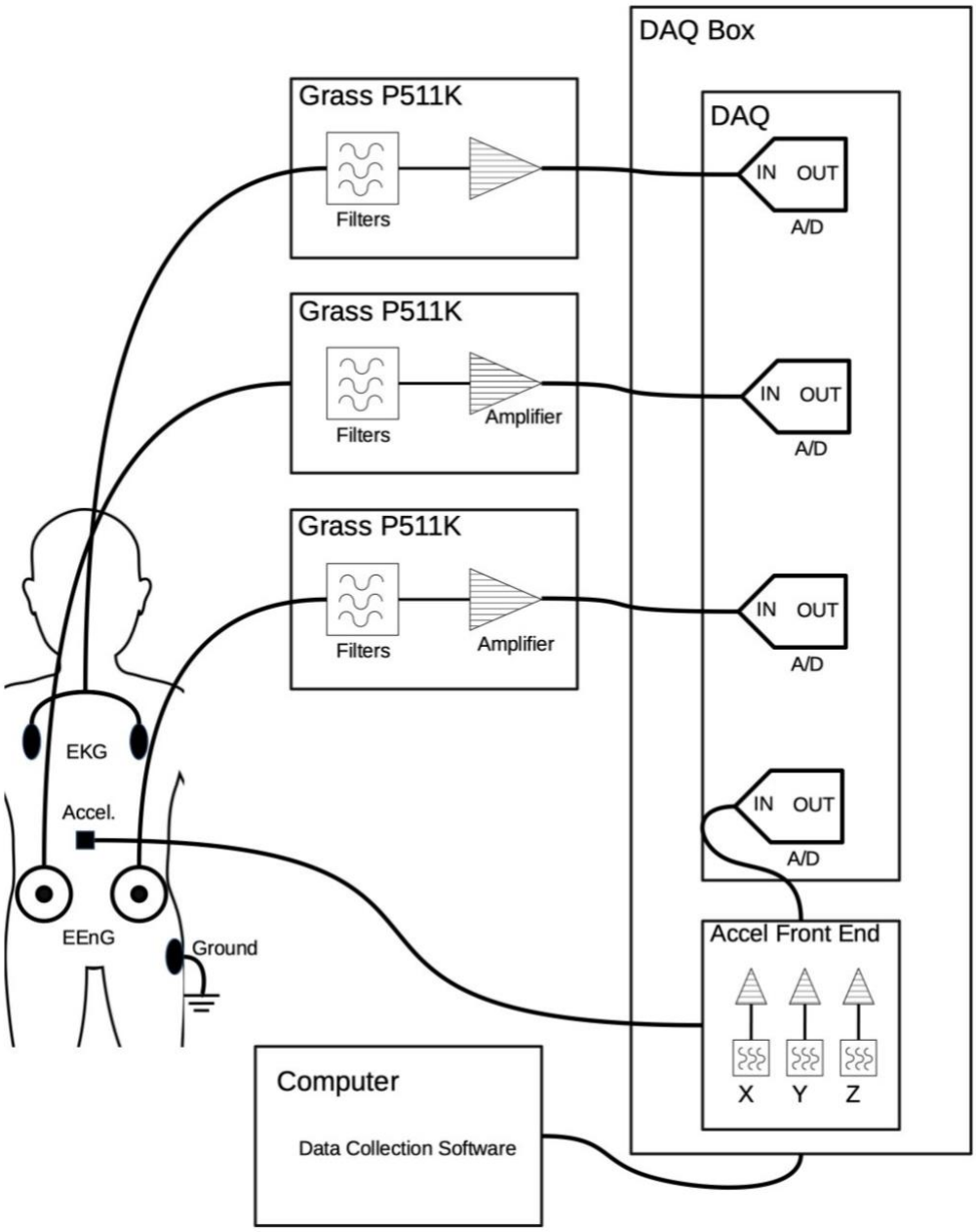


Figure 1. Simplified diagram of the complete measurement setup

## 2.1 Filters and Amplifiers

Grass P511k AC preamplifiers were used because they are IEC-60601 compliant which is an international human rating standard for research instruments. Currently, the only active manufacturer with IEC-60601 rating is AD-Instruments. Grass Instruments was purchased was purchased by Natus from Astro-Med in 2012 at which point the P511K amplifier was discontinued, and the Grass website was taken down. As such there is limited information available online related to P511ks amplifiers, but a manual can be obtained by calling Natus.


Figure 2. Four Grass P511K amplifiers stacked atop a IPS 115 isolated power supply, all fastened to a 6U rack

## 2.2 Electrodes

Electrodes for measuring EKG neonates are plentiful, so a snap based connector was chosen to maintain compatibility with the EEnG electrodes (also snap based). The EKG electrodes were also chosen to be a wet gel-based electrode since it allows for lower impedance and the short duration of the study ensured the electrodes would not dry out. Neuroline 720 electrodes were chosen due to their small size and gentle adhesive. Three of these electrodes were used in each measurement, the third was used as a ground both for EKG and EEnG.

Figure 3. Neuroline 720 gel-based electrode, top (left) and bottom (right)

A previous study [3]found that the optimum electrode for measuring slow wave activity is a concentric electrode with a 30 mm outer diameter and a 10 mm inner diameter. The study found that concentric electrodes reduced interference due to EKG and respiratory signals when compared

to two bipolar electrodes. The 30 mm outer diameter electrode was found to be the best tradeoff between good signal integrity and spatial resolution. Commercial electrodes (CODE5000S0) were found with a 30 mm outer diameter and 10 mm inner diameter from Spes Medica in Italy. They are flexible and composed of a silver chloride compound (as found in the study). The electrodes have received FDA approval and as a result made IRB approval easier. The adhesive was found to be strong enough to allow for a good surface impedance without being too strong for neonate's delicate skin.

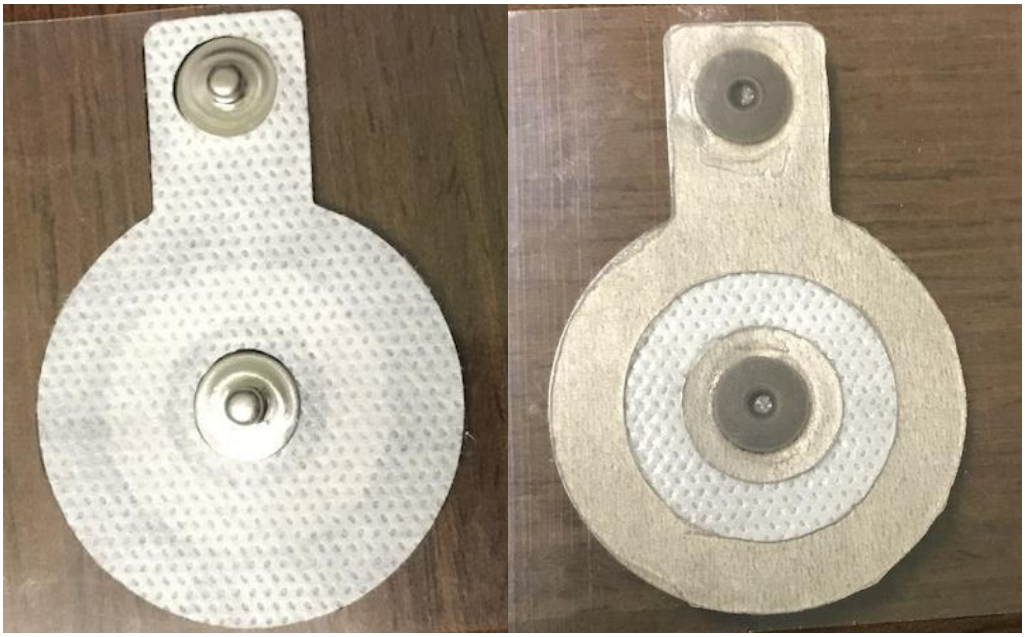

Figure 4. Spes Medica bipolar concentric electrode top (left) and bottom (right)

## 2.3 Cabling

Figure 5. Medical cable (left) and the custom cable (right)

Due to the fact that the measured signals are on the order of 10 $\mu V_{p-p}$, proper cabling is essential to avoiding noise overriding the signals. In an initial setup, a custom cable was made with simple PET outer coating, inner braided shield and highly flexible wires with silicon insulation as is shown in Figure 6. The cable was very flexible, which was the objective in order to avoid significant cable forces pushing or pulling on the electrodes. The outershield helped reduce EMI which primarily originated from 60 Hz noise from the power outlets since higher frequencies are filtered. Unfortunately, the interactions between the outer housing and conductor housing caused triboelectric charging and created charge buildup. This is because the inner conductor insulation

was made from a highly electronegative material: silicon. The creation of these charges due to motion on the cable induced significant currents in the conductors which flowed into the large impedance of the amplifiers and generated voltages in the 10 mV range for significant cable movements. These motion artifacts on the cable due to the triboelectric effect were enough to completely obscure the signal of interest which led to further cable investigation

Figure 6. Original cable used with a simple PET outer coating, inner braided shield and highly flexible wires with silicon insulation

Medical cables strive to reduce the effect of the triboelectric effect and achieve this by adding an additional inner housing that conforms to the inner conductors and is also relatively conductive at around 100 Ohms/inch. This inner housing helps pull charge from the insulation of the inner conductors when it is created to prevent charge buildup which can induce currents. This reduces the motion artifacts due to triboelectric effect by several orders of magnitude to around 10-20 $\mu V_{p-p}$ for significant cable movements. It is worth noting that this is still on the same order of magnitude

as the signal, so care must be taken to reduce large cable movements, while small movements may not significantly affect the signal.

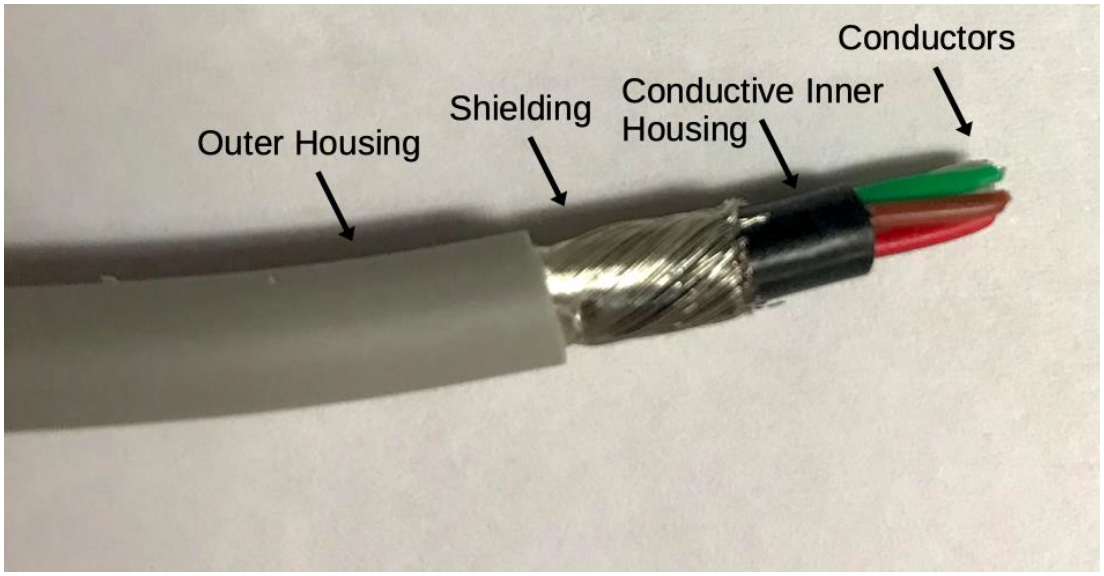


Figure 7. Breakdown of the low noise medical cabling used

There is a medical standard, AAMI ANSI EC53, which defines the maximum amount of triboelectric noise to be 50 μV "peak to trough" when a weight equal to 40x the weight of 1 ft of wire is dropped on the center of a 7 foot wire with 5 feet of play. For the purposes of this study, it was not necessary to characterize the exact triboelectric noise of the setup. However, in order to compare the performance of the custom-made cables without the internal conductive housing to the true medical cables, a similar setup was performed as shown in Figure 8.

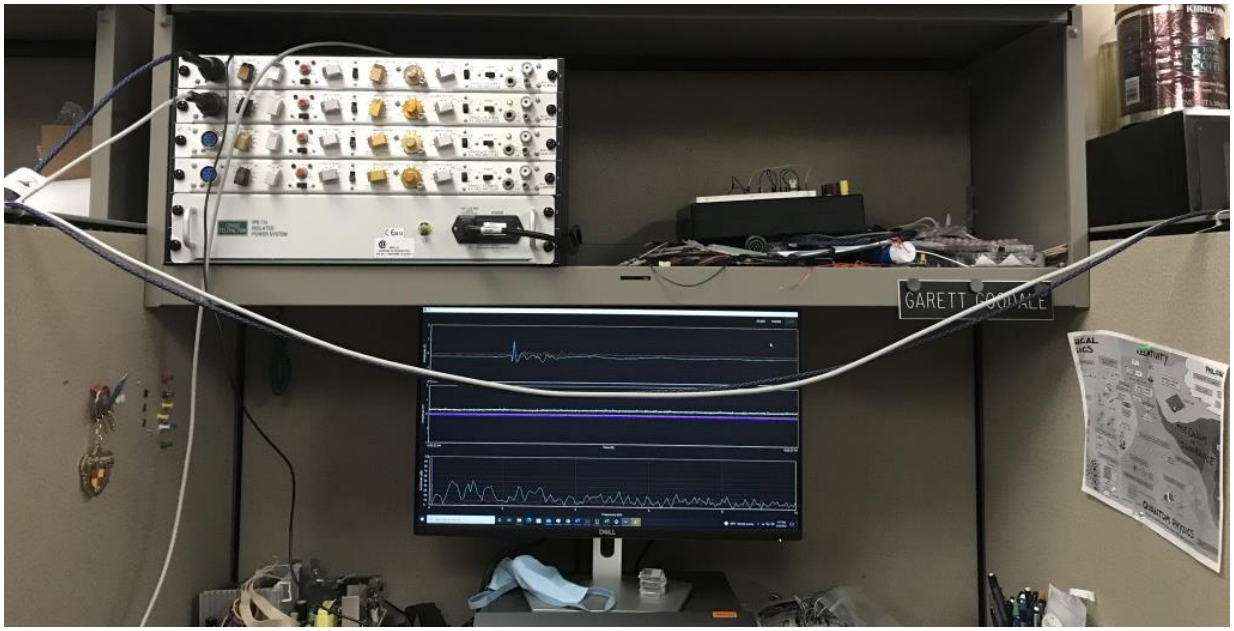

Figure 8. Medical and custom cable hung across the cubicle for drop testing

The cables were secured to either side of a cubicle which was about 3.5 ft apart with 4 ft of cable between. The cables were then dropped simultaneously with a 100 g weight attached to each. As can be seen in Figure 9, the custom cable experienced about a 3.805 mV peak to trough jump while the medical cable experienced about a 0.739 mV peak to trough jump. These jumps are not as precise as a true ANSI measurement since the ends of the cable were not carefully fixed and the weight of each cable was not considered. However, this shows how much cable structure and materials matter is reducing triboelectric noise.

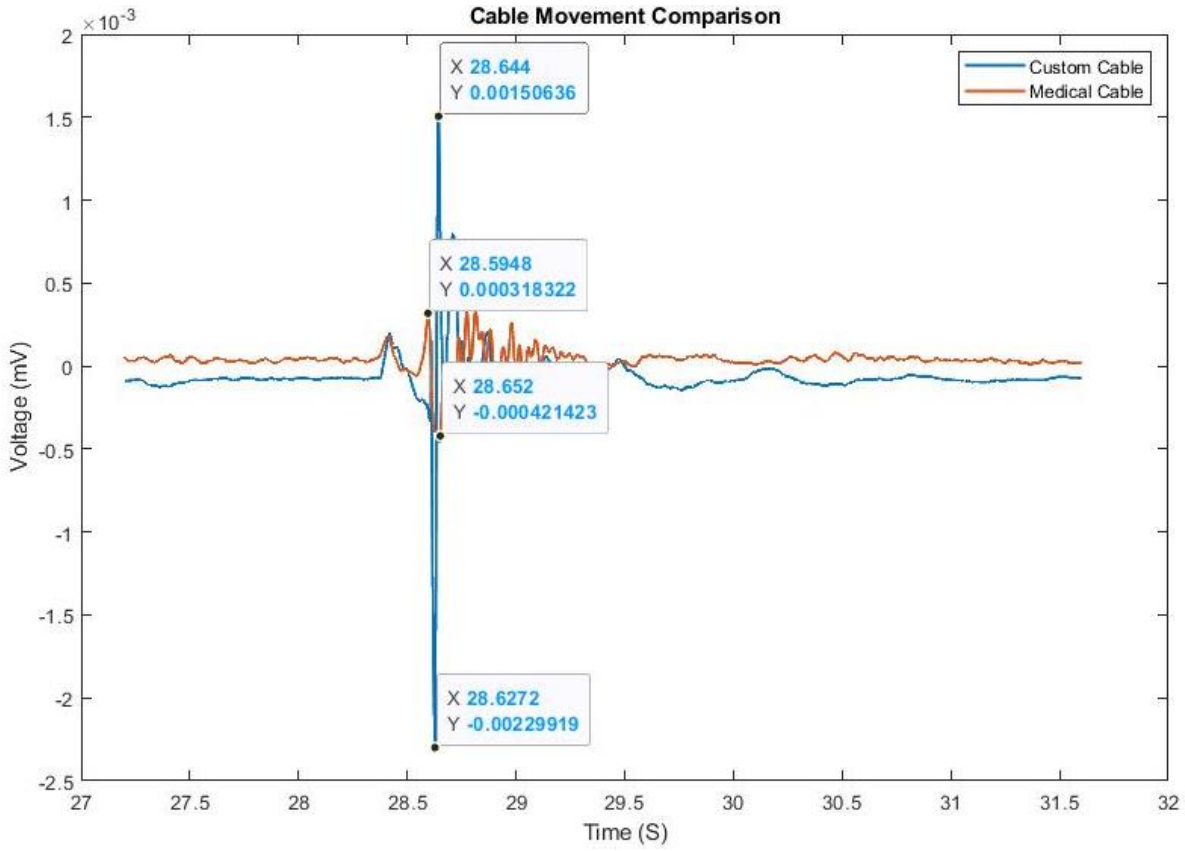


Figure 9. Comparison of the triboelectric noise due to a cable drop between a custom-made cable (blue) and a true medical cable (orange)

The overall takeaway from the cable discussion is to ensure good medical cables are used with low triboelectric noise which drastically reduces the noise due to motion artifact in the cable.

## 2.4 DAQ Box

Figure 10. DAQ box with the accelerometer and USB cable connected

The DAQ box is simply a box that houses the DAQ as well as the amplifiers and filters for the accelerometer measurement. The output of the top Grass P511k amplifier (EEnG 1 or the left

EEnG) is connected to channel 0, the second Grass amplifier (EEnG 2 or the right EEnG) is connected to channel 1 and the third Grass amplifier (EKG) is connected to channel 2. Channels 3, 4 and 5 take the output from the amplifier and filter PCBs which are connected to the accelerometer. As can be seen from the wiring and composition of the box (plastic), interference is not a major concern since the accelerometer inputs are biased and fairly large (mV range) and the inputs of the biological signals have already been filtered and amplified. This setup could be improved by using PCB edge mount connectors and then interfacing everything via PCB traces which could be better isolated. A metal box would also act as a faraday cage reducing interference. These additional measures weren't taken due to time and significant interreference was not created from this part of the measurement chain.

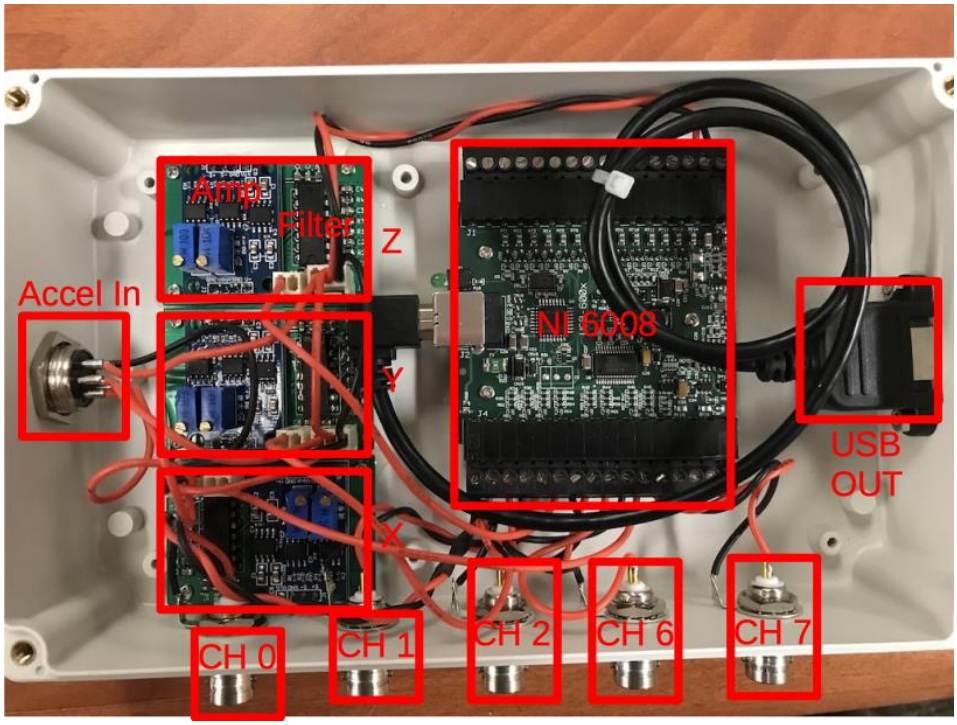


Figure 11. Labeled diagram of the DAQ box which includes the DAQ and the accelerometer front end hardware

## 2.5 Accelerometer and Frontend

An accelerometer was desired in order to detect when the patient moved in order to be able to mark that data as contaminated when performing the post processing. In addition to detecting motion artifacts, the accelerometer could also be used to detect the respiratory rate. If an accelerometer with sufficiently low frequency response and sensitivity was selected, and paired with appropriate filtering and amplification, respiratory rate could be detected by the rising and falling of the diaphragm when placed appropriately. Previous studies [3] used an ADXL 335 triaxial accelerometer from Analog instruments to measure movements and their results showed that accurate respiration was also picked up by the accelerometer though they don't mention what

amplification and filtering was applied to the accelerometer signals. Looking online for approximate breathing rates allowed for proper tuning of the filters while in person testing was used to find the necessary gain as is shown in the following sections.

An inexpensive ADXL335 evaluation module was purchased which includes the ADXL335 as well as the necessary decoupling capacitors and a low drop out linear regulator which regulates an input voltage of ~3.3 V – 9 V down to the 3.3 V required for operation while also providing reverse polarity protection. Due to the fact that the accelerometer needs to be used in a hospital environment, it was sealed and connected to a medical grade shielded cable as shown in Figure 12.

Figure 12. ADXL335 accelerometer evaluation module (left) and ADXL335 packaged and connected to a shielded cable (right)

### 2.5.1 Accelerometer Filter Design

Adults tend to have a breathing rate around 12 - 20 breaths/minute [4] which corresponds to 0.1 – 0.33 Hz, while neonates tend to have a breathing rate that is much higher, 45 – 60 breaths/minute or 0.66 – 1 Hz. As such the filtering function should be able to accommodate all breathing frequency ranges while rejecting DC and higher frequencies. To summarize, a filter is desired with a pass band from 0.1 – 2 Hz with a really flat pass band and reasonably good out of band rejection. This filter can be designed using equations, but with an abundance of online calculators it is not necessary. Analog Instruments offers a nice online filter design tool which can be found from the following link (https://tools.analog.com/en/filterwizard/). If the values in Table 1 are used as the input for the tool, it will generate the component values needed for both a Bessel filter and Butterworth filter. Those component values are then adjusted to match the values that are available in the resistor and capacitor kits available in the lab.

Table 1. Filter design parameters using the Analog Instruments filter design tool

| Gain | 0 dB |
|---|---|
| Passband (-3 dB) | 2 Hz |
| Stopband (-60 dB) | 1 KHz |
| Center Frequency | 0.5 Hz |

The circuits were then simulated in LTSpice simulation software as shown below, in order to determine the filter characteristics. As is well known in filter design, the Butterworth filter offers

a sharper roll off rate than the Bessel filter while sacrificing phase linearity. This is shown in Figure 14 which compares the simulated frequency response of the Butterworth and Bessel filters.

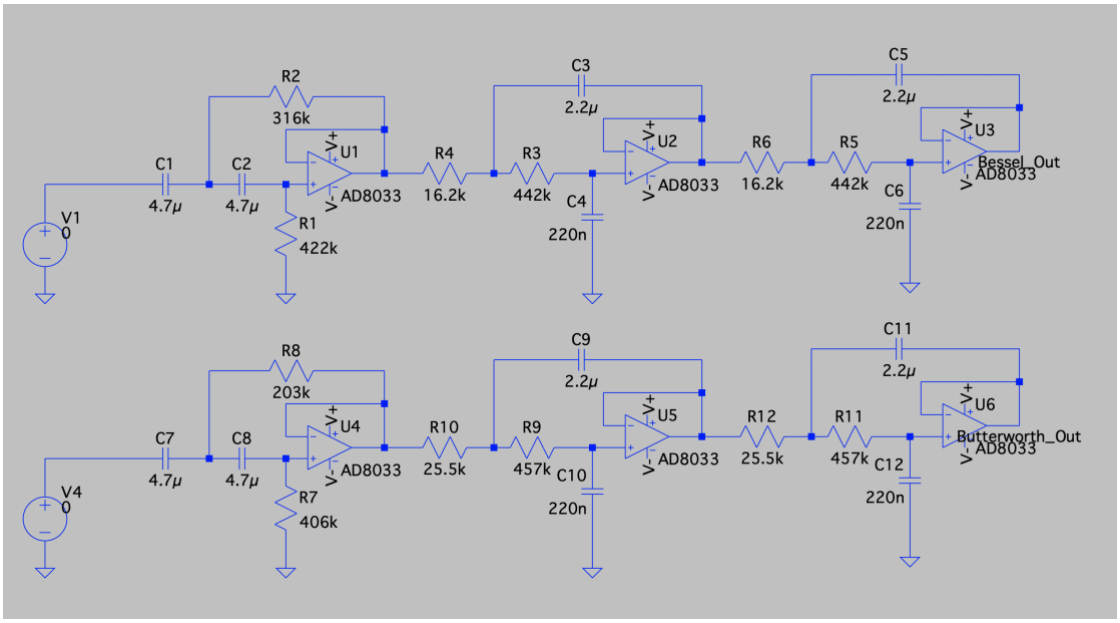


Figure 13. Schematic and values of the 6th order Bessel filter (top) and 6th order Butterworth filter (bottom)

As shown in Figure 13, the first stage of the filter is a second order high pass filter and the next two stages are second order lower pass filters. Since there are two low pass stages, the high side of the bandpass filter has a sharper roll off rate than the low side. This was done to ensure that the filter showed strong rejection at high frequencies like 60 Hz while it was not so critical to reject lower frequencies that were below the 0.1 Hz cutoff. As is shown in the left side of Figure 14, the frequency response clearly shows the non-symmetry of the filter with respect to the roll off rate on the high and low side. As previously mentioned, the Butterworth filter has a slightly better role off

rate than the Bessel filter. However, since the order of the filter is high enough, this additional roll off is not needed and the slightly better phase performance of the Bessel filter is chosen in order to reduce distortion.

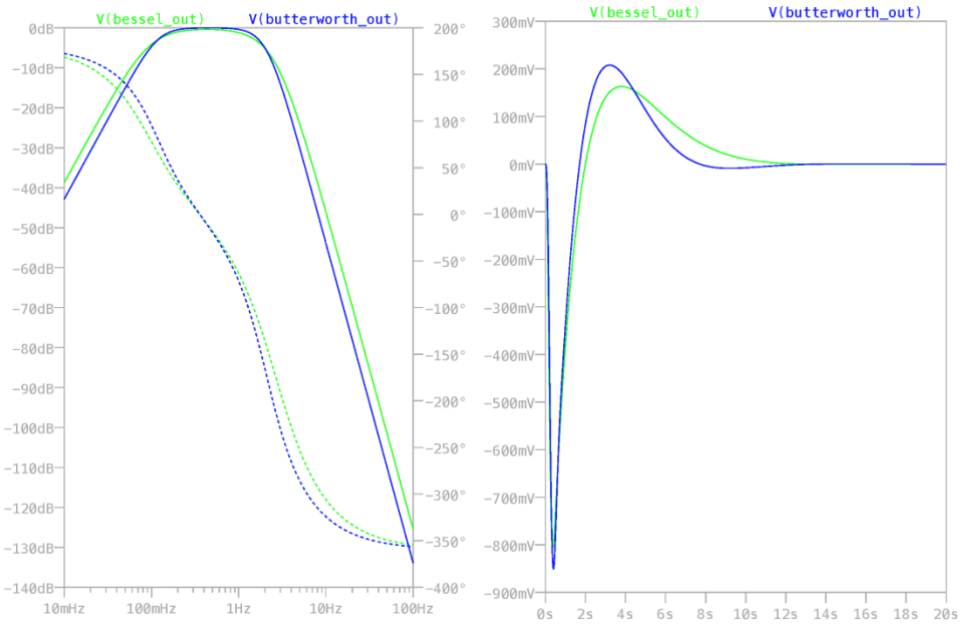


Figure 14. Frequency response (left) and step response (right) of the Bessel filter (green) and Butterworth filter (blue)

The right side of Figure 14 shows the filter response in the time domain to an infinite impulse. This allows the time constant of the filters to be determined. Both filters stabilize about 12 s after an impulse is applied so neither filter choice greatly improves the time constant performance. A smaller time constant is desired because large movements create a sharp voltage spike and render the circuit useless until the filter unwinds.

### 2.5.2 Accelerometer Amplifier Selection

The ADXL335 has acceleration sensitivity of approximately 0.3 V/g (g being 9.8 $m/s^2$) in all 3 axis with the Z axis being the most sensitive. Testing of the accelerometer placed on the abdomen showed that breathing creates around a 5-10 mV signal which will require an amplifier gain of approximately 100 to put the signal into the range of the ADC (-1 V to 1 V). A low frequency instrumentation amplifier is the ideal for this application since they offer high gain, linearity, and common mode rejection. The AD620 from analog instruments is a popular choice for this application and has been used in other studies for many biometric signals like EEG [5] and EKG [6]. The AD620 is also available in many evaluation modules which made getting a simple setup up and running quite easy.

The AD620 evaluation module is show in Figure 15 along with each of its features. The evaluation module includes a HT7660 which is operated in the inverting charge pump configuration to provide the inverse of the input voltage. This allows the AD620 to operate from -5 V – +5 V and is also used to power the op-amps for the filters shown in the previous section. The module also uses an LM358 op-amp in order to provide DC offset which allows for tuning to get the output exactly at 0 V even if the input has some DC drift. The evaluation module also uses a potentiometer as the gain resistor for the AD620 which allows for variable gain and a potentiometer to set the DC offset. The board also allows for the AD620 to be operated in differential mode with a S+ and S- input, but S- is tied to ground since the filter implementation is single ended.

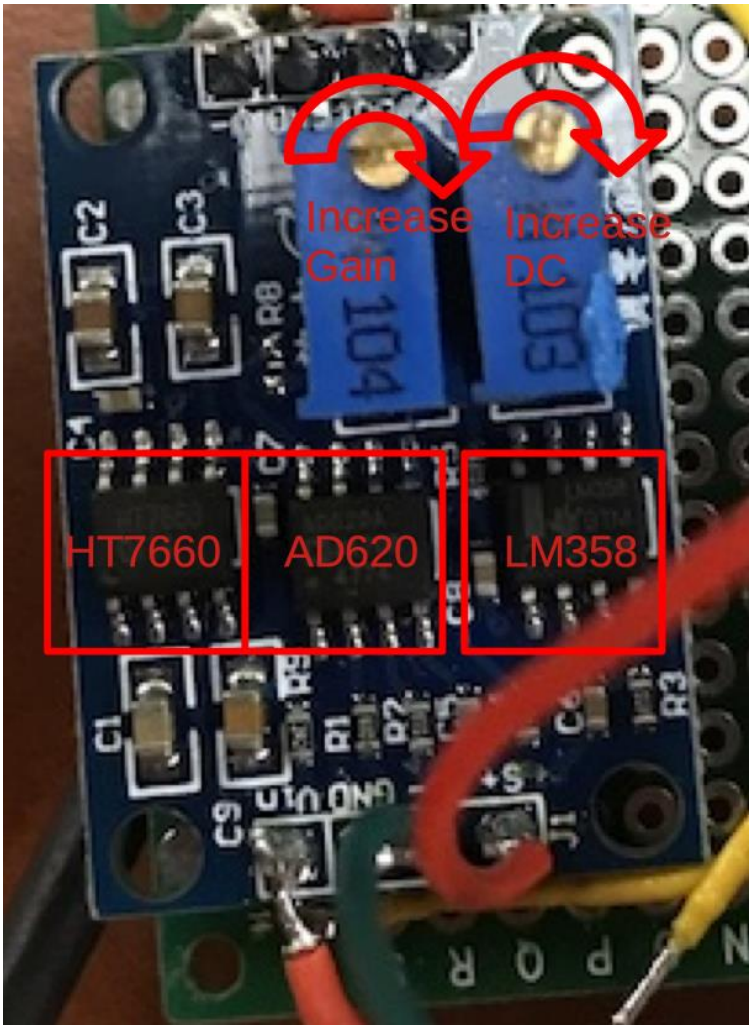


Figure 15. Labeled AD620 evaluation board

The evaluation board works well for this application and allowed for faster iteration, but it comes with one small drawback. The HT7660 is obviously a switched mode device, and it operates at 10 kHz. The ripple shown on the output is very small, typically around 20 – 30 mV, but it still has the potential to show up on the output. The AD620 has excellent CMRR, around 100 dB, which means that the effect of the ripple should be on the order of 0.04 – 0.06 mV which is quite small compared

with typical output voltages of 1 V – 2 V. Nevertheless, if this board were redesigned it would likely to be wise to go with a linear based inverter in order to ensure no noise on the output.

## 2.6 Software

The purpose of the software is to pull data from the DAQ, display all channels in a manner that is easy to understand, display the frequency spectrum of EEnG and to reliably save all data to file. This isn't a very significant task and can be completed in an environment such as National Instruments (NI) LabVIEW. LabVIEW wasn't chosen for this application due to the cost of the software, the undesirable appearance and lack of control over the data acquisition process. Instead, the acquisition software was written in Python due to is broad acceptance as well as the availability of modules to perform data acquisition from the DAQ (nidaqmx) and modules to create a nice frontend user interface (kivy). The code is available on GitHub (https://github.com/gmgoodale/Bioamp) and is also included in the appendix.

### 2.6.1 Data Acquisition

NI provides a python module called nidaqmx which allows for both discrete and continuous analog and digital acquisition from NI products. The module places the c API in a wrapper, so the NI-DAQmx software needs to be installed for python to interface with the c drivers. NI-DAQmx is available for Linux and Windows, but only a basic version is available for Mac which doesn't support most NI products. As such, this code is compatible with Windows and Linux, but not Mac.

Data acquisition is performed in a continuous fashion using the StreamReader functionality provided in the nidaqmx module. That means that the DAQ continues to push data into its IO buffer regardless of the IO read rate and without the computer requesting it. If the computer is unable to keep up with the output from the DAQ then the internal buffer on the DAQ overflows and causes an error. In order to ensure that the computer reads from the DAQ promptly, a dedicated process was used for the purpose to push the data to internal memory. This needs to be a dedicated process and not a thread in Python because Python only runs one thread at a time, so it is not truly parallel. This is due to the global interpreter lock (GIL) which simply doesn't support concurrent multithreading.

Multithreading does still have some benefits in Python though because when one thread isn't actively using CPU Python will switch to another waiting thread which still improves throughput compared to a purely sequential program. For that reason, the main thread which is responsible for updating the GUI is kept separate from the DAQ thread which is responsible for reading in the data from the DAQ process. Moving data between python process is limited to certain multiprocessor data types which is why a regular queue is used in this part of the code whereas a double ended queue (deque) is used for buffering the data in the main process.

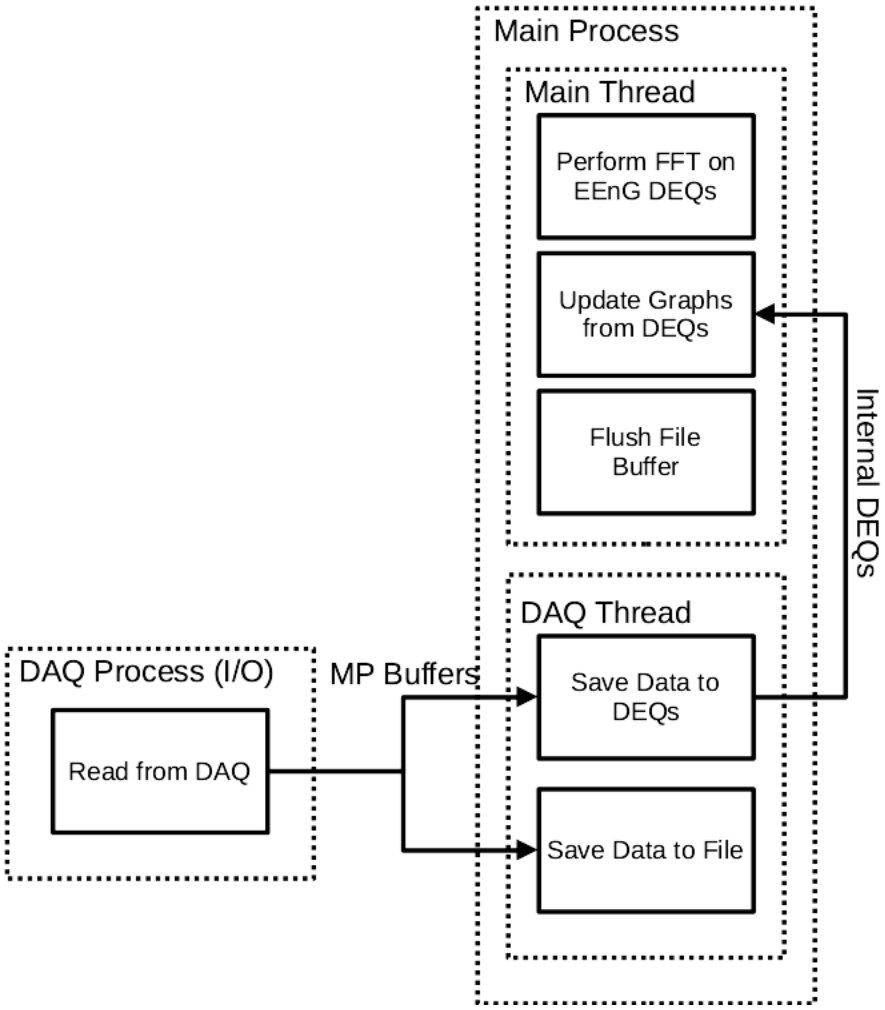


Figure 16. Simplified diagram of the process, threads and tasks

### 2.6.2 GUI

The GUI is responsible for displaying the time domain and frequency domain waveforms as well as gathering the patient number and starting and stopping the recording. Much more additional functionality could be added in the future to allow for things like modifying the bounds on the graphs (both X and Y) as well as changing the sampling rate, file saving location, or even

controlling the gain and filter settings if digital enabled amplifiers were used. However, for the time being the GUI was kept very simple to ensure reliable functionality. The GUI was written using a GUI module for Python called kivy. Kivy supports basic GUI functionality like buttons, labels and media, but also has plugins from the "kivy garden" which can be used for more niche functionality. Kivy garden contains a graph plugin which is used for the graphing functionality of the time domain signals and FFT.

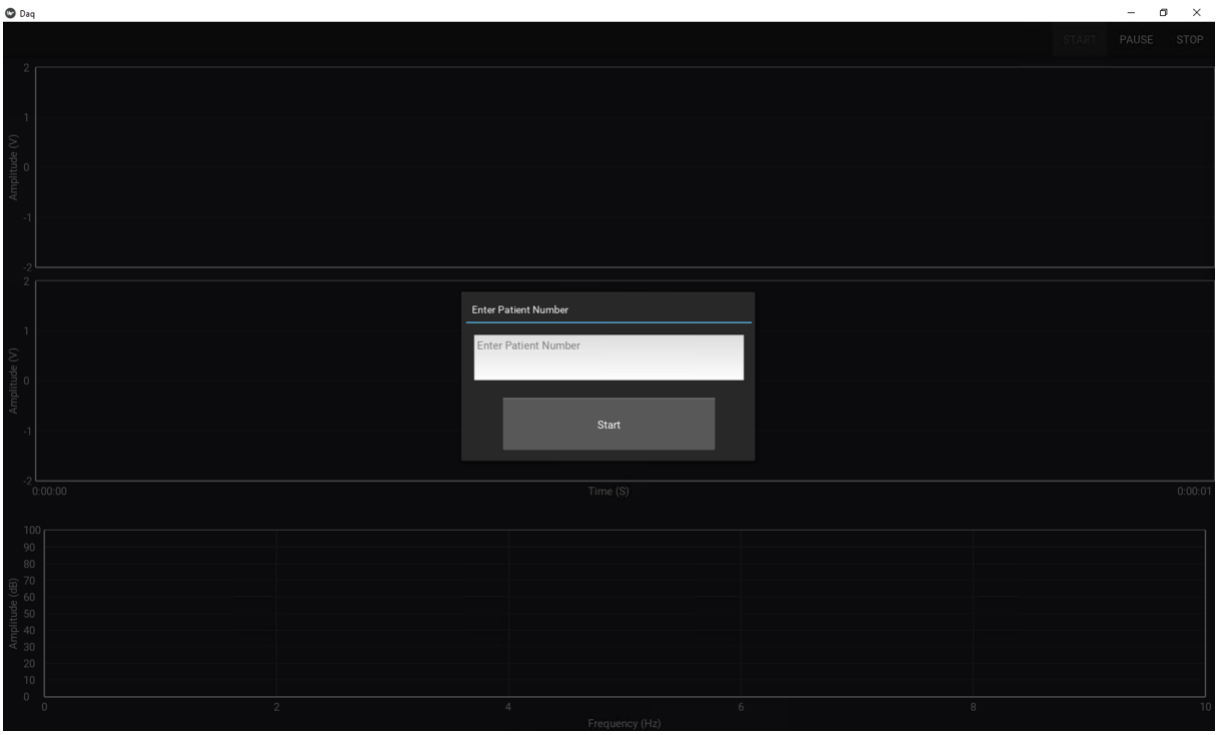


Figure 17. GUI asking for patient number after start

As is shown in Figure 17, whenever a measurement is started (the start button is pressed), a popup will appear asking for a patient number which can be any string used to identify the patient. Since

most studies require deidentified data, this is typically some long string generated by the liaison at the hospital. The patient number is often an integer number but it is stored as a string so it really can contain any Unicode character so long as they are not illegal directory characters (like "/" or "."). The string is used to create a directory within the "/Bioamp/Data" folder which will be named after string entered for the patient. Whenever that patient number is entered, the recorded files will be saved in that patient's directory.


Figure 18. Labeled GUI, EEnG 1 (light blue), EEnG 2 (red), ECG (green), X (dark blue), Y (purple) and Z (white)

As can be seen in Figure 18, the GUI has a control bar on top which is created using the "ActionBar" functionality from kivy which create a nice auto sizing bar to place buttons and other

useful things (like search) in. The top window displays the EEnG 1 and 2 signals, the second window contains the EKG and accelerometer data and the third window contains FFT on the EEnG signals. FFT is only performed on the EEnG for practical and performance reason as will be covered in the FFT section.


Figure 19. The GUI when gathering data

### 2.6.3 FFT

Live FFT is performed on the EEnG time domain data in order to see what frequencies are present. As has been shown in previous studies [1], slow waves typically have a frequency of around 9-13

cycles per minute (CPM) or 0.15 – 0.2 Hz. Performing a live FFT quickly allows the operator to determine if the measurement seems to be working in real time without having to wait until post processing the data. The FFT algorithm has been heavily optimized though the years and specifically for Python. As such, many fast implementations exist with the fastest (for most even numbered data sets) being the FFT from the scipy module which was used for this application.

In order to give some concept of the logic for the updating times used in the program, some run times are stated in this paragraph. However, it is important to note that the run times stated are for a specific Windows 10 computer which had 16 GB of RAM and a Ryzen 5 3400G processor. Times on other systems may vary due to hardware variations and even from run to run since Windows is not a real time operating system. Now, with that aside, the scipy module is quite fast, typically on the order of 1 – 2 ms for datasets of around 4,000 data points. However, some of the support operations required to run FFT involved converting the DEQs to NumPy arrays and interlacing two lists which takes an additional 3 – 5 ms. The clocked update time for the time domain graphs on the GUI is every 50 ms (20 Hz) in order to give a smooth viewing experience. Since other tasks must be performed during the 50 ms needed to update the graphs, the 4 – 7 ms taken by the FFT can often become too much and slow the update speed of the GUI down giving a suboptimal viewing experience. The FFT does not need to be updated as often since it doesn't have the same smooth scrolling nature as the time domain graphs, so the update time for the FFT is dropped to 500 ms which allows for smooth operation of the program.

The sampling rate of the DAQ is held at 1 kHz which based on the Nyquist theorem allows frequencies up to 500 Hz to be detected. Of course, as mentioned previously the frequencies of interest are at less than 1 Hz. Additionally, FFT performs a better approximation of the frequencies present when more cycles of that frequency are available. At a frequency of 0.1 Hz, one cycle only occurs every 10 s, so a 10 s FFT window would barely be able to approximate that frequency. As such, a large window of around 40 s is desired, but running FFT on 40,000 points would also be quite slow. The solution is to reduce the sampling rate for the data that FFT is run on. It is still desired to store the data from a 1 kHz sampling rate such that more analysis can be run later, but the processed FFT data can be sampled at a much lower rate of say 100 Hz which still leads to a buffer size of 4,000 but a window size of 40 s. This is what is done, there is a parameter in the main.py script called "FFTSampleReduction" which reduces the sampling rate by whatever integer that is set to. The default value is 10, which reduces the sampling rate from 1 kHz to 100 Hz and results in the scenario mentioned before.

### 2.6.4 File Saving

File saving is fairly straight forward process, but there are a few nuances that should be noted. File saving is a slow I/O operation, so it is ensured that the file I/O takes place on a separate thread than the one running the GUI, so the GUI continues to be responsive during each file save. Saving a line to a file in Python using a method like "file.write()" theoretically stores that values to file, but sometimes the OS will just keep them in a buffer in memory until the program stops running. This defeats the purpose of saving the data to a non-volatile memory space in the event of a power

outage or other failure. As such, the “os.fsync” function should be used in order to ensure the data is written to disk. This should be run occasionally to ensure the data is written to file. Using the clock scheduler from kivy, this task is run about every 5 s to ensure no more than 5 s of data can be lost in the event of a failure.

# CHAPTER 3: SYSTEM TESTING

Before patient tests were performed, the system needed to be validated to ensure all the measurements were working as expected. Many tests were performed on subsystems to improve their individual performance, but this section focuses on two tests that were performed on a complete system for final system functionality.

## 3.1 Motion Artifacts

Motion artifacts are a well-known phenomenon in the biomedical space. This section aims to describe the motion artifacts encountered in testing and some methods that were used to reduce them. As mentioned in the earlier cabling section, motion artifacts can be manifested in cabling due to the triboelectric effect and the cabling was changed in order to reduce that source, but motion artifacts also originate from changing of electrode-body impedance. That is, as the electrode moves, the contact resistance changes between the skin and the electrode which results in motion artifacts. This was minimized by resting cables with proper slack between the cable and patient to reduce cable forces on the electrode. Obviously gel based electrodes reduce impedance and even using a gel based electrode for ground greatly improved the performance of the dry electrodes used for EEnG. Beyond that, motion artifacts were able to be easily identified by the correlation between the biometric signals and the accelerometer signals as can be seen in Figure 20. Once motion artifacts were identified they had to be manually removed in post processing.

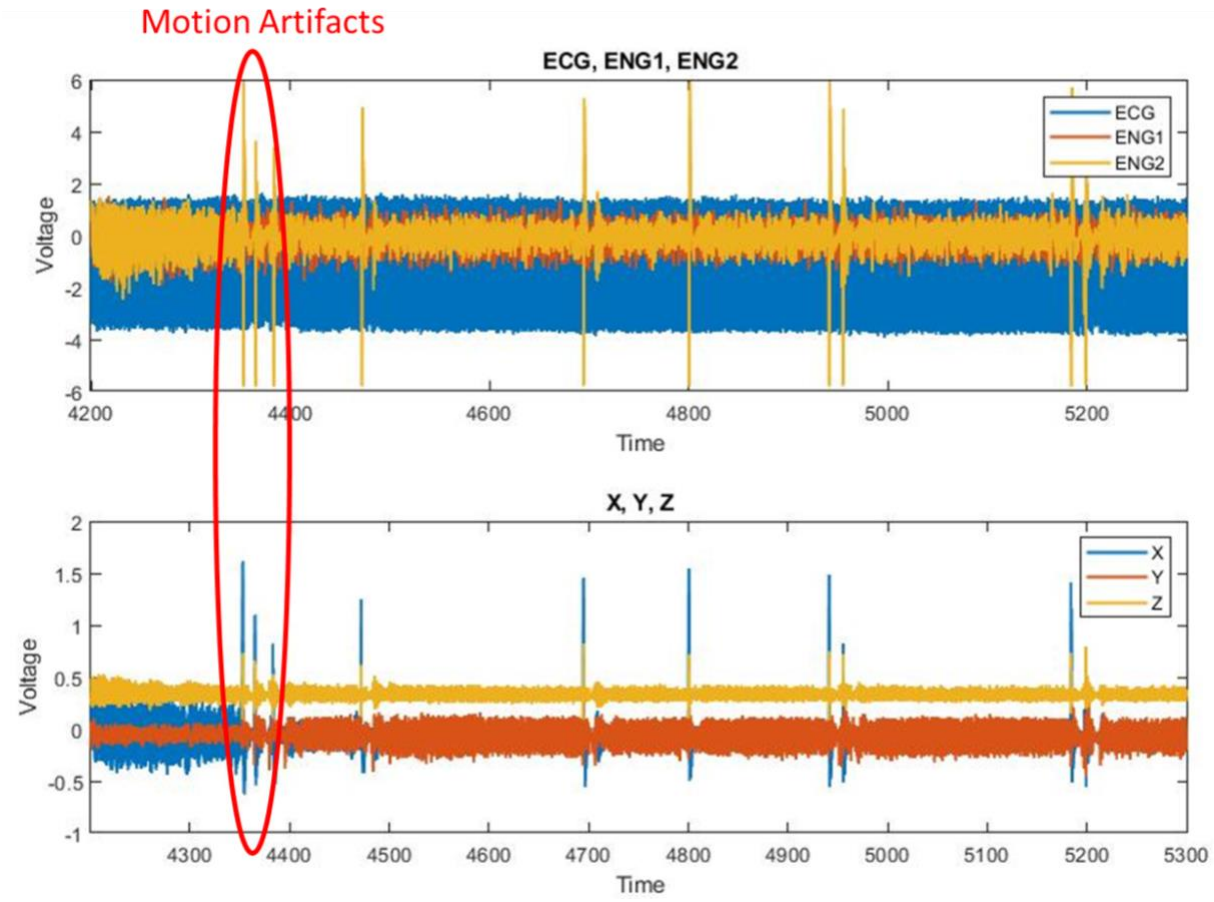


Figure 20. Time domain data from EEnG, EKG and accelerometer highlighting motion artifacts

## 3.2 Bipolar Concentric Electrode Testing

As mentioned previously, many studies [1] [2] [3] have measured slow waves in adults. Particularly [3], which used similar, flexible concentric electrodes to measure slow waves. As such, a measurement was taken on an adult in order to compare the results to the study to confirm the setup is working as expected.

### 3.2.1 Measurement Configuration

Similar to [3], a bipolar concentric electrode (CODE5000S0) was placed below the naval for the EEnG measurement. The ground electrode was place on the left hip, EKG electrodes were placed on the chest and an accelerometer was connected to measure breathing. As mentioned earlier Grass P511k amplifiers were used for the EEnG and EKG while a custom filter and amplifier circuit was used for the accelerometer. The EEnG measurement was amplified with a gain of 20,000 which was the highest value that allowed the signal to be within bounds of the ADC without significant noise levels. The EKG signal was amplified by 2,000 which was the correct value to keep it in bounds of the ADC. Both EKG and EEnG were run through a bandpass filter from 0.1 Hz – 30 Hz before amplification. The accelerometer signals were run through a bandpass filter from 0.1 Hz – 3 Hz before and then amplified by 100. Recording took place for 10 minutes in which the patient was kept still and advised to breathe normally. The Grass P511k amplifier settings are summarized in Table 2.

Table 2. Setting used for the analog filters and amplifiers

| Amplifier Settings | |
|---|---|
| **Parameter** | Value |
| **ECG Gain** | 2,000 |
| **Resp Gain** | 100 |
| **ENG Gain** | 20,000 |
| | |
| **ECG Filter** | 0.1 - 30 Hz |
| **ENG Filter** | 0.1 - 30 Hz |
| **60 Hz Notch** | Yes |
| | |
| **Resp Filter** | 0.1 - 3 Hz |

Figure 21. Electrode and accelerometer placement for an adult test to compare system performance

### 3.2.2 Results

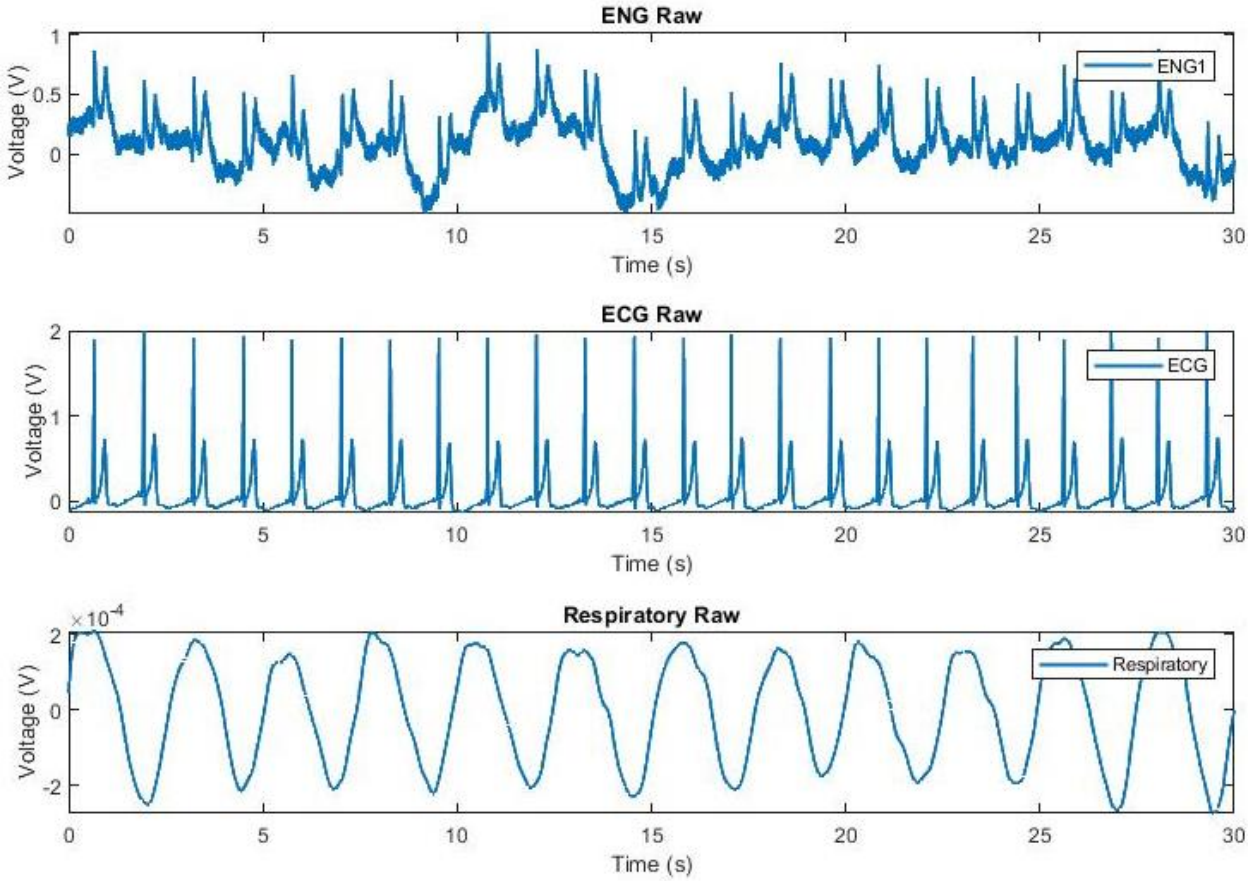


Figure 22. 30 s view of the raw time domain waveforms collected from the EEnG electrode (top), the EKG electrodes (center) and the accelerometer (bottom) all without gain normalization

The results were first normalized by gain in order to get the proper amplitude of each signal and then digitally band pass filtered by a bidirectional 4$^{th}$ order Butterworth filter which was implemented using the "butter" and "filtfilt" functions in Matlab. EKG was filtered between 0.1 – 10 Hz, EEnG was filtered between 0.1 – 0.5 Hz and respiratory was filtered between 0.1 – 3 Hz. The signals were then resampled at a lower frequency. EKG was resampled at 10 Hz, EEnG was

resampled at 4 Hz and respiratory was resampled at 10 Hz. The post processed signals are shown in Figure 23.

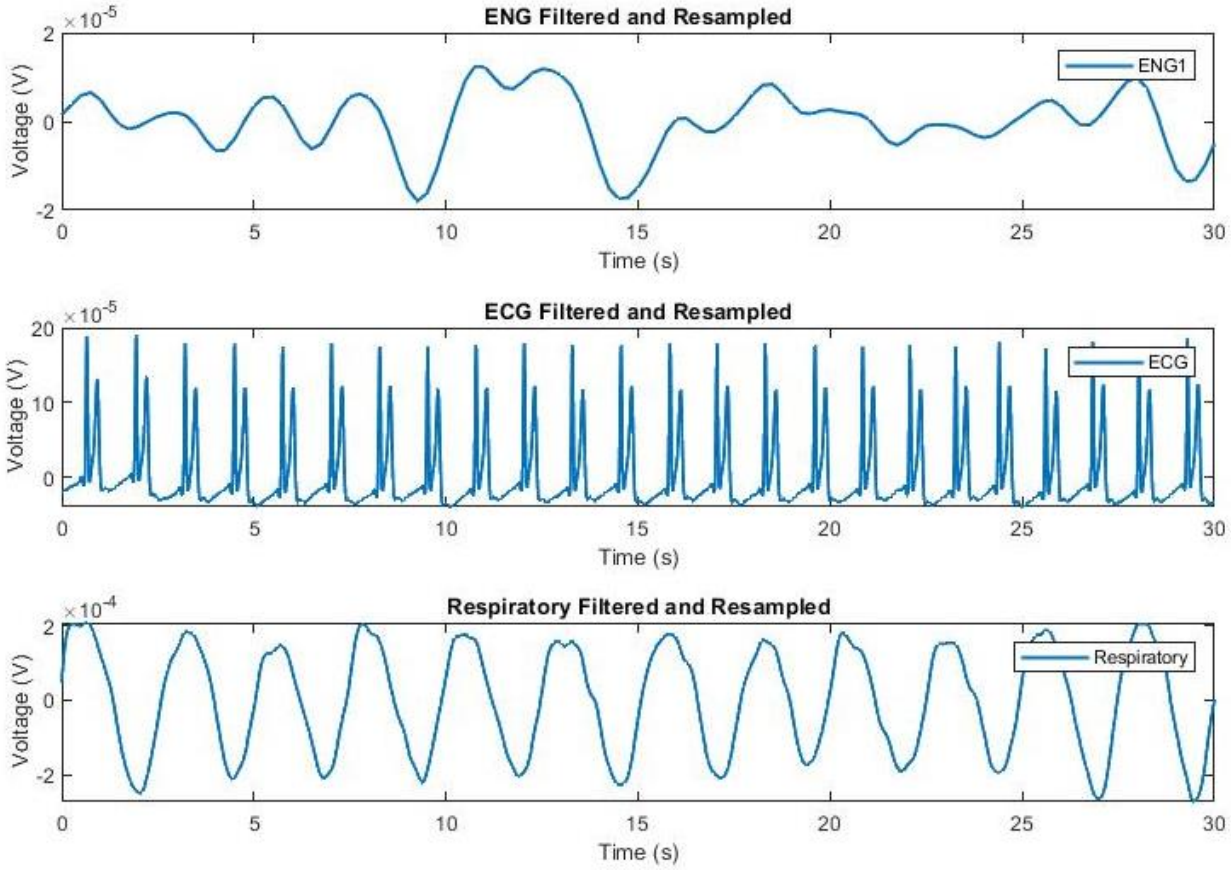


Figure 23. 30 s view of the time domain waveforms of post processed data collected from the EEnG electrode (top), the EKG electrodes (center) and the accelerometer (bottom) with gain normalization

As can be seen in the top plot of Figure 23, the EEnG signal shows significant respiratory artifacts which make it difficult to make out the overall trend of the data. As a result, spectral analysis is used in order to determine the frequency components present in the EEnG signal and determine

the frequency of the slow wave signal. In order to get reasonable frequency resolution (0.5 CPM) while avoiding significant non-stationarities, a window size of 120 s was chosen for the spectral analysis which is consistent with previous published work.

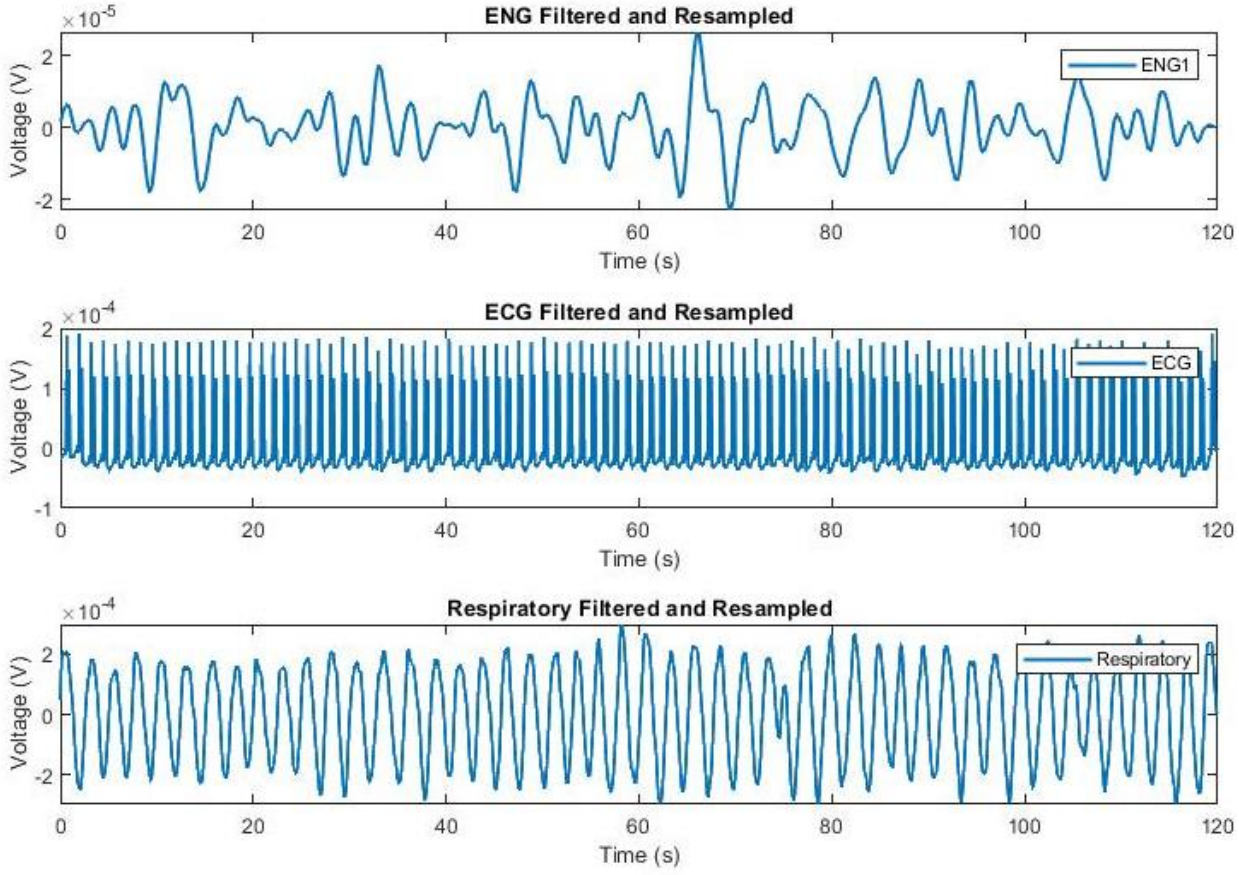


Figure 24. 120 s view of the time domain waveforms of post processed data collected from the EEnG electrode (top), the EKG electrodes (center) and the accelerometer (bottom) with gain normalization

The autoregressive Yule-Walker method was used to go from the time domain to the spectral domain due to its unconditional stability. The order chosen for the EEnG signal was 120 (the length

of the window in seconds) and the order for EKG and respiratory was chosen as 3,000 since the sampling rate for both signals was 25 times higher than EEnG. The autoregressive order can be as large as the number of samples in the window (480), as which point further increasing order will yield no further changes. The order of 120 was chosen because it provides a good tradeoff between resolving all peaks without adding additional noise, which is why it has been chosen in other papers The results of the PSD are shown in Figure 25.

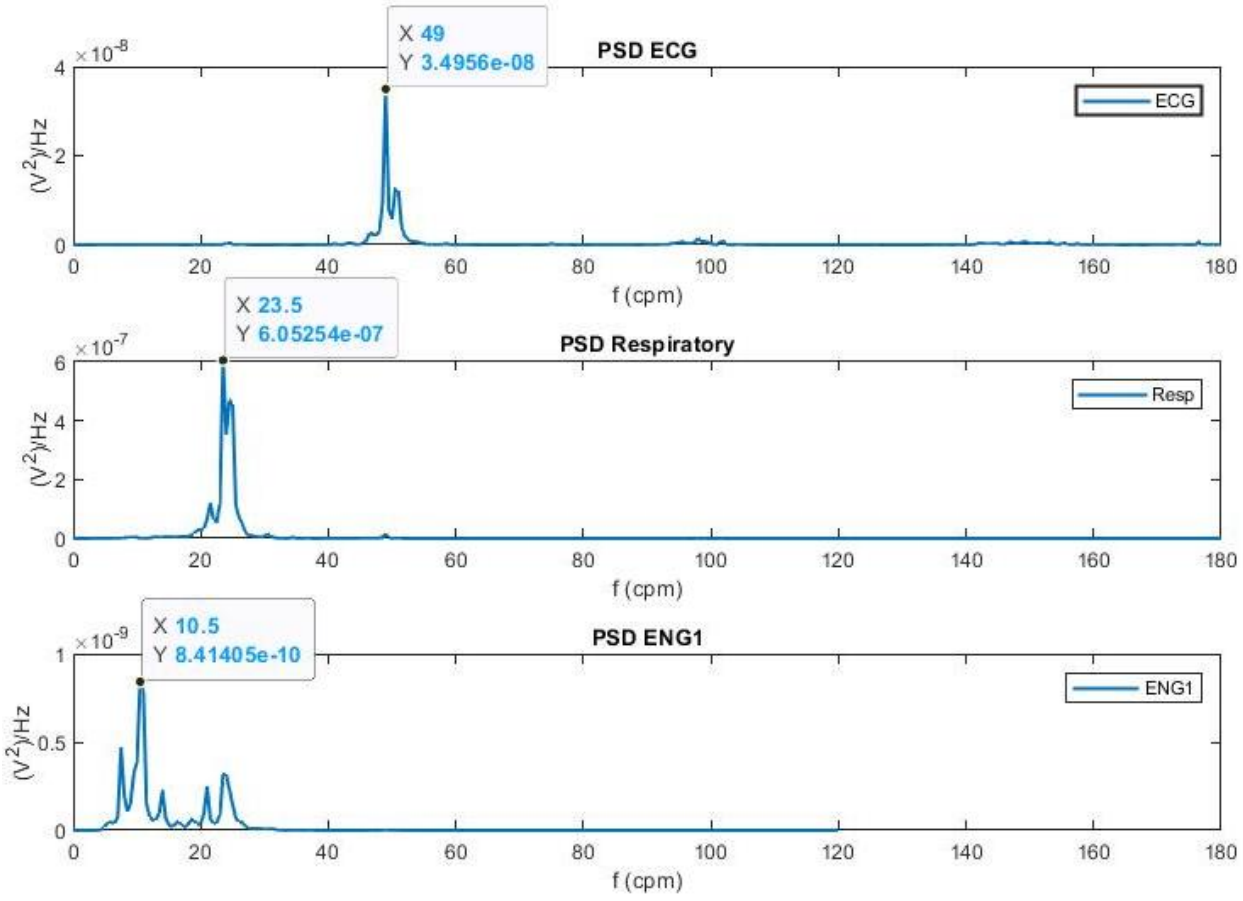


Figure 25. PSD of the adult test using a bipolar concentric electrode

As expected from visual inspection of the EEnG, the respiratory frequencies are present in the EEnG spectrum in addition to the slow waves. The 10.5 CPM peak is attributed to the slow wave and is consistent with previous studies which have found slow waves to be between 10 – 12 CPM in adults. There is a peak below the primary slow wave peak that is around 7 CPM. This lower frequency may be due to the colon or may be due to baseline drift in the signal but is not regarded as the slow wave frequency due to it's lower amplitude.

## 3.3 Noise Measurement Test

To validate that the frequencies observed during measurement were of biological origin and not an artifact of the system, a measurement was performed where there were no signals. Two electrodes were placed atop a damp paper towel with their plastic backing still on. The damp paper towels provided a weak amount of grounding to wick away charge and prevent significant charge buildup. The plastic backing was left on so the electrodes could be used for further experiments later and because the high resistance of the plastic made the experiment noisier if anything. The amplifiers were then set to the same settings as the previous section and the data was post processed in the same way in order to provide a direct comparison between the spectrum of a noise measurement and an actual measurement. Performing the data analysis in the same way also allowed as a check to ensure that the spectra observed were not a result of the post processing.


Figure 26. Concentric electrodes used for measuring EEnG placed

As can be seen in Figure 27, the noise measurement is more than an order of magnitude below the measured signal in the time domain and 3-4 orders of magnitude smaller in the frequency domain. The reason that the noise appears to have similar spectral content is because of filtering and FFT binning. Since both signals are band pass filtered between 0 CPM – 30 CPM (0.1 – 0.5 Hz), their spectra only shows up in that range, and the noise appears to have peaks because frequency resolution is only 0.5 CPM so noise that is in-between gets grouped into those peaks. 3 – 4 orders of magnitude is considered to be a high enough margin for this measurement and proves that the measurement is more than just noise.

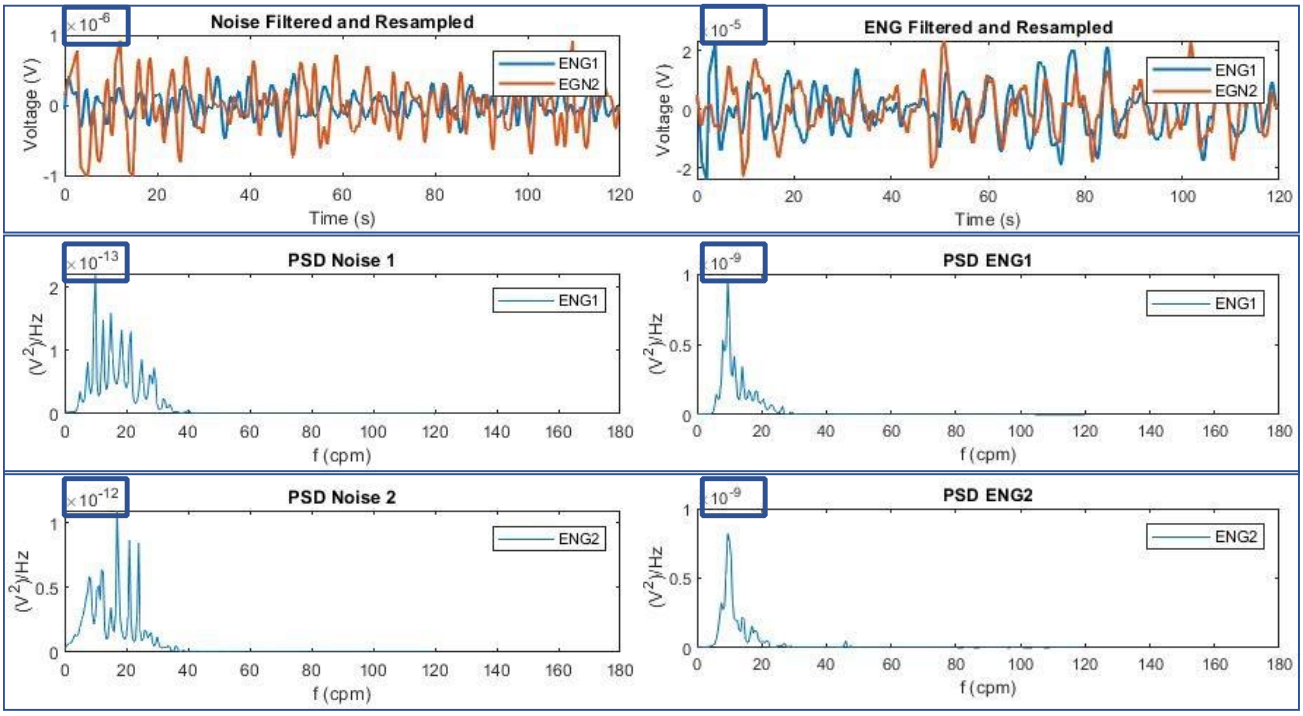


Figure 27. Time domain and spectral density of noise measurement (left) and a real measurement (right)

Figure 27 is a nice comparison because it shows the difference between a real signal and a noise signal when all signal processing has been applied. This allows the effects of both the system and signal processing to be seen, but some may wonder what the entire spectrum looks like when filtering is not applied. That is to say, noise doesn't really look like noise when it is being filtered because then it is not strewn across the measured spectrum as is typically expected. To satisfy this curiosity, the results were analyzed with a high pass filter instead of a bandpass filter in order to see the rest of the spectrum and the AR order was increased from 120 to 480 to give maximum resolving in order to see all peaks.

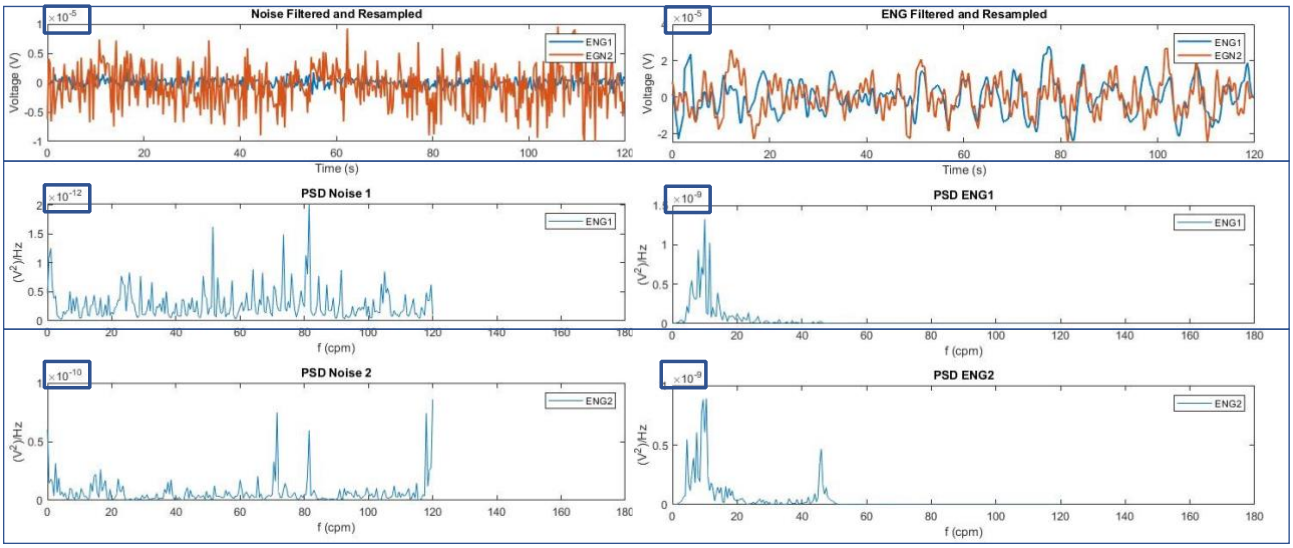


Figure 28. Comparison between noise (left) and measurement (right) when a high pass filter is applied at 3 CPM to remove DC offset

As can be seen in Figure 28, the time domain and frequency domain look quite different without the bandpass. In the noise measurement on the left, if can be seen from the time domain that EEnG1 is much quieter than EEnG2 which is not as apparent when the bandpass filter is applied because it filters the frequencies that show up more strongly in EEnG2. The other thing that can be seen is that with the higher frequencies present in EEnG2, the order of the amplitude is similar to the amplitude of our measured signal. This is a bit concerning, but it must be remembered that this higher amplitude frequencies get filtered out and the frequencies of interest are still much smaller in amplitude. The other good news is that while the noise measurement does have a surprising number of distinct peaks, they do not occur in the measurement range of interest and are likely random. The final thing to consider is that the actual noise floor in a real measurement is somewhat lower that what is shown here. This is because the outer and inner conductors of the electrodes

have a very high impedance between them in this measurement and a much lower impedance in an actual measurement. This high impedance makes the electrodes much more susceptible to noise.

# CHAPTER 4: PATIENT TESTING

Measurements were taken over a 90-minute period occurring just after feeding. This was done for logistical reasons; the neonate is typically awake for feeding which is a good time to place electrodes and the neonate typically sleeps after feeding which helps reduce motion artifacts. All patient testing was performed at Nemours Children's Hospital (Orlando, FL) in the neonatal intensive care unit (NICU). Deidentified data was provided from the measurements for further analysis which will be presented in this section.

## 4.1 Measurement Configuration

As described in the measurement setup section, two bipolar concentric electrodes were placed on either side of the naval to measure EEnG. EKG electrodes were placed on the chest, slightly to the side so as not to disturb the EKG electrodes placed by the hospital. The grounding electrode was placed on the side and the accelerometer was placed in the middle of the abdomen where respiration displacement appeared to be the greatest. The two EEnG signals were amplified by 20,000 and band pass filtered before amplification from 0.1 Hz – 30 Hz. EKG signals were amplified by 10,000 and also bandpass filtered between 0.1 Hz – 30 Hz. Both EKG and EEnG were filtered and amplified with P511k amplifiers. The accelerometer signals were amplified by 100 and pass filtered before amplification from 0.1 Hz – 3 Hz using custom hardware.

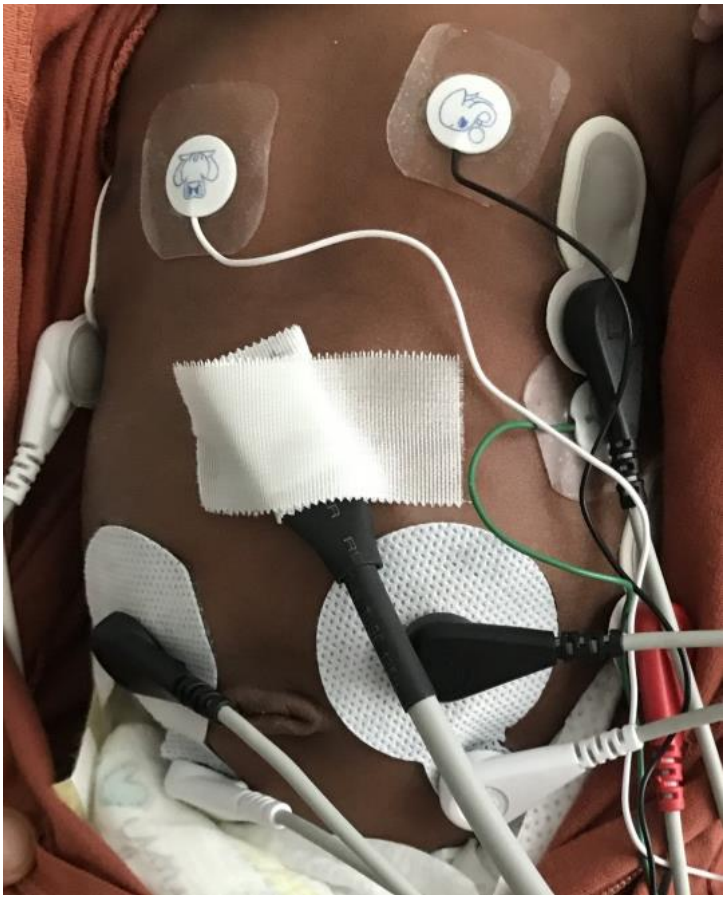

Figure 29. Electrode placement on a neonate

Most neonates were awake after feeding and took some time to settle down which led to 30 – 50 minutes of artifacted data at the beginning of the measurement. Some neonates had mild gastric distress which would result in motion every few minutes to pass gas which also resulted in motion artifacts. Overall, it was difficult to get long periods of artifact free data which limited the data analysis to artifact free windows.

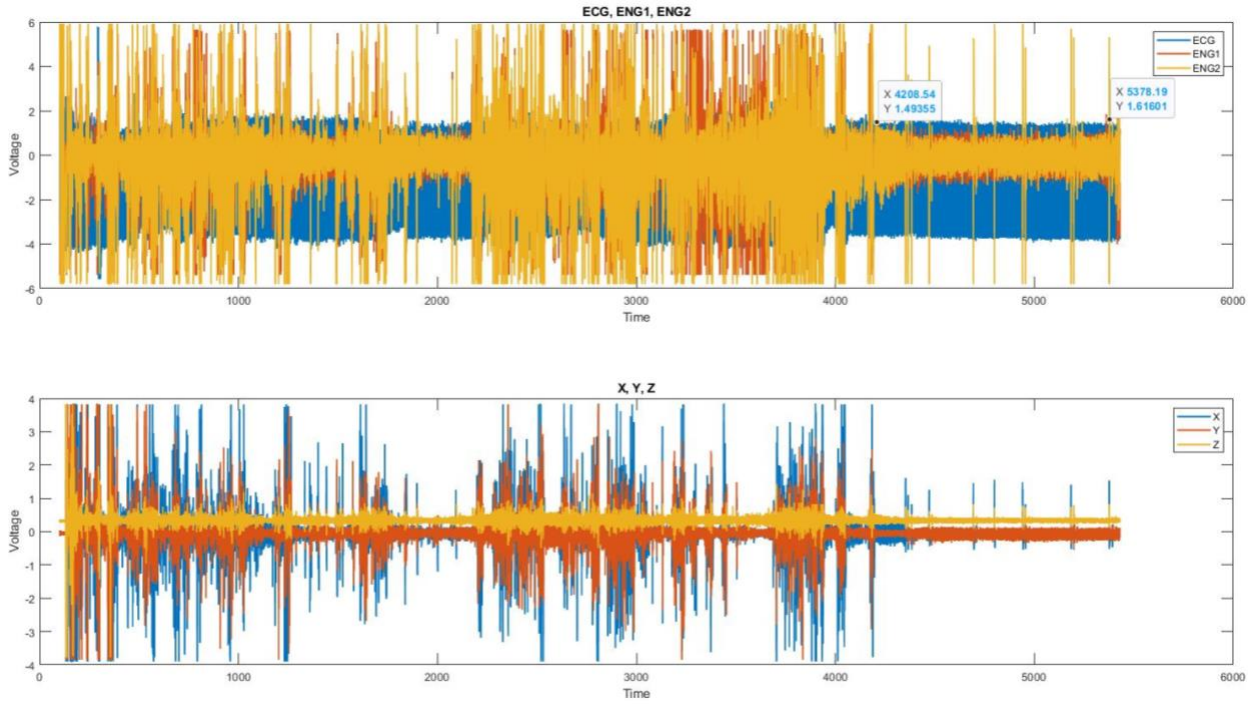


Figure 30. Raw signals from 90 minutes of recording

As can be seen in Figure 30, there are significant motion artifacts in the first 60 minutes of recording. The motion artifacts can be easily identified because they appear both on the accelerometer and electrode measurements. The analysis for this particular measurement took place on data from the last 20 minutes and the same is true for most other measurements.

## 4.2 Post Processing Methods

The signals from the patients were processed in the same way as the results from an adult in Bipolar Concentric Electrode Testing. To avoid regurgitating the same sentences the parameters are

summarized in Table 3. The same processing methods are used for all patient data when analyzing 120 s windows.

Table 3. Post processing parameters used for 120 s windows of data

| Post Processing Parameters | |
|---|---|
| **Parameter** | **Value** |
| **Data Size** | 120 s |
| | |
| **ECG Filter** | 0.1 - 10 Hz |
| **Resp Filter** | 0.1 - 3 Hz |
| **ENG Filter** | 0.1 - 0.5 Hz |
| | |
| **ECG Sampling Rate** | 100 Hz |
| **Resp Sampling Rate** | 100 Hz |
| **ENG Samplin Rate** | 4 Hz |
| | |
| **ECG AR Order** | 3000 |
| **Resp AR Order** | 3000 |
| **ENG AR Order** | 120 |
| | |
| **PSD Method** | Yule-Walker |

## 4.3 Results

The resampled and filtered time domain data is show in Figure 31 for EEnG, ECG and respiratory signals. The time domain data for the neonates (patients) is somewhat different than the data collected from adults shown in earlier section. The most notable difference is that the ECG and respiratory signals are much higher in frequency. ECG is on the order of 150 CPM instead of 50 CPM and respiratory is around 50 CPM instead of 25 CPM. As such, ECG and respiratory can be easily filtered out from EEnG using the bandpass Butterworth filter which allows for more clear viewing of the slow waves in the time domain. Since the EEnG signal has still been highly filtered

the slow waves appear more sinusoidal in nature than when compared to the slow waves measured directly on the small intestine.

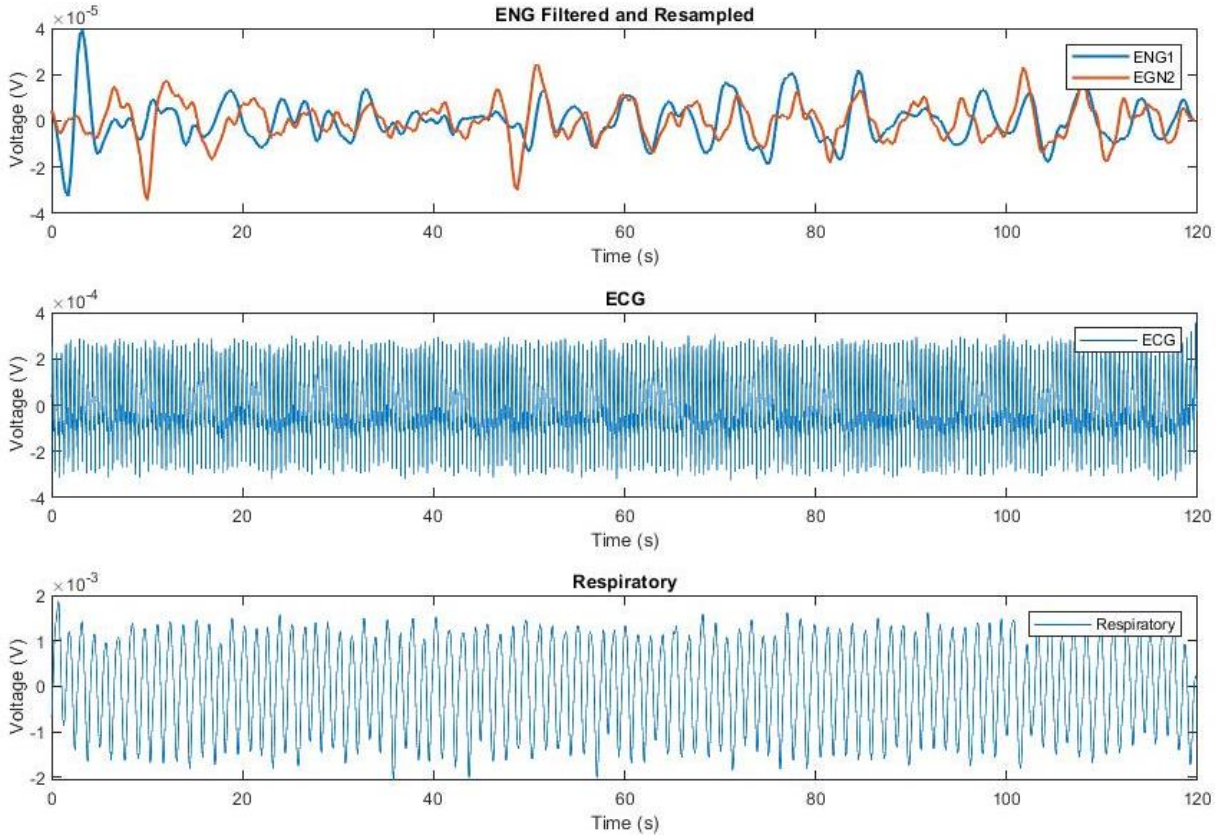


Figure 31. Time domain data of the filtered and resampled EEnG (top), ECG (middle) and respiratory (bottom)

Again, moving to the spectral domain provides some insight into what signals are present in the measurement. As mentioned earlier, the ECG and respiratory in neonates is quite high compared to adults, 130.5 CPM and 46 CPM respectively. The slow wave frequency measured on both the right and the left electrode were the same at 9.5 CPM.

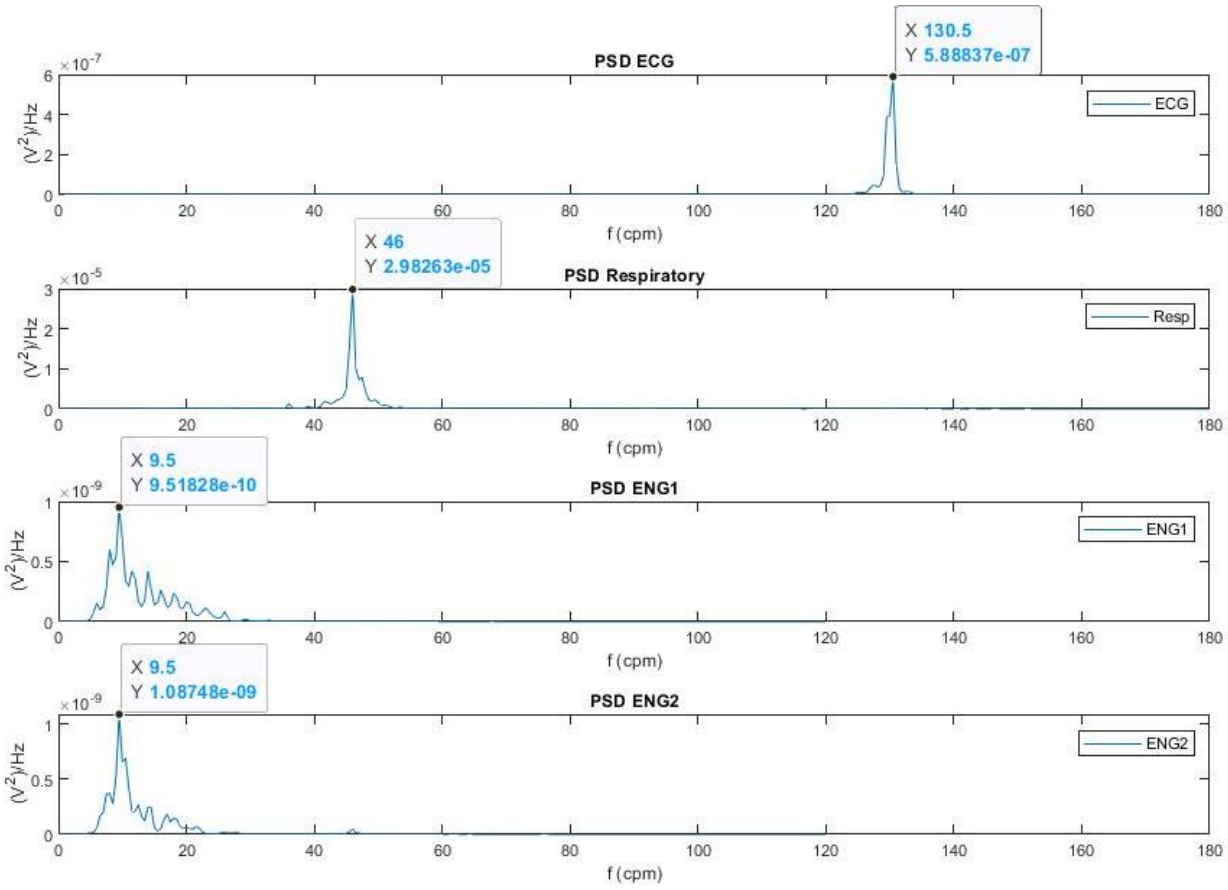


Figure 32. PSD on 120 s window of the ECG (top), respiratory (middle) and EEnG (bottom)

As is well known in signal processing, frequency resolution is directly related to the window size. In Figure 32, a 120 s window is being evaluated which results in a frequency resolution of 0.5 CPM. This results in some "frequency binning" issues as alluded to in the noise measurement section. Even though we see a peak a 9.5 CPM all spectral content between 9.25 CPM and 9.75 CPM is being placed into that signal peak, so the peak location and height may differ.

$$Frequency\ Resolution = \frac{Sampling\ Rate}{Number\ of\ Samples} = \frac{4\ (Hz)}{480} = 0.0083\ (Hz) = 0.5\ (CPM)$$

Equation 1. Frequency resolution of a FFT transform

In order to increase the frequency resolution, the window size was increased from 120 s to 480 s which improves the frequency resolution from 0.5 CPM to 0.125 CPM. This allow for better resolution in the peaks which can be seen in Figure 33. The trouble with larger window sizes is that it is more difficult to find artifact free windows and non-stationarities become more significant. The peak of EEnG1 moves from 9.5 to 9.75 CPM and the peak on EEnG2 moves from 9.5 CPM to 9.125 CPM.

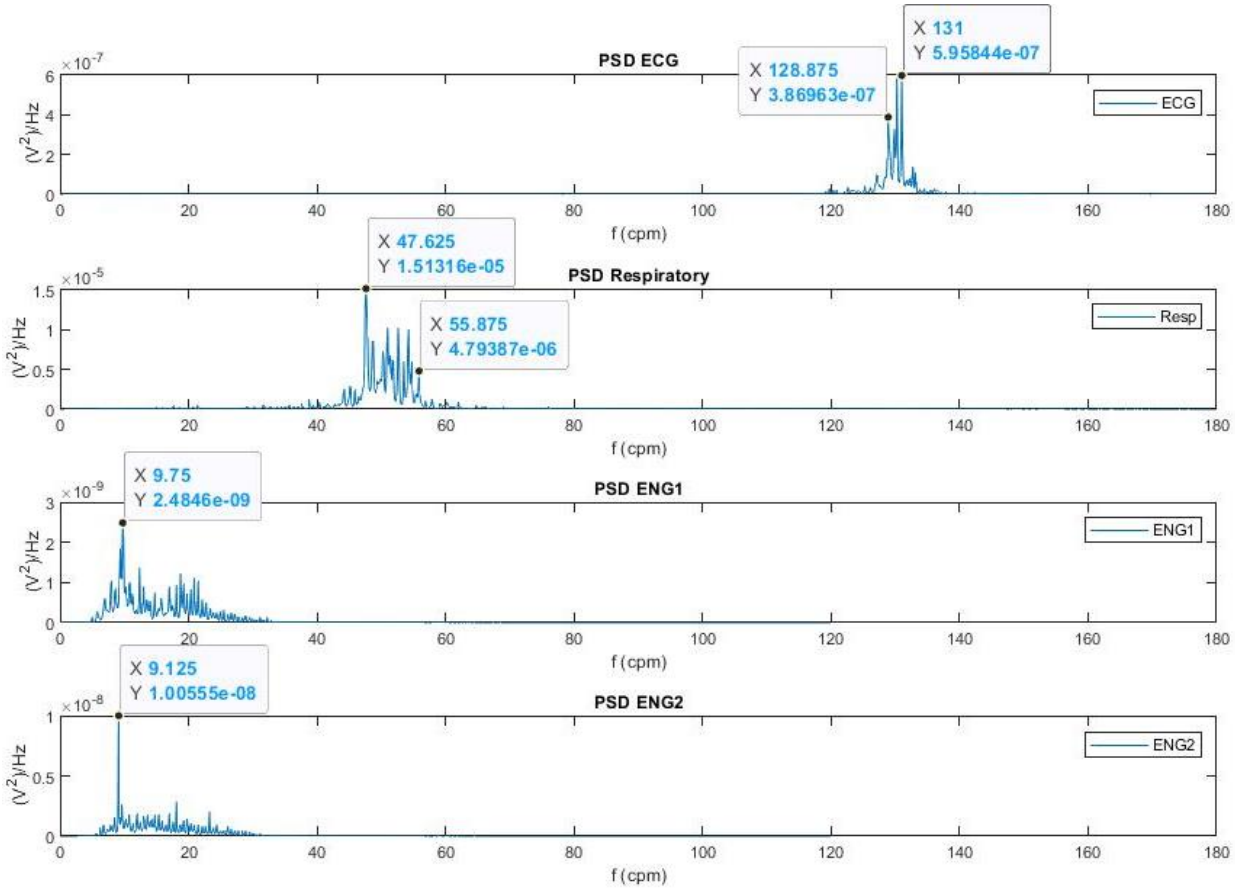


Figure 33. PSD on 480 s window of the ECG (top), respiratory (middle) and EEnG (bottom)

Three more measurements of neonates were taken in the same fashion as the initial measurement and with the same post processing. The primary peak in EEnG was identified for the 120 s recording window and summarized in Table 4.

Table 4. Summary of patient slow wave frequencies

| Patinet Number | EEnG 1 (CPM) | EEnG2 (CPM) |
|---|---|---|
| 1 | 9.5 | 9.5 |
| 2 | 9 | 8.5 |
| 3 | 10.5 | 8.5 |
| 4 | 8.5 | 8.5 |

## 4.4 Results Discussion

As mentioned previously, the range of slow waves in adults is between 8 CPM – 12 CPM, which places the measured result within range. One interesting observation is that in two of the four measurement, the signal on the right electrode (EEnG2) showed a lower frequency than the signal on the left electrode (EEnG1). This is slightly surprising because the right electrode should be closer to the duodenum which is the top of the small intestine. Generally, the frequency of the slow waves decreases when moving down the small intestine [7]. However, since the electrodes are being placed relative to the naval without any ultrasound or other locating method, it is possible that the left electrode is far enough below the duodenum that it primarily picks up signal from the jejunum. Overall, the frequencies observed are consistent with previous studies in adults and align with the expected values.

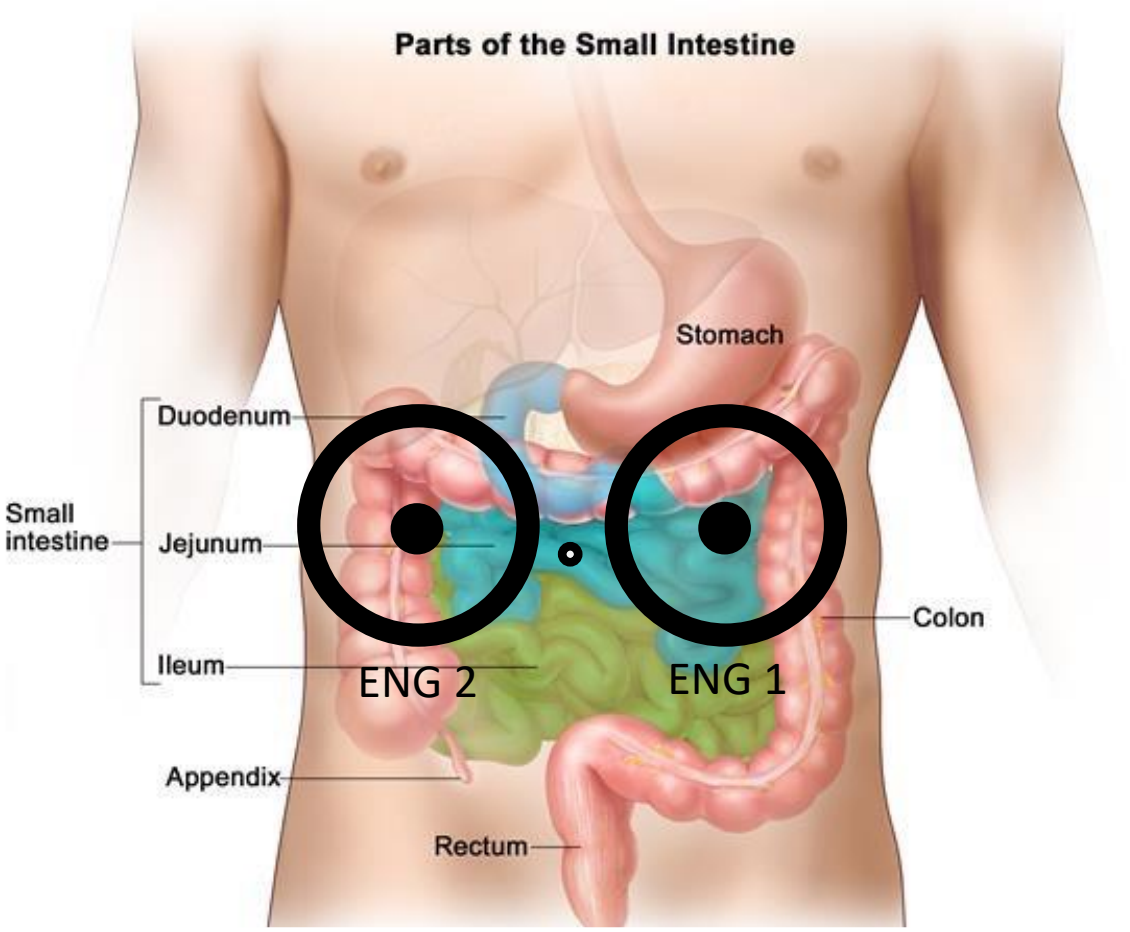


Figure 34. Diagram of the abdomen showing the relative scale of the EEnG electrodes to the intestines

# CHAPTER 5: CONCLUSION

## 5.1 Accomplishments

A custom system was built with the capability to measure ECG, respiratory and EEnG and to record and display those signals in live time. The system was validated on adults and shown to align with results from previous studies in adults and shown that the measured results were not a function of system noise. The system was then used to perform measurements of EEnG in neonates resulting in the first slow waves measured in neonates.

## 5.2 Future Works

Significant work still remains in this area, primarily in relation to improving the measurement setup. Motion artifacts remain a prominent issue which may require the use of an "active electrode," that is a PCB that can stack atop the electrodes and provide amplification and possibly transmit via a current loop transmitter. Smaller electrodes are another possible improvement which would allow for more localized signals. Additionally, electrode arrays might allow for both localization and phase of signal to "follow" slow waves through the intestines. Some smaller improvements could be made in terms of integrating the system into a single PCB which would shrink it's size significantly.

As far as the non-technical future work goes, more measurements need to be taken to confirm that the slow waves measured at statistically significant. Once slow waves have been thoroughly vetted in healthy neonates, continuous monitoring will need to take place in neonates that are at high risk

for NEC. Continuous monitoring over long periods will allow for trends in slow wave frequency to be observed. These trends can then be correlated to the baby's health to determine if slow wave frequency trends relate to abdominal swelling and NEC.

# APPENDIX A

# CODE DAQ.KV

```
#:kivy 1.0.9
#:import MeshLinePlot kivy.garden.graph.MeshLinePlot
#:import Factory kivy.factory.Factory
#:import datetime datetime

<PatientPopup@Popup>:
    title: "Enter Patient Number"
    auto_dismiss: False
    size_hint: (0.25, 0.25)
    BoxLayout:
        orientation: "vertical"
        AnchorLayout:
            TextInput:
                id: input
                hint_text:'Enter Patient Number'
                multiline: False
                size_hint: 0.95, 0.7
                anchor_x: 'right'
                anchor_y: 'bottom'
        AnchorLayout:
            Button:
                id: pButton
                text: 'Start'
                size_hint: 0.75, 0.8

# Define your background color Template
<BackgroundColor@Widget>
    background_color: 1, 1, 1, 0.1
    canvas.before:
        Color:
            rgba: root.background_color
        Rectangle:
            size: self.size
            pos: self.pos
# Now you can simply Mix the `BackgroundColor` class with almost
# any other widget... to give it a background.
<BackgroundLabel@Label+BackgroundColor>
    background_color: 0, 0, 0, 0
    # Default the background color for this label
    # to r 0, g 0, b 0, a 0

<GraphValues>:
```

```
eng_graph: ENG
vitals_graph: ECG_ACCEL
frequency_graph: FFT
patient_popup: Factory.PatientPopup()

BoxLayout:
    size: root.width, root.height
    orientation: "vertical"
    ActionBar:
        size_hint: (1, 0.05)
        ActionView:
            ActionPrevious:
                with_previous: False
                app_icon: ''
            ActionButton:
                id: start_button
                text: "START"
                on_press: root.start()
                disabled: False
            ActionButton:
                id: pause_button
                text: "PAUSE"
                on_press: root.pause()
                disabled: True
            ActionButton:
                id: stop_button
                text: "STOP"
                on_press: root.stop()
                disabled: True
    Graph:
        id: ENG
        size_hint: (1, 0.3)
        plot: MeshLinePlot
        background_color: [0.15, 0.15, 0.17, 1]
        ylabel:'Amplitude (V)'
        #x_ticks_major:1
        y_ticks_major:1
        y_grid_label:True
        x_grid_label:True
        padding: 5
        x_grid:True
        y_grid:True
```

```
    xmin:root.xMin
    xmax:root.xMax
    ymin:-2
    ymax:2

Graph:
    id: ECG_ACCEL
    size_hint: (1, 0.3)
    plot: MeshLinePlot
    background_color: [0.15, 0.15, 0.17, 1]
    ylabel:'Amplitude (V)'
    #x_ticks_major:1
    y_ticks_major:1
    y_grid_label:True
    x_grid_label:True
    padding: 5
    x_grid:True
    y_grid:True
    xmin:root.xMin
    xmax:root.xMax
    ymin:-2
    ymax:2

# Time axis labels
BoxLayout:
    size_hint: (1, 0.05)
    orientation: "horizontal"
    BackgroundLabel
        text: str(datetime.timedelta(seconds=root.xMin))[0:10] + ' '
        background_color: [0.15, 0.15, 0.17, 1]
        text_size: self.size
        halign: 'left'
        valign: 'top'
        padding:  (40, 0)
    BackgroundLabel
        text: "Time (S)"
        background_color: [0.15, 0.15, 0.17, 1]
        text_size: self.size
        halign: 'center'
        valign: 'top'
    BackgroundLabel
        text: str(datetime.timedelta(seconds=root.xMax))[0:10] + ' '
```

```
        background_color: [0.15, 0.15, 0.17, 1]
        text_size: self.size
        halign: 'right'
        valign: 'top'

Graph:
    id: FFT
    size_hint: (1, 0.3)
    plot: MeshLinePlot
    background_color: [0.15, 0.15, 0.17, 1]
    xlabel:'Frequency (Hz)'
    ylabel:'Amplitude (dB)'
    x_ticks_major:2
    y_ticks_major:10
    y_grid_label:True
    x_grid_label:True
    padding:5
    x_grid:True
    y_grid:True
    xmin:0
    xmax:10
    ymin:0
    ymax:100
```

# APPENDIX B

# CODE MAIN.PY

```
# NI imports
import nidaqmx
from nidaqmx.stream_readers import (AnalogSingleChannelReader,
AnalogMultiChannelReader)
from nidaqmx.constants import (AcquisitionType, CountDirection, Edge,
   READ_ALL_AVAILABLE, TaskMode, TriggerType, TerminalConfiguration)

# General imports
import collections
import numpy
import time
from threading import Thread
import threading
import multiprocessing
from copy import copy
from datetime import datetime
import os
from scipy.fftpack import fft
import queue

# Kivy imports
from kivy.app import App
from kivy.lang import Builder
from kivy.garden.graph import MeshLinePlot, Graph
from kivy.clock import Clock
from kivy.uix.widget import Widget
from kivy.properties import (NumericProperty, ReferenceListProperty,
   ObjectProperty)
from kivy.uix.boxlayout import BoxLayout

class FFT:
   # Takes in a deq in the time domain and returns a list of tuples (freq, mag)
   @staticmethod
   def FFTFromDEQ(timeDEQs, samplingRate):

      # Convert to a numpy array and take the fft for each channel
      chFrequencies = []
      i = 0
      for DEQ in timeDEQs:
         timeArray = numpy.asarray(DEQ)[:,1]
         chFrequencies.append(numpy.abs(fft(timeArray)))
         # Logarithimic mode, this is a bit slower
```

```
            #chFrequencies.append(20*numpy.log10(numpy.abs(fft(timeArray))))
            i+=1

        # Generate the X axis values which are the discrete frequency values
        N = len(chFrequencies[0])
        n = numpy.arange(N)
        T = N/samplingRate
        freqValues = n/T

        # Generate a list of tuples from the X and Y values for graphing
        freqGraphs = []
        for ch in chFrequencies:
            freqGraphs.append(tuple(zip(freqValues, ch)))

        return freqGraphs

class FileHandling:

    def __init__(self, patientNumber, numChannels):
        self.patientNumber = str(patientNumber)
        self.numChannels = numChannels
        self.fileName = str(patientNumber) + "_" + str(datetime.now().strftime("%Y_%m_%d
%I_%M")) + ".csv"
        self.directory = str(os.getcwd()) + "\\Data\\" + str(patientNumber) + "\\"

        # Make the directory for the patient if it doesn't exist already
        if (not os.path.isdir(self.directory)):
            os.mkdir(self.directory)

        # Create the file, I suppose this could be done inline instead
        self.CreateFile()

    def CreateFile(self):
        self.file = open(self.toRaw(self.directory + self.fileName), "x")

        headerLine = "Time"
        for channel in range(self.numChannels):
            headerLine = ".join([headerLine, ",Ch", str(channel)])

        headerLine = ".join([headerLine, "\n"])
        self.file.write(headerLine)
```

```
    def SaveData(self, time, samplesRead):
        dataLine = str(time)

        for sample in samplesRead:
            dataLine = ''.join([dataLine, ",", str(sample)])

        dataLine = ''.join([dataLine, "\n"])
        self.file.write(dataLine)

    def Close(self):
        # Flush the buffer and ensure everything is saved to disk before closing
        self.file.flush()
        os.fsync(self.file.fileno())
        self.file.close()

    def FileSync(self):
        self.file.flush()
        os.fsync(self.file.fileno())

    def toRaw(self, string):
        return fr"{string}"

# Responsible for getting data from the NI DAQ and storing it
# If a new DAQ is used, write a new DAQ class and as long as it has channel
# buffers as deques it will be compatible with all of the code
class NIDAQ:

    def __init__(self, device, numChannels = 1, samplingRate = 5, histLen = 20,
FFTSampleReduction = 4, FFTChannels = 2, fileName = "01"):
        self.device = device
        self.samplingRate = samplingRate  # Sampling rate in Hz
        self.historyLength =  histLen # Number of samples in buffer to be displayed
        self.numberOfChannles = numChannels # Assumes channels are 0 -> numberOfChannles
        self.minVal = -2 # Sets the range for the DAQ in volts, reducing range increases precision
        self.maxVal = 2
        self.timeElapsed = 0 # Keeps track of how long data has been recorded
        self.sampsAtATime = 4 # This sets the number of samples to grab at a time
        self.FFTLen = histLen
        self.FFTSampleReduction = FFTSampleReduction
        self.FFTChannels = FFTChannels
        self.DAQThread = None
        self.process = None
```

```
        # This creates a list of FIFO buffers of a fixed size, these buffers are how you access the
channel data
        self.channelBuffers = []
        for i in range(self.numberOfChannles):
            # deque(dataype, maxlen of deque), note [(0,0)] is a list of tuples
            self.channelBuffers.append(collections.deque([(0, 0)], self.historyLength))

        # This creates a list of buffers for FFT since FFT may require more data
        self.FFTBuffers = []
        for i in range(self.FFTChannels):
            # deque(dataype, maxlen of deque), note [(0,0)] is a list of tuples
            self.FFTBuffers.append(collections.deque([(0, 0)], self.FFTLen))

        # Run the file creation tool for file handling
        self.file = FileHandling(fileName, self.numberOfChannles)

    # This sets up the daq task and then spins out a process and thread to read from the DAQ
    def startUpdatingChannels(self):
        self.queues = []
        self.queues.append(queue.Queue())
        self.queues.append(queue.Queue())

        # Sets up a process to collect from the daq
        self.queues = []
        for i in range(self.numberOfChannles):
            self.queues.append(multiprocessing.Queue(maxsize=2*self.historyLength))

        self.process = multiprocessing.Process(target=NIDAQ.readFromDaqContinuosly,
              args=(self.numberOfChannles, self.device, self.samplingRate,
                    self.minVal, self.maxVal, self.sampsAtATime, self.queues))
        self.process.start()

        # Starts a thread to read from the DAQ process
        self.DAQThread = Thread(target=self.readIntoBuffersContinuosly)
        self.DAQThread.start()

    # Stops the recording process by killing the thread and the process
    def stopUpdatingChannels(self):
        # The thread must be terminated first or the queue.put waits for more items
        if (self.DAQThread != None):
            self.DAQThread.continueRunning = False
```

```
            time.sleep(0.05)
            self.DAQThread.join()

        # Like the thread, only terminate if it was defined
        if (self.process != None):
            self.process.terminate()
            time.sleep(0.05)
            self.process.join()

        # Close the fileno
        self.file.Close()

    # Must be called called on an individual thread to handle constant updating
    def readIntoBuffersContinuosly(self):
        # This allows the thread to be stopped from the function that called it
        NIDAQThread = threading.currentThread()
        flag = 0
        while getattr(NIDAQThread, "continueRunning", True):

            channelValues = []
            for i in range(self.numberOfChannles):
                daqValue = self.queues[i].get()
                self.channelBuffers[i].append((self.timeElapsed, daqValue))
                channelValues.append(daqValue)
                # Ensures FFT buffer saves some smaller number of samples
                if (flag == self.FFTSampleReduction):
                    self.FFTBuffers[i].append((self.timeElapsed, daqValue))
                    # Ensures all channels are updated before resetting
                    if (i == self.FFTChannels - 1):
                        flag = 0

            self.file.SaveData(self.timeElapsed, channelValues)
            flag += 1
            self.timeElapsed += (1/self.samplingRate)

    # Must be called called on an individual process to handle constant updating
    @staticmethod
    def readFromDaqContinuosly(numChannels, device, samplingRate, minVal, maxVal,
sampsAtATime, queues):
        # Create a new task to perform the reading, this task will die when this method ends
        with nidaqmx.Task() as readTask:
            # Add all of the channels up to the self.number of channels
```

```
        for i in range(numChannels):
            # RSE = reference single ended
            readTask.ai_channels.add_ai_voltage_chan(device + "ai" + str(i),
                max_val=maxVal, min_val=minVal,
                terminal_config=TerminalConfiguration.RSE)

        # This ensures that the DAQ is constantly sampling without prompt
        readTask.timing.cfg_samp_clk_timing(samplingRate,
            sample_mode=AcquisitionType.CONTINUOUS, samps_per_chan=samplingRate)
        readTask.start()

        # Stream reading allows for more elegant acquition at high rates
        reader = AnalogMultiChannelReader(readTask.in_stream)

        # Streamreader requires a numpy array to save values to
        holderArray = numpy.zeros((numChannels, sampsAtATime),
            dtype=numpy.float64)

        # Must be terminated by the parent process
        while (True):
            # Returns the number of samples read (same for each channel)
            # Waits until sampsAtATime number of samples are availabe
            reader.read_many_sample(holderArray,
number_of_samples_per_channel=sampsAtATime)

            # Append read values into the queues for each channel to be read
            # by the parent process
            for i in range(len(queues)):
                for j in range(sampsAtATime):
                    queues[i].put(holderArray[i][j])

# Responsible for displaying and updating the data to graph
class GraphValues(Widget):
    patient_popup = ObjectProperty(None)
    eng_graph = ObjectProperty(None)
    vitals_graph = ObjectProperty(None)
    frequency_graph = ObjectProperty(None)
    xMin = NumericProperty(0)
    xMax = NumericProperty(1)

    # Init must take in *kwargs for some reason. Something to do with inheriting from the Widget
class
```

```
    def __init__(self, NIDevice, **kwargs):
        super(GraphValues, self).__init__(**kwargs)
        self.patientNumber = "01"

        self.NIDevice = NIDevice
        self.DAQSampleRate = 1000
        self.histLen = 4000
        self.firstStart = True

        # DAQSampleRate/FFTSampleReduction = rate at which the fft is resamples. This reduces
computation
        self.FFTSampleReduction = 4

        # All of these channels will run a recording and get saved to file you cannot skip channels
(i.e [0, 1, 3] is not allowed, nor is [1, 2, 3] becase 0 is skipped)
        self.channelsToRecord = [0, 1, 2, 3, 4, 5] # Typically [ENG1, ENG2, ECG, X, Y, Z]
        # These are the channels that will show up on the top ENG plot, must be a subset of
channels to record
        self.engChannelsToPlot = [0, 1]
        # These are the channels that will show up on the middle vitals plots, must be a subset of
channels to record
        self.vitalsChannelsToPlot = [2, 3, 4, 5]
        # These are the channels that will be re-sampled and have FFT run, must be a subset of
channels to record
        self.FFTChannels = [0, 1]

        # Add the plots to the graphs
        self.engPlots = self.addGraphingPlots(self.engChannelsToPlot, self.eng_graph)
        self.vitalsPlots = self.addGraphingPlots(self.vitalsChannelsToPlot, self.vitals_graph)
        self.freqPlots = self.addGraphingPlots(self.FFTChannels, self.frequency_graph)

    # This adds plots to a kivy graph widget
    def addGraphingPlots(self, plotsToAdd, graphToAddTo):
        plots = []
        for ch in plotsToAdd:
            plot = MeshLinePlot(color=self.ColorGenerator(ch))
            plots.append(plot)
            graphToAddTo.add_plot(plot)

        return plots

    # Starts the DAQ and the plotting, reserves the DAQ and the buffers
```

```
    def start(self):
        # Disable the start button to avoid trying to reserve the DAQ again
        self.ids.start_button.disabled = True
        self.ids.stop_button.disabled = False
        self.ids.pause_button.disabled = False

        # Get the patient number for file directory to save to
        if (self.firstStart):
            self.firstStart = False
            self.patient_popup.open()
            self.patient_popup.ids.pButton.bind(on_press=self.startPopup)
        else:
            # This reseres the DAQ and it continusly gathers voltages in the buffers
            self.DAQ.startUpdatingChannels()
            # This gets the graph to update every 0.05 seconds
            Clock.schedule_interval(self.updateGraph, 0.05)
            Clock.schedule_interval(self.updateFFT, 1)
            Clock.schedule_interval(self.syncFile, 5)

    def startPopup(self, *args):
        # Store the input from the user as the patient number
        self.patientNumber = str(self.patient_popup.ids.input.text)

        # The popup should now disappear
        self.patient_popup.dismiss()

        # This instantiation is important, it sets up the number of channels, buffer size and things of
the like
        self.DAQ = NIDAQ(self.NIDevice, numChannels = len(self.channelsToRecord),
                samplingRate = self.DAQSampleRate, histLen = self.histLen,
                FFTSampleReduction = self.FFTSampleReduction,
                FFTChannels = len(self.FFTChannels), fileName = self.patientNumber)

        # This reseres the DAQ and it continusly gathers voltages in the buffers
        self.DAQ.startUpdatingChannels()

        # This gets the graph to update every 0.02 seconds
        Clock.schedule_interval(self.updateGraph, 0.05)
        Clock.schedule_interval(self.updateFFT, 0.5)
        Clock.schedule_interval(self.syncFile, 5)

    def pause(self):
```

```
        pass

    def stop(self):
        # Re-enable the start Button, disable stop button for aesthetics
        self.ids.start_button.disabled = False
        self.ids.stop_button.disabled = True

        # This stops the loop updating the buffer and waits for the thread to finish
        self.DAQ.stopUpdatingChannels()

        # This stops the graph from updating
        Clock.unschedule(self.updateGraph)
        Clock.unschedule(self.updateFFT)
        Clock.unschedule(self.syncFile)

        # The file has been closed so we will ask for a new patient number
        self.firstStart = True

    # dt is update time interval and must be passed to any funciton called from clock
    def updateGraph(self, dt):

        # Update the x boundaries to "follow" the graph
        self.xMin = self.DAQ.channelBuffers[0][0][0]
        self.xMax = self.DAQ.channelBuffers[0][0][0] + (self.histLen/self.DAQSampleRate)

        # Update the points on ENG graph on the top
        for plot, ch in zip(self.engPlots, self.engChannelsToPlot):
            plot.points = self.DAQ.channelBuffers[ch]
        # Update the points on vitals graph in the middle
        for plot, ch in zip(self.vitalsPlots, self.vitalsChannelsToPlot):
            plot.points = self.DAQ.channelBuffers[ch]

    def updateFFT(self, dt):
        #start = time.perf_counter()
        ffts = FFT.FFTFromDEQ(self.DAQ.FFTBuffers,
self.DAQSampleRate/self.FFTSampleReduction)
        #print("FFT Time: " + str(time.perf_counter() - start), flush=True)

        # Update the points on frequency graph on the bottom
        for plot, fft in zip(self.freqPlots, ffts):
            plot.points = fft
```

```
    # Ensures file save info gets pushed to disk
    def syncFile(self, dt):
        self.DAQ.file.FileSync()

    # This just returns a unique color for the first 5 channels, if there is a better way to do this,
please do
    def ColorGenerator(self, ch):
        if (ch == 0):
            return [0, 1, 1, 1]
        elif (ch == 1):
            return [1, 0, 0, 1]
        elif (ch == 2):
            return [0, 1, 0, 1]
        elif (ch == 3):
            return [0, 0, 1, 1]
        elif (ch == 4):
            return [1, 0, 1, 1]
        else:
            return [1, 1, 1, 1]

# This is the main GUI function and must be named after the kivy file. (e.g. DaqApp -> Daq.kv)
class DaqApp(App, BoxLayout):
    def build(self):
        Window.bind(on_request_close=self.onClose)
        self.graphValues = GraphValues("Dev1/")
        return self.graphValues

    def onClose(self, *args):
        self.graphValues.DAQ.stopUpdatingChannels()

if __name__ == '__main__':
    # This must be imported after the main qualifer or it will create a seperate
    # blank window for every sub-process called. Poor implementation from Kivy
    # Documented at https://github.com/kivy/kivy/issues/4744
    from kivy.core.window import Window
    DaqApp().run()
```

# APPENDIX C

# IRB EXEMPTION

UNIVERSITY OF CENTRAL FLORIDA

**Institutional Review Board**
FWA00000351
IRB00001138, IRB00012110
Office of Research
12201 Research Parkway
Orlando, FL 32826-3246

NOT HUMAN RESEARCH DETERMINATION

March 30, 2022

Dear Reza Abdolvand:

On 3/30/2022, the IRB reviewed the following protocol:

| | |
|---|---|
| Type of Review: | Initial Study |
| Title of Study: | Feasibility of detecting neonatal intestinal motility through EEnG |
| Investigators: | Reza Abdolvand<br>Garett Goodale |
| IRB ID: | STUDY00003951 |
| Funding: | None |
| Grant ID: | None |
| Documents Reviewed: | • Data Points<br>• HRP-250-FORM- Request for NHSR-rev1.docx |

The IRB determined that the proposed activity is not research involving human subjects as defined by DHHS and FDA regulations.

IRB review and approval by this organization is not required. This determination applies only to the activities described in the IRB submission and does not apply should any changes be made. If changes are made and there are questions about whether these activities are research involving human in which the organization is engaged, please submit a new request to the IRB for a determination. You can create a modification by clicking **Create Modification / CR** within the study.

If you have any questions, please contact the UCF IRB at 407-823-2901 or irb@ucf.edu. Please include your project title and IRB number in all correspondence with this office.

Sincerely,

Renea Carver
UCF IRB

# REFERENCES


[1] J. D. Z. Chen, B. D. Schirmer and R. W. McCallum, "Measurement of electrical activity of the human small intestine using surface electrodes," *IEEE Transactions on Biomedical Engineering,* vol. 40, pp. 598-602, 1993.

[2] J. Garcia-Casado, J. L. Martinez-de-Juan, J. Silvestre, J. Saiz, J. L. Ponce and G. Prats-Boluda, "Relationship between intestinal motility indexes from internal and surface recordings of electroenterogram," in *2001 Conference Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2001.

[3] V. Zena-Giménez, J. Garcia-Casado, Y. Ye-Lin, E. Garcia-Breijo and G. Prats-Boluda, "A Flexible Multiring Concentric Electrode for Non-Invasive Identification of Intestinal Slow Waves," *Sensors,* vol. 18, 2018.

[4] C. Chourpiliadis and A. Bhardwaj, "Physiology, Respiratory Rate," 21 September 2021. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK537306/. [Accessed 23 Febuary 2022].

[5] B. Chatterjee, L. M. Saini and T. K. Gandhi, "Non-invasive wireless EEG monitor," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017.

[6] G. Prats-Boluda, Y. Ye-Lin, J. M. Bueno Barrachina, E. Senent, R. Rodriguez de Sanabria and J. Garcia-Casado, "Development of a portable wireless system for bipolar concentric ECG recording," *Measurement Science and Technology,* vol. 26, p. 075102, July 2015.

[7] S. Somarajan, N. D. Muszynski, J. D. Olson, L. A. Bradshaw and W. O. Richards, "Magnetoenterography for the Detection of Partial Mesenteric Ischemia.," *The Journal of surgical research,* vol. 239, pp. 31-37, July 2019.

[8] C. E. Bunker, L. P. Johnson and T. S. Nelsen, "Chronic in Situ Studies of the Electrical Activity of the Small Intestine," *Archives of Surgery,* vol. 95, pp. 259-268, August 1967.