Somatosensory Stimulation Apparatus for Rodent Cages

Tim Lieb, Luke DeZellar, Emmy Russell, Alli Abolarin, and Albert Anderson

Abstract -- **An electrode device has been designed to address the problem of phantom limb pain, a neuropathic disorder caused by nerve damage during limb amputation. Before this device can be tested in humans, it must prove successful in animal models. In order to generate functional outcome data from rats, a healthy rat is trained to conditionally respond to a specific somatosensory stimulus. Then, the peripheral nerve is surgically deafferented and the electrode implanted. The device can then be used to apply the same stimulus and the rat's response can be measured. Here we report the development of a device to produce the somatosensory stimulation necessary to train the rats. It consists of a testing enclosure for the rat and motors to facilitate the stimulation that are controlled by a microcontroller. The final design features a plexiglass enclosure with two platforms that vibrate via speaker actuator motors underneath. The motors are connected using a vibration damping foam to ensure isolation. The Arduino Due with corresponding code produces sine waves to drive the motors, and the frequencies are set between 150-350 Hz by the user via the I/O console. Testing has shown that the current device is able to successfully output vibrational frequencies within 3.8% error of those set by the user. Using this device for animal testing will be crucial for determining the efficacy of the electrodes and completing the FDA requirements.**

Index Terms -- **peripheral nerves, nerve damage, electrode therapy, nerve regeneration, somatosensory stimulation, animal testing**

I. Introduction

There are 185,000 amputee surgeries every year in the United States. In these surgeries, patients receive amputations of one or more limbs as well as other body parts. [1] All of these amputations can lead to phantom pain, however amputation of limbs is cited as the most commonly reported source of this pain. Phantom limb pain (PLP) is a neuropathic shooting or burning pain that is caused by the misfiring of the nerves damaged during amputation. [2] Some 42.2-78.8% of amputees suffer from PLP either immediately following surgery or years later. Treatments consist of both pharmacological and non-pharmacological approaches, but due to the lack of knowledge in the mechanism behind the pain, these treatments are often ineffective.[3]

A proposed solution to PLP, which is currently being investigated, is an electronic interface for peripheral nerves. This interface has electrodes that could be implanted around the nerve(s) deafferented during amputation and prevent phantom limb pain, potentially serving as a means to restore the amputee's sense of touch through tactile sensors embedded in a prosthetic limb. For the most part, phantom limb pain is a result of the damaged nerves near amputated body parts misfiring, transmitting pain through the nervous system and causing our brain to register it as pain coming from or near the missing body part. This electrode device could potentially reduce the frequency of or otherwise mitigate the effects of this misfiring, ultimately serving to reduce or relieve phantom limb pain in the patient.

The device is in the animal testing stage of the FDA approval process, Phase-1, and a device that allows for proof of concept testing in rat models is needed. Functionality testing for this type of device in humans is simple since the researcher can ask the subject if he or she felt a sensation in their amputated limb. In rats, however, a nonverbal method is needed in order to determine the device's functionality.

A common method for obtaining functionality data from rats is to collect data and create an "S" shaped histogram of percent correct responses versus stimulation amplitude (Figure 1). This stimulus-response (SR) data can be collected by training a healthy rat to respond in a specific way to a certain stimulus. For example, a rat can be trained to poke its nose in a hole on its right side after receiving a vibrational stimulus on its right foot. Ideally, using a large range of vibrational frequencies should then produce an "S" shaped curve of data points. In order to test this device, the SR curves of healthy rats and rats with that have the device implanted will be compared. Success of this device will be demonstrated by observing similar curves between the normal rat and the surgically altered rat. The similarity indicates that stimulation of the deafferented nerve after electrode therapy feels that same as the stimulus on the normal limb of the rat.

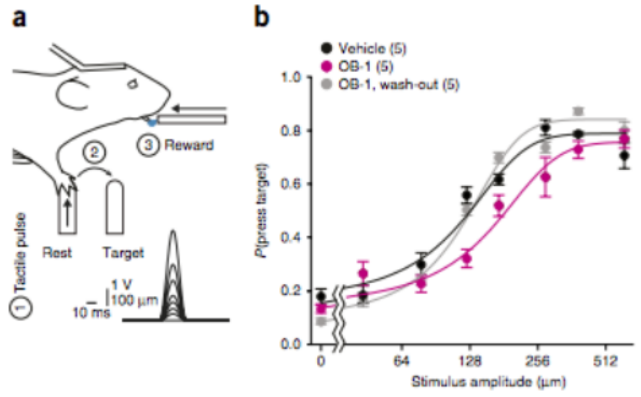


Figure 1. An example of an "S" histogram of percent correct response versus stimulation amplitude in trained rats. (a) The experimental setup for obtaining the data 1). The rat receives a tactile pulse stimulus. 2) Upon detection of the stimulus, the rat presses the target button. 3) The rat is rewarded with water for correctly responding to the stimulus. No reward is given if the response is incorrect. (b) The resulting curves of percent of times the target was pressed vs the stimulus amplitude in this experiment under three different conditions. [4]

II. Existing Technology

Current rat training enclosures on the market include a rat testing cage from Coulbourn (Figure 2). These cages lack floor or wall panels, but accommodate additional training modules. These training modules include a shocking floor and nose poke holes with optic and olfactory stimuli.[5] While a shocking floor could provide stimulus to both of a rat's hindlimbs, it does not allow for isolation of the stimulus onto a single limb. This detail is critical for testing the efficacy of the new electrode device.

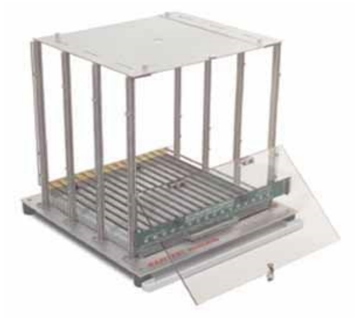


Figure 2: Rat test cage from Coulbourn. This enclosure accommodates different floors and wall panels for different types of rat training. [5]

In order to perform rat testing that will prove the efficacy of the electrode, a device is needed that can stimulate a single rat hindlimb at a range of frequencies from 150-350 Hz. This stimulus must not cause any residual artifacts that may lead the rat to give the correct response without proper functioning of the electrode device. For example, the stimulus must be completely isolated to one hindlimb so that vibration cannot be felt in the opposite, healthy leg at the same time. There must also not be any noise differentiation between stimulus levels so that the rat cannot form associations and respond based on auditory stimulation.

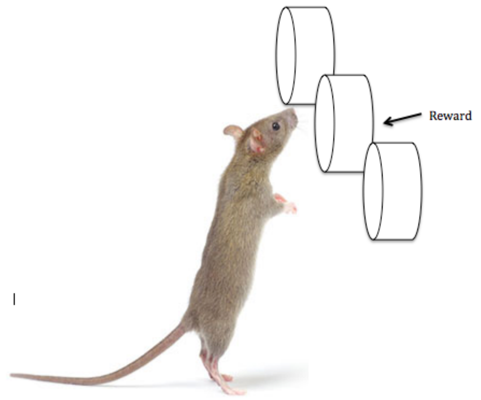


Figure 3: The rearing rat receives its reward by poking its nose through one of the three holes in the cage wall. [6]

In addition to the part of the device that will provide the stimulation, the device must also include an enclosure in which the rats can be trained. The enclosure must allow the user to view the rat during testing. It should restrict non-experimental movement, but not limit the rat's ability to respond to the stimulus. In order to do this, the cage should include three holes at nose-height of a rearing rat so the rat can poke its nose in the appropriate hole and receive its reward (Figure 3). There are not any other devices on the market that combine these design elements in a satisfactory way, thus, the creation of a new device is required.

III. Background

In order to elicit a response from the rat, the somatosensory system must transmit the stimulus from the peripheral nervous system neurons to the brain, where a response action can be determined. The somatosensory system has different receptor cells for each modality of stimulation. Both humans and rats have mechanoreceptors, thermoreceptors, proprioceptors, pain receptors, and chemoreceptors. Mechanoreceptors will be the focus of this device as low mechanical vibration is used as the stimulus. In the skin, humans and rats have three main cell types that sense vibration (Figure 4). Merkel cells are closest to the surface of the skin and sense low frequency vibrations (5-15 Hz). Slightly deeper in the skin are tactile corpuscles, which sense frequencies of 10-50 Hz.[7]

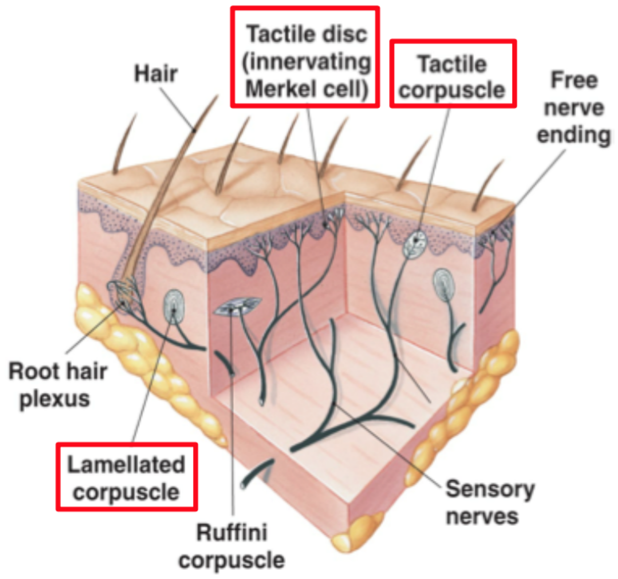


Figure 4: Mechanoreceptors in the skin. Merkel cells and tactile corpuscles sense low frequency vibrations on the surface of the skin while lamellar corpuscles sense higher frequency vibrations and they transfer the information to the brain via the somatosensory system.[7] Red boxes added by authors to emphasize key cells.

Lamellar corpuscles can sense higher frequencies, with optimal sensing at 250 Hz. The Lamellar corpuscles will be the main target for the vibrations in this project. These cells often cannot be targeted without propagating the vibration throughout the rat's body. Due to their anatomy and physiology, rat bodies tend to attenuate vibrational frequencies of 31-50 Hz.[8] For this project, only the hindlimb is stimulated with low amplitude vibrations, and so vibration of the rat's body should not be an issue. Training with stimulation of other body parts may confound the results of the efficacy of the electrode.

The rats that will be used in this experiment are adult Lewis breed that range in size from 250-350 g. These rats are commonly used as animal models due to their physiological similarities to humans as well as their mild temperament.

IV. Fabrication/Development

A. Materials and Methods

Plexiglass was selected for the enclosure to allow for viewing of the rats during testing. The six pieces of the cage were laser cut with interlocking tabs and put together with acrylic glue. The enclosure measures 0.25 x 0.28 x 0.3 m (10 x 11 x 12 in). The floor piece included two cut outs for the actuators, which provided the vibrations with 0.08 m (3 in) left underneath it to hold the electronics. The highly damping foam is used for the connection between the cage and the actuator to isolate the vibrations and prevent propagation throughout the cage. 3D renderings of the enclosure are depicted in Figure 5 and Figure 6.

The electrical components include surface transducers, an Arduino Due microcontroller, are wires. The transducers have the actuator surface replaced with a plexiglass platform that protrudes through the floor of the enclosure. The microcontroller is responsible for providing the output waveform at the desired frequency.

Vibration-damping foam was molded into two cylinders with 2 in diameter and 3 in height. The vibrational motors were pressed into the top of the cylinders and set out to air dry for 24 hours. These foam cylinders with motors were then placed on a piece of base plexiglass directly under the rectangular cut-outs in the floor of the cage.

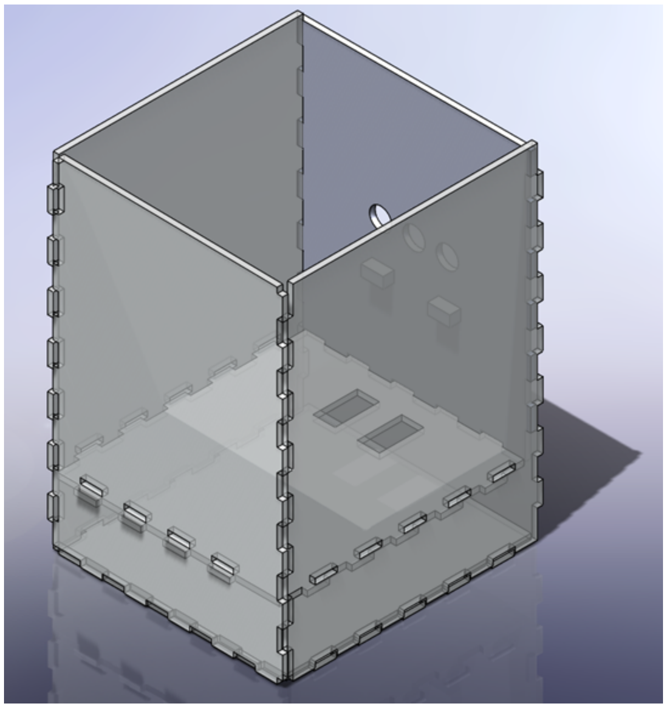


Figure 5: 3D rendering of the rodent cage using SolidWorks consisting of 6 pieces with interlocking tabs.

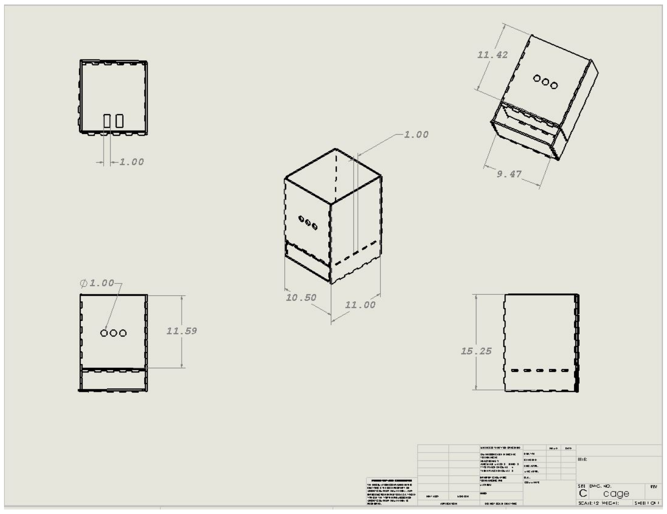


Figure 6: Dimensioned drawing of the rodent cage with measurements in inches.

B. Final Prototype

The final prototype begins with two surface transducers. The original top of the transducer has been replaced with a platform that fits into rectangular cutouts in the floor of the enclosure. The modified surface transducer can be seen in Figure 7**.** Each of these transducers are placed in a pillar of damping foam to increase the isolation of the resulting vibration

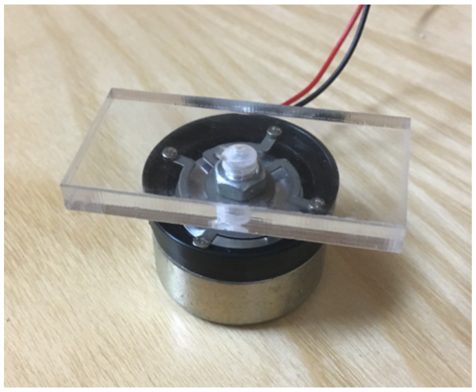


Figure 7: Surface transducer with attached platform

The system diagram (Figure 8) shows the circuit/electronic connections. First, the system is controlled using an Arduino Due microcontroller (Figure 9) as well as the corresponding code. The Due features two output ports with 12-bits resolution and digital to analog conversion (DAC) built-in. These specs will allow the Arduino to create a precise sine wave. The sine wave will have a max 2.75 V peak-to-peak signal, as limited by the microcontroller. A frequency range of 150-350 Hz will be used for the sine wave to provide a graded stimulus.

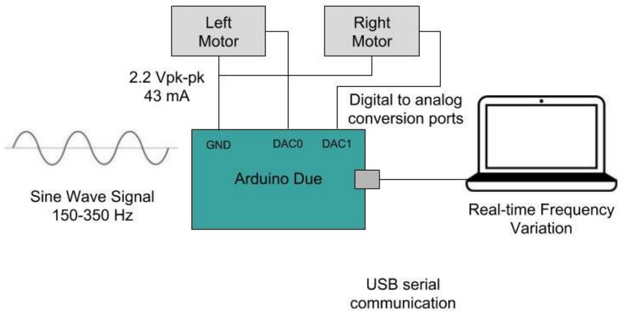


Figure 8: System diagram of electronics

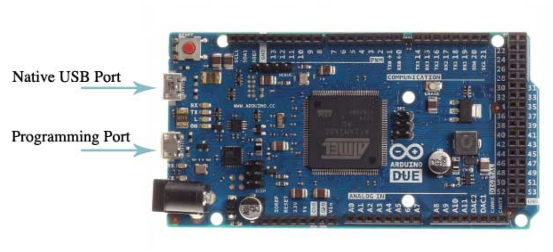


Figure 9: Arduino Due microcontroller showing the two USB ports for communication. [9]

The frequency range of the sine waves was determined based on the surface transducer's ability to produce certain vibration amplitudes at various frequencies. At very low frequencies, the transducer produced very low amplitude vibrations, which could not be felt when the motors were driven directly by the Arduino. However, at higher frequencies, greater than 150 Hz, the vibrational amplitude was sufficient enough to be felt without amplification. In addition, the surface transducers behave more linearly at higher frequencies, and this provides the device with more consistent vibrations.

Therefore, the motors were able to be driven by the DAC ports of the Arduino directly. Even though the Arduino actually only produced a 2.2 V pk-pk and a 43mA signal, the surface transducers were still able to provide sufficient amplitude. Driving the motors from the Arduino directly allowed the team to greatly simplify the electronics design by eliminating the need for an amplification system as well as an external power source. Simplifying the design helps to eliminate unnecessary sources of error.

The stimulus is applied as desired using a laptop. The user interface also allows for real-time frequency variation. The user can set the operating frequencies for both limbs, set the stimulus duration and then apply the stimulus. This interface allows for easy manipulation of the device while requiring no knowledge of the electronics or code. More specifics code and user interface follows in the software section.

C. Software

The software for this project is split up into 3 different files: a .ino Arduino file, a .cpp C++ file, and a .h Header file. The Arduino file contains the code for the user interface as well as error handling and the C++/Header files contain the code for creating and writing the sine waves.

The Arduino file contains code entirely written by this team. The user interface in implemented using the serial monitor console, which is built into Arduino. The code communicates with the console by using serial communication along with a variety of serial commands built into Arduino software. The team determined that optimal communication/function of the code occurred when the analogReadResolution() and the analogWriteResolution() where set to 12 and the Serial.begin() was set to 9600, which is the baud rate.

Another important consideration for the serial communication was which USB port on the Arduino was best to use. The Arduino Due contains a Native USB port and a Programming port (both ports are labeled in Figure 9). The Native port allows for higher speed serial communication but the Programming port is better for uploading code to the Arduino. Unlike other Arduinos, the flash memory on the Arduino Due must be erased before new code can be programmed to the board. The Programming port does what is known as a “hard erase” which is more reliable than the “soft erase” of the Native port. Since the Programming port also provides decent serials speeds (up to 115200 bits per second), the team decided to use the Programming port for the device’s communication.

The full Arduino code can be found in **APPENDIX III**, but the software block diagram in Figure 10 provides a brief overview of the code. Essentially, the code needs the left frequency, the right frequency, and the stimulus duration from the user. The stimulus duration can be set as one global stimulus that is used for each iteration, or the duration can be set for each stimulus individually. Once all three parameters are set with valid inputs, the code can apply the stimulus by calling the functions outlined in the C++/Header code.

The C++/Header files were based on free source code found online which was then altered to fit the needs of this project (code found at source [10]). These files create a SineWave object that contains different methods and parameters than can be used and manipulated. The Header file defines the constants and the methods and the C++ file contains the actual method code. Some examples of the variability provided by the methods include playing 1 or 2 sine waves, setting a duration or not, and changing the output pins of the Arduino.

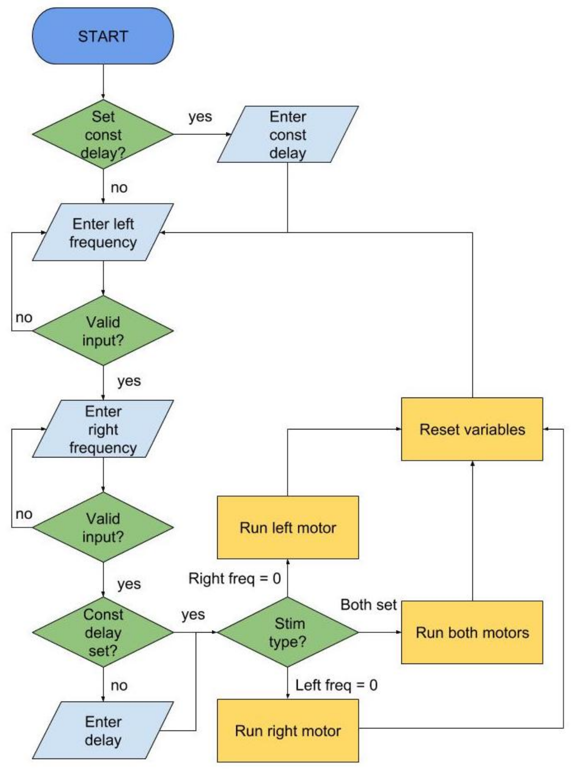


Figure 10: Software block diagram of the Arduino code showing the user input and error handling.

The values for outputting the sine wave are computed in real-time by the C++ code. The sine wave has the form A*sin(omega*sampling frequency). Then each value is created using filter coefficients and 3 registers to compute and shift the values. This allows each value of the sine wave to be computed and written in real time. These algorithms are based on the source code algorithms as well. This part of the code exists behind the scenes and is likely unknown to the user. The full code files for the C++/Header files can be found in **APPENDIX IV** and **APPENDIX V**.

One key feature that had to be changed from the source code was stopping the sine waves. The source code had a stopTone() function that was supposed to stop the sine wave, but it did not work correctly since the code would get stuck in a loop in the compute sine wave functions. The team implemented a counter in order to call stopTone() after the correct number of iterations. The code would update and print the next value of the sine value until it reached the delayCount, and this count equaled the stimulus duration / sampling frequency. Therefore, the stopTone() would correctly get called from the compute methods and the sine waves would stop at the correct times.

V. Testing

The testing was split into two parts: bench testing and animal testing. Bench testing verifies that the code worked as intended and the vibration stimulus was applied at the correct time to the correct limbs. Each motor should be driven at the programmed frequency regardless of the frequency of the other motor.

A. Bench Testing

The main part of our testing was verifying that the transducers were operating at the correct frequency and amplitude. This is critical since the sine waves are created from the Arduino using the code the team wrote. This verification consists of frequency testing at two different points. Both the left and right motors were tested simultaneously since they are produced by different DAC ports and they will be used together in real applications.

The first step in the testing was sending the output from the Arduino directly into the oscilloscope. This was done to analyze the output waveform and verify the frequency of the sine wave. Plots of voltage versus time were created on the oscilloscope, and the y-axis cursors were be set to calculate the frequency of the signal. 3 data points were taken at each 150, 200, 250, 300, and 350 Hz, and the mean and standard deviations were calculated. Then, the percent error between the programmed frequency and the mean frequency was computed. This process was intended to eliminate random outliers.

Then, the team verified that the motors were operating at the correct frequency. This step involved using an accelerometer to measure the mechanical vibrations that can be observed on the oscilloscope. This step was again done for the 5 frequencies mentioned above, and the mean and standard deviation were calculated. Then, the percent error between the programmed frequency and the mean frequency was computed. The team goal was to verify that the vibration frequencies are within +/- 0.5 Hz of the target frequency.

Lastly, the team used the same accelerometer to try and measure any noise from the vibration source on other parts of the cage. The accelerometer was placed on other parts of the cage while the motors were on, and the corresponding electrical signal was sent to the oscilloscope for analysis, concluding our testing of the functionality of the device.

B. Animal Testing

After the functionality of the device is verified, the team wants to test the device with live rats. Before any testing is conducted, the rats will be trained to rear with their hind limbs on the vibrating platform and respond on the left or right side of the device depending on which side received the stronger stimulation. Once trained, healthy rats that have not received the electrode stimulation will be tested with the device to evaluate their response and sensitivity. Next, those same rats will have their sciatic nerve deafferented and receive the electrode therapy. Finally, those rats will receive the same testing and their responses will be recorded. This data can then be compared to that of the healthy rats in order to determine the success of the electrode device. The exact animal testing protocols will need to be discussed with the client in the future.

VI. Results

A. Bench Testing

The bench testing produced both qualitative and quantitative results. The qualitative results that the team found were that once the correct specifications have been input by the user, there is no time delay in application of the stimulus by the transducers. Also, it was verified that the stimulus only lasted for the specified amount of time. The team also verified that the output in the oscilloscope had the correct sine wave waveform. Then, the team used quantitative data from the oscilloscope to evaluate the output frequency of the surface transducers.

The data for all trials related to frequency testing can be found in **APPENDIX I** under Tables 1 and 2. The results of the bench testing are displayed below in Figure 11. The data points shown in this figure represent the mean percent error (the percent error between the mean frequency and the intended frequency value).

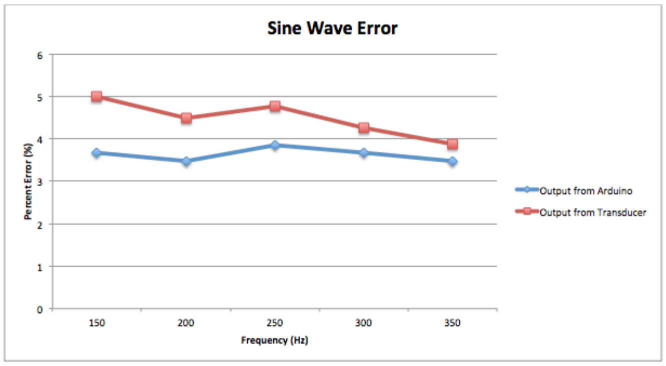


Figure 11: Comparison of the percent error of the input and output frequencies for both the output from the Arduino and the output from the accelerometer. 3 data points were taken every 50 Hz from 150-350 Hz for each trial and the means were plotted above.

It can be seen in Figure 11 that the signal from the Arduino Due (shown as the blue data points) had a consistent percent error between 3.5-3.8%. The percent error for frequency of the physical vibration begins around 5% then returns to a value closer to the percent error of the Arduino Due signal as the input frequency increases. All the samples taken directly from the Arduino had an amplitude around 2.2 V and those taken from the oscilloscope had an amplitude around 680 mV. **APPENDIX II** contains some samples images that were taking from the oscilloscope during testing. The first figure shows the code running two sine waves a different frequencies, the second shows two sine wave running at the same frequency (used to gather blue data points above), and the third shows the output from the accelerometer (used to gather red data points above).

Lastly, the team tried to measure any resulting vibrations and/or noise in the rest of the enclosure from the motors using the accelerometer. After placing the accelerometer on different parts of the floor and walls, no significant data was produced. The oscilloscope output looked like a baseline measurement. This means that either the vibrations are successfully being isolated to each limb, or the low sensitivity and small output voltages of the accelerometer could not capture the small vibrational signals.

B. Animal Testing

The team did not get to perform animal testing at this time, but this testing will also produce both qualitative and quantitative results. First, the qualitative results will be verifying that rats respond to the desired stimulus in the correct way. Without this verification, the data from rat testing will not be reliable.

Next, the team will collect quantitative data. We will collect a variety of data points and perform statistical analysis to compare the rats before and after their surgery. The primary result of interest will be a stimulus-response histogram. This will compare the percent of correct responses with the level of stimulus provided. The histogram will not only depict how the rats respond to the stimulus but also make it easy to compare the healthy and postoperative rats' behavior.

VII. Discussion

In this section, the result will be broken down to confirm that the device functions properly as well as address any limitations and shortcomings of the device. The experimental data for the signal directly from the Arduino showed a percent error that stayed within a constant range of 3.5-3.8%. While the output frequencies from the Arduino were consistent, they were not quite as precise as the team would have liked. Since the percent error is pretty consistent, the team believes that this can possibly be corrected by tweaking the algorithm code. It would take more testing to determine if that would work, but there are many parameters such as the amplitude, offset, and sampling frequency that can be altered to observe the frequency response.

The percent error of the accelerometer output at 150 Hz was roughly 5%, but the percent error at 350 Hz was 3.8%. This percent error at 350 Hz is much closer to that of the Arduino signal, showing that the percent error decreased as the frequency was increased. There are two explanations for this change in percent error. The first is that the surface transducer has a worse frequency response at the beginning of the frequency range, compared to that of the upper limit of the frequency range. This means that the transducer is being driven at too low of a frequency, and thus is not able to translate that electronic signal to a physical output. The second explanation is that the accelerometer, not the surface transducer, is the component that has the poor frequency response. This would mean that the transducer is outputting the correct physical vibration, but the accelerometer is not able to detect that vibration accordingly.

After further examination, it is believed that the surface transducer has a poor frequency response. This was based off of the data sheets for both of these components. The accelerometer has a corner frequency of 50 Hz and a bandwidth of greater than 500 Hz. This bandwidth range comfortably contains our frequency range. The data sheet of the surface transducer also indicates that the device has a passband beginning at 507 Hz, which is outside of the intended frequency range.

The team was not able to perform animal testing during this design period. These results would not have provided information about the functionality of the device but rather its efficacy. Even if the device functioned as intended, there is no guarantee that the rats would respond in the predicted way. In addition, if the results did not indicate significant differences in the rats before and after surgery, then other options to differentiate the stimulus would need to be considered. For example, the frequency range or amplitude would need to be changed moving forward.

VIII. Conclusion/Future Development

The team designed a somatosensory stimulation device that can be used to verify the efficacy of our client's electrode interface for peripheral nerve therapy. Current rat training enclosures do not offer stimulation that can be isolated to the individual hindlimbs. The device consists of an enclosure that is able to apply a graded stimulus to the two hindlimbs individually using an Arduino microcontroller. The final prototype includes clear plexiglass walls and floors, and the stimulus is provided by two platforms that vibrate via speaker actuators using the Arduino. A rearing rat will stand on these platforms during training and testing. The motors are secured using a vibration damping foam.

The team's bench testing demonstrated that the device operates correctly. It was confirmed that the Arduino inputs a sine wave at a frequency within 3.5-3.8% of the mechanical vibrations and the platform also operate at a frequency within 3.8-5% error of the desired frequency. By adjusting the sine wave algorithms and/or shifting the frequency range, the team believes the devices could produce even more precise signals. The team also wants to perform animal testing to demonstrate that the device can be used for verifying the efficacy of the electrodes. By creating this device to evaluate the effectiveness of the electrode therapy, the research can continue to move forward and lead to vast improvements for nerve regeneration and phantom limb pain in humans.

Acknowledgements

The authors would like to thank our client Dr. Aaron Dingle, our advisor Professor Mitch Tyler, Dr. Aaron Suminski, and Dr. Amit Nimunkar for their help and guidance.

References

[1] "Limb Prosthetics Services and Devices", *Semantic Scholar*, 2017. [Online]. Available: https://pdfs.semanticscholar.org/c3ae/f3563844e2e2835411fcbc2b0fe3091ac30b.pdf. [Accessed: 20- Sep- 2017].

[2] "Neuropathic Pain Management", *WebMD*, 2017. [Online]. Available: https://www.webmd.com/pain-management/guide/neuropathic-pain#1. [Accessed: 07-Oct- 2017].

[3] B. Subedi and G. Grossberg, "Phantom Limb Pain: Mechanisms and Treatment Approaches", *Pain Research and Treatment*, vol. 2011, pp. 1-8, 2011.

[4] C. Wetzel, S. Pifferi, C. Picci, C. Gök, D. Hoffmann, K. Bali, A. Lampe, L. Lapatsina, R. Fleischer, E. Smith, V. Bégay, M. Moroni, L. Estebanez, J. Kühnemund, J. Walcher, E. Specker, M. Neuenschwander, J. von Kries, V. Haucke, R. Kuner, J. Poulet, J. Schmoranzer, K. Poole and G. Lewin, "Small-molecule inhibition of STOML3 oligomerization reverses pathological mechanical hypersensitivity", *Nature Neuroscience*, vol. 20, no. 2, pp. 209-218, 2016.

[5] R. TEST CAGE - RAT - INCLUDES INFUSION AND STIMULATION LID, "TEST CAGE - RAT - INCLUDES INFUSION AND STIMULATION LID, REQUIRES FLOOR PURCHASED SEPARATELY", 2017. [Online]. Available: http://www.coulbourn.com/product_p/h10-11r-tc.htm. [Accessed: 26- Sep- 2017].

[6] "Automatic Rat Behavior Recognition", *Noldus.com*, 2017. [Online]. Available: http://www.noldus.com/EthoVision-XT/Rat-Behavior-Recognition. [Accessed: 08- Oct- 2017].

[7] A. Basbaum, *The Senses: A Comprehensive Reference*. Oxford, U.K.: Elsevier, 2008, pp. 33-38.

[8] K. Rabey, Y. Li, J. Norton, R. Reynolds and D. Schmitt, "Vibrating Frequency Thresholds in Mice and Rats: Implications for the Effects of Vibrations on Animal Health", *Annals of Biomedical Engineering*, vol. 43, no. 8, pp. 1957-1964, 2014.

[9] "Arduino Due Documentation." *Arduino*. 2018. [Online]. Available: https://store.arduino.cc/usa/arduino-due

[9] C. Masenas. "Sinewave Library for the Arduino Due". *GitHub*. 6 Nov 2015. [Online]. Available: https://github.com/cmasenas/SineWaveDue

APPENDIX I

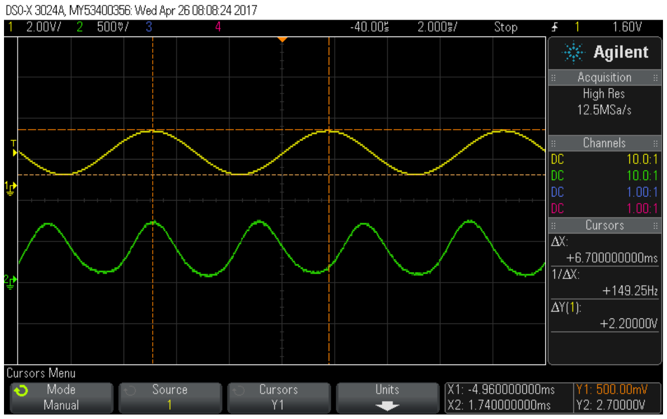
Table 1: Experimental data for frequency testing of the Arduino Due signal

Due		Intended frequency (Hz)				
		150.00	200.00	250.00	300.00	350.00
Trials	1.00	144.20	194.20	241.05	289.16	338.45
	2.00	142.38	193.78	240.65	288.85	337.50
	3.00	146.80	191.18	239.45	289.06	337.56
	Average Freq	144.46	193.05	240.38	289.02	337.84
	Std. Dev	1.81	1.34	0.68	0.13	0.43

Table 2: Experimental data for frequency testing of the physical output from the transducers

Accelerometer		Intended frequency (Hz)				
		150.00	200.00	250.00	300.00	350.00
Trials	1.00	143.28	190.13	237.78	288.36	335.47
	2.00	142.43	191.03	237.88	287.05	336.85
	3.00	141.78	192.00	238.65	286.20	337.09
	Average Freq	142.50	191.05	238.10	287.20	336.47
	Std. Dev	0.61	0.76	0.39	0.89	0.71

APPENDIX II

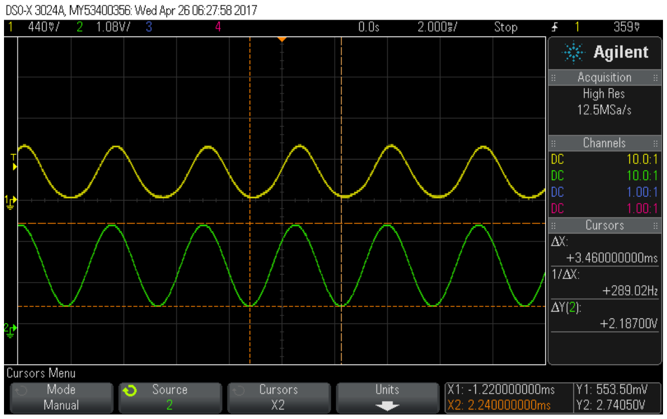

DSO-X 3024A, MY53400356: Wed Apr 26 08:08:24 2017

Acquisition: High Res 12.5MSa/s

ΔX: +6.700000000ms
1/ΔX: +149.25Hz
ΔY(1): +2.20000V

X1: -4.960000000ms X2: 1.740000000ms Y1: 500.00mV Y2: 2.70000V

Figure 1: This shows the oscilloscope output with one DAC port outputting 150 Hz (yellow) and the other outputting 250 Hz (green).

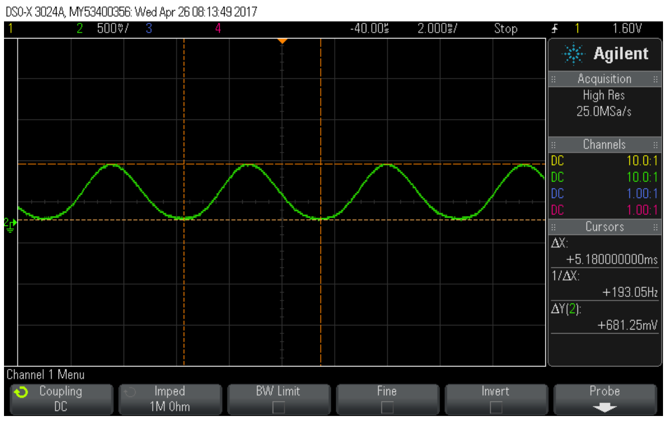

DSO-X 3024A, MY53400356: Wed Apr 26 08:27:58 2017

Acquisition: High Res 12.5MSa/s

ΔX: +3.460000000ms
1/ΔX: +289.02Hz
ΔY(2): +2.18700V

X1: -1.220000000ms X2: 2.240000000ms Y1: 553.50mV Y2: 2.74050V

Figure 2: This shows the oscilloscope output with both DAC ports outputting at 300 Hz.


DSO-X 3024A, MY53400356: Wed Apr 26 08:13:49 2017

Acquisition: High Res 25.0MSa/s

ΔX: +5.180000000ms
1/ΔX: +193.05Hz
ΔY(2): +681.25mV

Figure 3: This shows the oscilloscope output from the accelerometer when driving the motor at 200 Hz.

APPENDIX III
Arduino Code

```
/*
    SineWave.ino
    UW Madison BME Design: Team Hindlimb Stim.
    Written by Timothy Lieb- April 25, 2018.
*/


#include <DueTimer.h>
#include <SineWave.h>
int lowerFreqBound = 150;
int upperFreqBound = 350;
int leftFreq = 0;
int rightFreq = 0;
bool setDelay = false;
bool globalDelay = false;
int delayTime = 0;


void setup() {
  analogReadResolution(12);
  analogWriteResolution(12);
  Serial.begin(9600);
  while (!Serial);
  Serial.println("<Arduino is ready>");
}


void loop() {
  if (setDelay == false) {
    createDelay();
  }
  setLeftFreq();
  setRightFreq();
  applyStimulus();
}


void createDelay() {
  bool isSet = false;
  char answer;
  Serial.println("Would you like to set a constant stimulus duration (y/n): ");
  while (!isSet) {
    if (Serial.available() > 0) {
      answer = Serial.read();
      isSet = true;
    }
  }
  if (answer == 'y') {
    isSet = false;
    Serial.println("Enter constant stimulus duration: ");
    while (!isSet) {
      if (Serial.available() > 0) {
        delayTime = Serial.parseInt();
        globalDelay = true;
        Serial.print("A constant stimulus duration of ");
        Serial.print(delayTime);
        Serial.println(" seconds has been set.");
        isSet = true;
      }
    }
  }
  else {
    Serial.println("The duration will be set for each stimulus individually.");
  }
  setDelay = true;
}


void setLeftFreq() {
  bool isSet = false;
  int input = 0;
  Serial.println("Enter left frequency: ");
  while (!isSet) {
    if (Serial.available() > 0) {
      input = Serial.parseInt();
      if (lowerFreqBound <= input && input <= upperFreqBound) {
        leftFreq = input;
        Serial.print("Left frequency set to ");
        Serial.print(input);
        Serial.println(" Hz.");
        isSet = true;
      }
      else if (input == 0) {
        //left motor off
        leftFreq = 0;
        Serial.println("Left frequency set to 0 Hz.");
        isSet = true;
      }
      else {
        Serial.print("Frequency must be between ");
        Serial.print(lowerFreqBound);
        Serial.print(" Hz and ");
        Serial.print(upperFreqBound);
        Serial.println(" Hz.");
        Serial.println("Enter left frequency: ");
      }
    }
  }
}


void setRightFreq() {
  bool isSet = false;
  int input = 0;
  Serial.println("Enter right frequency: ");
  while (!isSet) {
    if (Serial.available() > 0) {
      input = Serial.parseInt();
      if (lowerFreqBound <= input && input <= upperFreqBound) {
        rightFreq = input;
        Serial.print("Right frequency set to ");
        Serial.print(input);
        Serial.println(" Hz.");
        isSet = true;
      }
      else if (input == 0) {
        //right motor off
        rightFreq = 0;
        Serial.println("Right frequency set to 0 Hz.");
        isSet = true;
      }
      else {
        Serial.print("Frequency must be between ");
        Serial.print(lowerFreqBound);
        Serial.print(" Hz and ");
        Serial.print(upperFreqBound);
        Serial.println(" Hz.");
        Serial.println("Enter right frequency: ");
      }
    }
  }
}


void applyStimulus() {
  if (globalDelay == false) {
    bool isSet = false;
    Serial.println("Enter the stimulus duration: ");
    while (!isSet) {
      if (Serial.available() > 0) {
        delayTime = Serial.parseInt();
        Serial.print("A stimulus duration of ");
        Serial.print(delayTime);
        Serial.println(" seconds has been set.");
        isSet = true;
      }
    }
  }


  bool isApplied = false;
  Serial.println("Press any key to apply stimulus: ");
  while (!isApplied) {
    if (Serial.available() > 0) {
      char clearBuffer = Serial.read();
      if (leftFreq == 0 && rightFreq == 0) {
        Serial.println("No stimulus applied, both frequencies set to 0.");
        isApplied = true;
      }
      else if (rightFreq == 0) {
        sw.setPin(0); //DAC0 for left motor
        sw.setSamplingFreq(100/12);
        sw.playTone(leftFreq, delayTime);
        isApplied = true;
        resetParams();
      }
      else if (leftFreq == 0) {
        sw.setPin(1); //DAC1 for right motor
        sw.setSamplingFreq(100/12);
        sw.playTone(rightFreq, delayTime);
        isApplied = true;
        resetParams();
      }
      else {
        sw.setSamplingFreq(100/6);
        sw.playTone2(leftFreq, rightFreq, delayTime);
        isApplied = true;
        resetParams();
      }
    }
  }
}


void resetParams() {
  Serial.println("Stimulus applied.");
  rightFreq = 0;
  leftFreq = 0;
}
```

APPENDIX IV
Header Code

```
/*
 *              SineWave.h
 *      UW Madison BME Design: Team Hindlimb Stim.
 *      Updated by Timothy Lieb- April 25, 2018.
 *      Sine wave algorithms by C. Masenas, November 8, 2015.
 *
 */


#ifndef SineWave_h
#define SineWave_h


// use the DueTimer library for timing
#include <DueTimer.h>
// DAC0 for left motor
#define OUTPIN0 DAC0
// DAC1 for right motor
#define OUTPIN1 DAC1


class SineWave{
    private:
        const float pi = 3.14159;
                // amplitude of sine signal
                const float A = 2000;
                // offset for sine wave
                const float D = 2000;
                // set default output pin
                int pin = OUTPIN0;
                int count = 0;
                int delayCount = 0;
                // set default sampling frequency in microseconds
                float SF = (100/6)/1000000.0;
                // c1 = first filter coefficient, c1b used for second tone
                float ca, cb;
                // filter registers, updated from interrupt so must be volatile
                volatile float a[3], b[3];


    public:
                // changes sampling frequency
                void setSamplingFreq(float interval);
                // changes the output to DAC0 or DAC1
                void setPin(int pinNew);
                // play one tone for a specified time
                void playTone(float freq, int duration);
                // play one tone
                void playTone(float freq);
                // play two tones for a specified time
                void playTone2(float freq1, float freq2, int duration);
                // play two tones
                void playTone2(float freq1, float freq2);
                // stop playing a tone
                void stopTone(void);
                // computes samples for one tone
                void compute(void);
                // computes samples for two tones
                void compute2(void);
};
```

```
// sw is instantiated in cpp file
extern SineWave sw;
#endif
```

APPENDIX V
C++ Code

```
/*
 *              SineWave.cpp
 *      UW Madison BME Design: Team Hindlimb Stim.
 *      Updated by Timothy Lieb- April 25, 2018.
 *      Sine wave algorithms by C. Masenas, November 8, 2015.
 *
 */


#include <SineWave.h>
// uses DueTimer library for interrupt timing
#include <DueTimer.h>


extern "C" {
   void external_compute(void);
   void external_compute2(void);
}
void SineWave::setSamplingFreq(float T){
   SF = T/1000000.0 ; //change the sampling frequency
}


void SineWave::setPin(int pinNew){
   if (pinNew == 0) {
              pin = OUTPIN0;
   }
   else {
              pin = OUTPIN1;
   }
}


void SineWave::playTone(float freq){
   // angular frequency in radians/second
   float omega = 2.0*pi*freq;
   // (omega * sampling frequency) squared
   float wTsq = SF*SF*omega*omega;
   // coefficient of first filter term
   ca = (8.0 - 2.0*wTsq)/(4.0+wTsq);
   // initialize filter coefficients
   a[0] = 0.0;
   a[1] = A*sin(omega*SF);
   a[2] = 0.0;


   Timer1.attachInterrupt(external_compute);
   Timer1.start(SF*1000000);
}


void SineWave::playTone2(float freq, float freq2){
   float omega = 2.0*pi*freq;
   float omega2 = 2.0*pi*freq2;
   float wTsq = SF*SF*omega*omega;
   float wTsq2 = SF*SF*omega2*omega2;
   ca = (8.0 - 2.0*wTsq)/(4.0+wTsq);
   cb = (8.0 - 2.0*wTsq2)/(4.0+wTsq2);
   a[0] = 0.0;
   a[1] = A*sin(omega*SF);
   a[2] = 0.0;
   b[0] = 0.0;
   b[1] = A*sin(omega2*SF);
   b[2] = 0.0;

   Timer1.attachInterrupt(external_compute2);
   Timer1.start(SF*1000000);
}

void SineWave::playTone(float freq, int duration){

   delayCount = duration / (SF);
   playTone(freq);


   //delay(duration);

   //stopTone();
   //Timer1.stop();
```

```
    //Timer1.detachInterrupt();
}


void SineWave::playTone2(float freq, float freq2, int duration){

    delayCount = duration / (SF*2);
    playTone2(freq, freq2);

    //delay(duration);

    //stopTone();
    //Timer1.stop();
    //Timer1.detachInterrupt();
}


void SineWave::stopTone(void){

    Timer1.stop();

    Timer1.detachInterrupt();
}

void SineWave::compute(void){
    // compute the sample
    a[2] = ca*a[1] - a[0];
    // shift the registers in preparation for the next cycle
    a[0] = a[1] ;
    a[1] = a[2] ;
    // write to DAC
    analogWrite(pin, a[2]+D);


    //Added because calling stopTone() was not working correctly
    if (count != delayCount){
        count ++;
    }
    else {
                sw.stopTone();
    }
}


void SineWave::compute2(void){
    a[2] = ca*a[1] - a[0];
    a[0] = a[1] ;
    a[1] = a[2] ;
    b[2] = cb*b[1] - b[0];
    b[0] = b[1] ;
    b[1] = b[2] ;

    sw.setPin(1);
    analogWrite(pin, a[2]+D);

    sw.setPin(0);
    analogWrite(pin, b[2]+D);


    //Added because calling stopTone() was not working correctly
    if (count != delayCount){
    if (count != delayCount){
        count ++;
    }
    else {
                sw.stopTone();
    }
}

// instantiate the SineWave sw instance here
SineWave sw;


void external_compute(void){
    sw.compute();
}


void external_compute2(void){
    sw.compute2();
}
```