8-2024

Methods for Improving Low Frequency Selection and Electrode Firing Order in Cochlear Implants

James H. Keen Jr.
University of New Orleans, jhkeen1@uno.edu

Methods for Improving Low Frequency Selection and Electrode Firing Order in Cochlear Implants

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Applied Physics

By
James Howard Keen, Jr.
B.S. University of New Orleans, 2020
August 2024

Table of Contents

List of Figures

List of Tables

Abstract

The Advanced Bionics cochlear implant devices allocate static frequency bins to electrode channels based on the natural tonotopic organization of the cochlea. These frequency bins are wide and limited, especially in the lower range. Considering that the fundamental tones of the human voice and the primary melodic tones in music are in this lower range, it is important to have accurate representation of this frequency content. Here, adjustments to the frequency bin allocation algorithm used in the crowdsourced CI Hackathon code are made to allow a more accurate representation of the original signal. First, the frequency bins allocated to each channel will be overlapped to allow the lower channels to utilize the parabolic fit method of frequency estimation. Second, a higher order Fourier transform and peak finding algorithm will determine the most important frequencies present, then an adjusted electrode firing order will be found to avoid interference in electrode signals.

Keywords: cochlear implant; Continuous Interleaved Sampling; Fourier analysis; music on cochlear implants

Chapter 1. Introduction

Many individuals at some point in their lives experience hearing loss, due to injury or disease. This loss can occur in one or both ears, and in many degrees of severity. One of the early technologies used to combat hearing loss was the ear trumpet, first used in the 16th Century to amplify incoming sound for the user. Over the next few centuries more advances were made, and in the 20th Century the ear trumpet evolved into the modern hearing aid, acting as a simple amplifier for incoming sounds. These devices work well when there is simple loss of sensitivity to sound in a working cochlea.

In many individuals, injury, disease, or even reactions to certain medications can cause irreparable damge to the cochlea itself. In this case, a simple hearing aid cannot allow the user to perceive sound anymore. The cochlear implant, a device which directly stimulates the nerves of the inner cochlea was developed and has been consitently improved upon for the last few decades.

a. Statement of the Problem

The cochlear implant is quite poor in terms of sound quality. Users have reported that understanding human speech can be difficult and taxing, especially in noisy environments, and that the enjoyment of music is even more elusive.

The limitations of the cochlear implant are predominantly due to the physical size of the device. The cochlea itself is a tiny organ, and there can only be a certain number of stimulating electrodes placed inside, while allowing a separation distance that prevents any electrode from interfering with the signal of the neighboring electrodes. This limitation on the number of electrodes in turn limits the number of frequencies that can be simultaneously represented at any time.

The work in this paper focuses on the signal processing aspect of the device, specifically in terms of frequency choice and resolution. It proposes adjustments to the Continuous Interleaved Sampling (CIS) algorithm, which is described in Chapter 2. The first proposition attempts to address a particular pitfall of the original algorithm, allowing for a more continuous frequency spectrum to be represented across the device. The second proposition approaches the processing of sound with music in mind, rather than speech. While speech comprehension is the typical focus of cochlear implant work, little work has been done to address the possibility of restoring the enjoyment of music to cochlear implant users. This method of approach alters significantly certain static aspects of the original algorithm, while attmepting to keep the computational complexity as low as possible.

b. Purpose of the Study

The goal is to incrementally improve the quality of sound perception in cochlear implant users by approaching the problem from a physics and signal processing perspective.

c. Scope of the Study

Two methods are proposed for adjusting the existing algorithm. One recorded speech sample was used to show how each algorithm might be applied to speech, and a few musical samples were either created or recorded. These samples were analyzed with the existing algorithm and the adjusted ones, and their outputs were analyzed and the results presented below.

d. Challenge of the Implant

The main challenge of this study was in analyzing the effectiveness of the applied methods without the use of a vocoder that uses the frequency information, or access to test subjects.

Results were therefore based on plots of the output of the peak frequency selection portion of the algorithm compared to plots of FFTs of the original signals.

Chapter 2. Background

The human ear is comprosed of two primary systems: the auditory system, responsible for hearing, and the vestibular system, which controls balance and contributes to spatial-positioning. The human hearing system consists of the inner, middle, and outer ear, as shown in Figure 2.1. The function of the ear is to translate environmental sound into an electrical signal that the brain interprets.

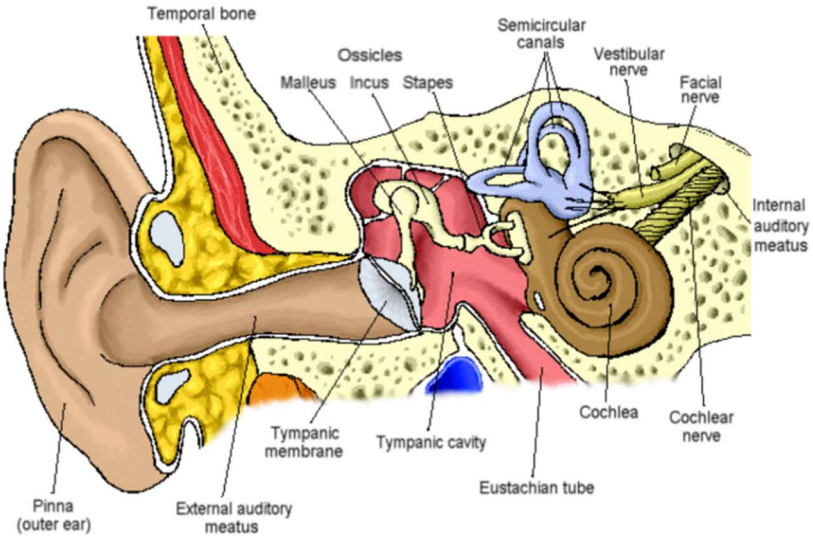

Figure 2.1 The anatomical structures of the outer, middle, and inner ear. (Cochlear Implants and Hearing, Cochlear Implant Hackathon (2021))

a. Parts of the Ear

- Outer Ear: the pinna and the external auditory canal
- Middle ear: the tympanic membrane and the three ossicles
- Inner Ear: the cochlea, the vestibule, and the semicircular canals

b. Hearing Mechanism

The pinna collects sound and directs the sound waves down the external auditory canal to the middle ear. These sound waves strike the tympanic membrane, causing it to vibrate. The vibrations are then transferred to the tiny ossicles—malleus, incus, and stapes—for amplification. As the vibrations move toward the end of the middle ear, they interact with the oval window, causing the organ of Corti to shake. This creates a pressure wave that travels along the basilar membrane in the cochlea. The outer hair cells amplify the incoming sound waves, while the inner hair cells convert these sound waves into electrical impulses. Low-frequency waves stimulate the inner hair cells in the apex, the innermost part of the organ of Corti, while high-frequency waves stimulate the inner hair cells in the base. As the sound waves are converted by the hair cells, the microvilli release ions, leading to the release of neurotransmitters that bind to receptors on the auditory nerve. This process sends the converted electrical signals to the brain (Brownell, 1997).

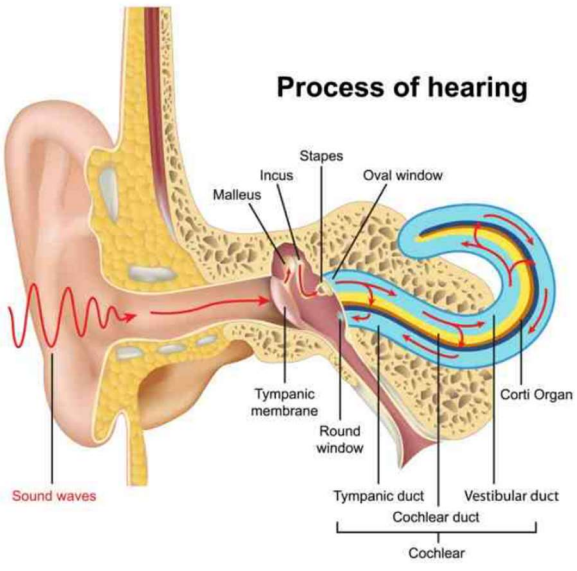

Figure 2.2 The mechanism of hearing, from sound passing the pinna, to vibrating the tympanic membrane and ossicles, to vibrating the hair cells in the Organ of Corti, which then transmits an electrical signal to the brain

c. Cochlea

The cochlea (Figure 2.3), which superficially resembles a snail's spiraled shell, contains three internal membranes and three fluid filled tubes. There are two openings on the surface of the cochlea called the oval window and the round window. The three membranes are Reissner's, the basilar, and the tectorial, and the fluid filled tubes are the scala vestibuli, the scala tympani, and the scala media. The scala vestibuli and scala tympani contain a fluid called the perilymph, and the scala media contains a fluid called the endolymph. The perilymph is rich in sodium ions, and the endolymph is rich in potassium ions.

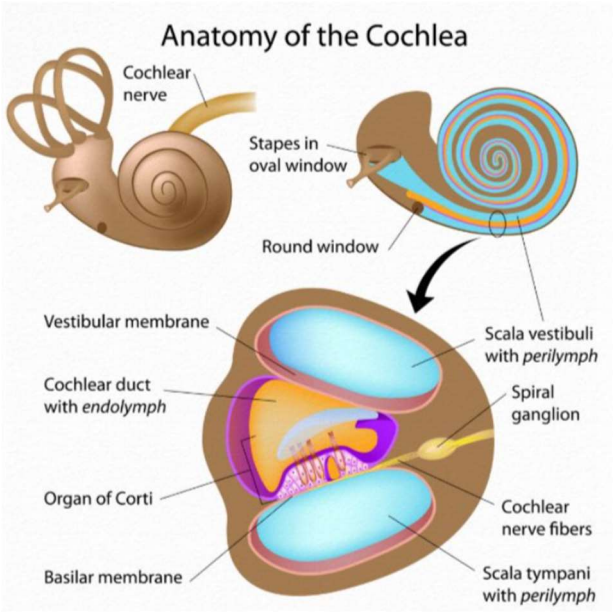

Figure 2.3 A cross-section of the human cochlea, with basic structures highlighted (Alila Medical Media 2019)

The scala vestibuli and scala media are separated by Reissner's membrane, which acts as a selective barrier for lymphatic fluids. The basilar membrane separates the scala tympani and the scala media, and is structurally designed to vibrate in a tonotopical organization. It will resonate with low frequencies near the apex and high frequencies near the base. The tectorial membrane is inside of the scala media and interacts with the inner and outer hair cells of the organ of Corti,

supporting the transmission of longitudinal waves at acoustically relevant frequencies. (Meaud & Grosch 2010).

The organ of Corti (Figure 2.4) is a complex organ with thousands of hair cells called stereocilia. As sound waves travel through the scala vestibuli and back up the scala tympani, the basilar membrane vibrates, causing the organ of Corti to move in relation to the tectorial membrane. This shearing motion tilts the hair cells at specific locations, causing potassium ion channels to open. These potassium ions cause depolarization and release neurotransmitters, creating action potentials in the auditory nerve, which then transmits the electrical signal to the brain.

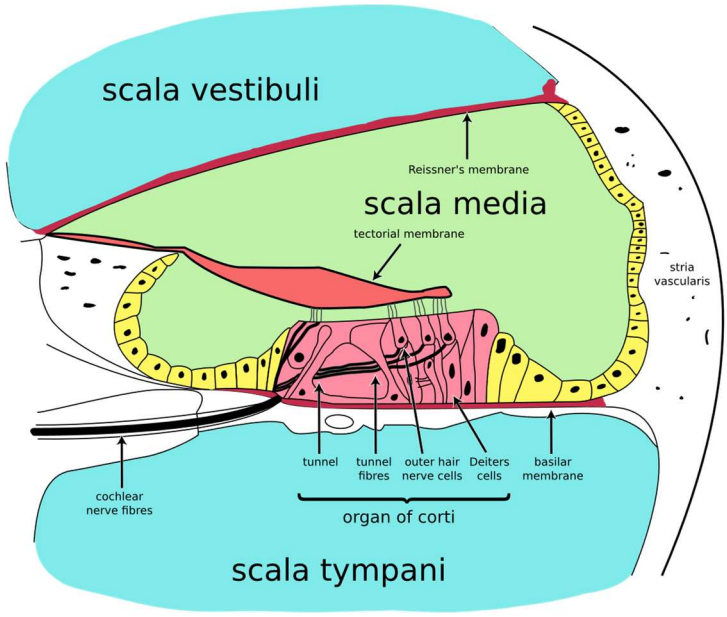

Figure 2.4 A cross-section of the human cochlea, with the parts of the organ of Corti labeled

Humans can perceive and analyze a wide range of frequencies, from as low as 16-20Hz up to 20kHz. This range tends to decrease over time, typically with a loss of sensitivity to higher frequencies.

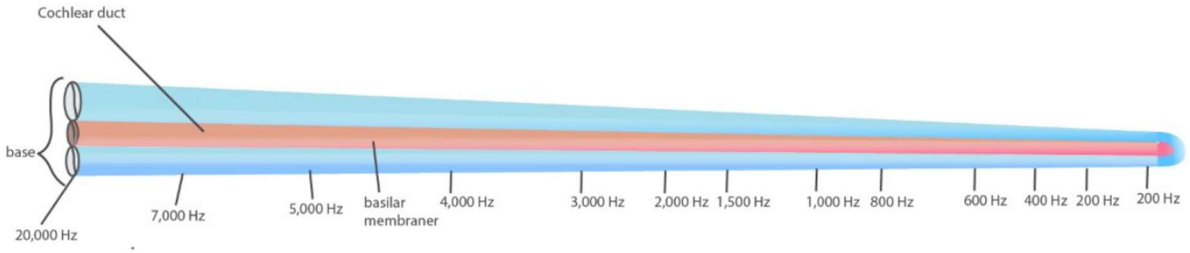

Figure 2.5 Cochlear frequency map of tonotopical organization

d. Hearing Loss

Hearing loss is caused by damage to the outer, middle, or inner ear due to congenital factors, aging, genetic factors, viral infection, disease, stroke, and other external factors like noise exposure and smoking. There are three types of hearing loss: conductive, sensorineural, and mixed (Brodie, Smith, & Ray, 2018)

- Conductive hearing loss is caused by damage to the outer and middle ear mainly due to obstruction in the region inhibiting the sound from traveling. The treatment for conductive hearing loss is medication, but if the condition worsens, surgery is required.
- Sensorineural hearing loss is damage to the inner ear, commonly hair cell and nerve impairment, causing obstacles for the sound transmission from the inner ear to the auditory nerve. Treatment includes surgery and hearing aids.
- Mixed hearing loss combines conductive and sensorineural hearing loss.

The type of hearing aid is recommended based on the degree of the hearing loss, whether moderate, severe, or profound. A nonsurgical hearing aid is preferred for mild hearing loss, while

severe hearing loss may require surgical help. Profound hearing loss often requires cochlear implantation.

e. Cochlear Implant

A cochlear implant device replicates the functions of the ear by capturing, transmitting, and delivering electrical impulses to the auditory nerves. Researchers report significant improvements in cochlear implants. These devices have become more reliable, with electrodes now able to stimulate multiple regions of the cochlea, and advancements in sound processing strategies based on user feedback. (Eshraghi, et al., 2012)

A cochlear implant is a small prosthetic device designed to assist individuals with severe to profound hearing loss (Dorman & Parkin, 2015). The implant mimics the functions of the outer, middle, and inner ear, directly bypassing the damaged regions to stimulate the auditory nerve. The device consists of components placed behind the ear and others surgically implanted inside the ear, as illustrated in Figure 2.5.

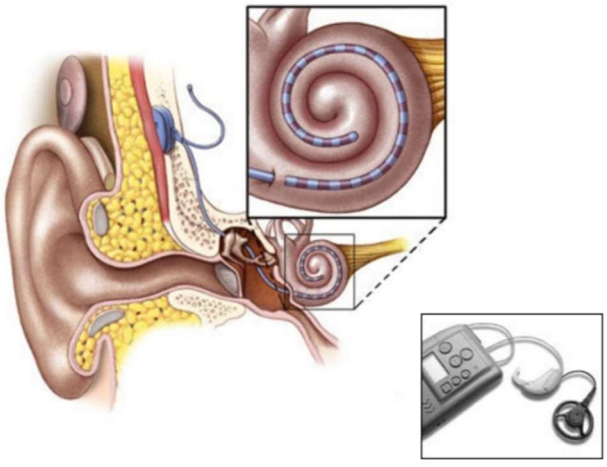

Figure 2.6 Illustration of a cochlear implant
(Cochlear Implants and Hearing, Cochlear Implant Hackathon (2021))

A cochlear implant consists of five parts:

1. Microphone: converts environmental sound to an electronic signal which is sent to the processor
2. Speech processor: analyzes and performs all processing on the sound, then sends the processed signal wirelessly to a surgically implanted receiver
3. Receiver: relays the received information from the speech processor and passes it down to the stimulator
4. Stimulator: converts the signal from the receiver into electrical impulses that are triggered on each of the electrodes in the electrode array\
5. Electrode Array: a strip of electrodes inserted into te cochlea which attempts to simulate the function of the organ of Corti's hair cells in the cochlea by sending electronic impulses directly to the nerve cells.

f. Continuous Interleaved Sampling

Continuous Interleaved Sampling (CIS) is a signal processing algorithm used in several cochlear implants on the commercial market. The overall process of the CIS algorithm has 7 steps:

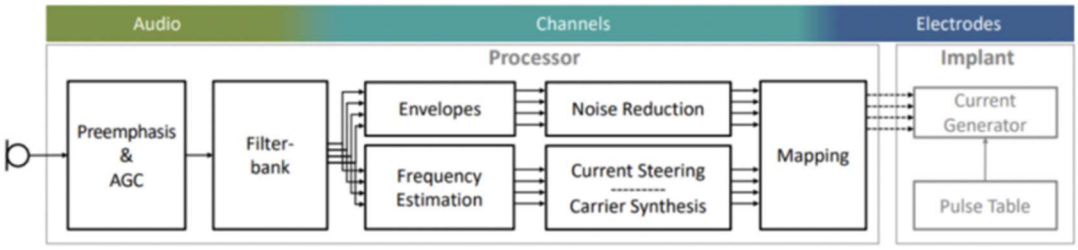

Figure 2.7 The Continuous Interleaved Sampling Steps (Cochlear Implant Hackathon (2021))

Step1: Pre-emphasis and Automatic Gain Control (AGC)

- Pre-emphasis

First, a high-pass filter is applied to reduce low-frequency noise in the environment, whitening the input spectrum, and providing reduction in the interference of the user's own voice. This mimics the reduced sensitivity of normal hearing listeners to low-frequency sounds.

- Automatic Gain Control

Dynamic compression is applied to the incoming signal automatically with a control loop with two parts. The slow loop compresses sound levels above 55dB without dimishing short-term intensity fluctuations in human speech patterns. The fast loop reduces gain immediately when sudden loud sounds occur, protecting the user from painful noises. Together these two loops provide the listener with a more comfortable experience without sacrificing the dynamics of the amplitude envelope's shape.

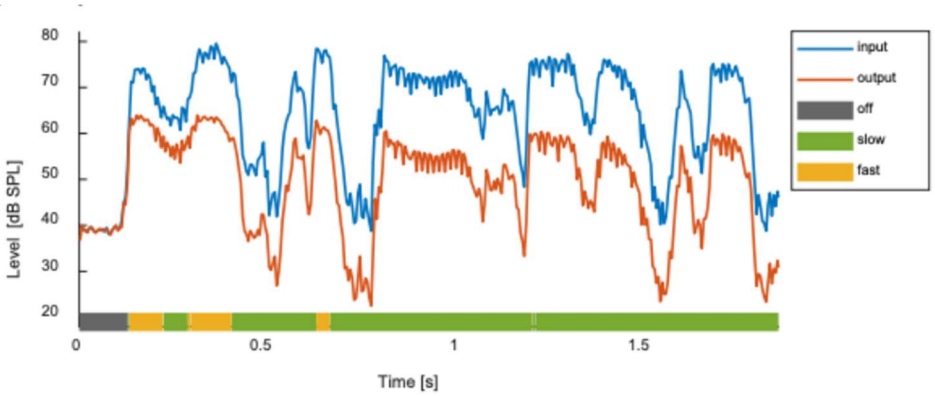

Figure 2.8 Automatic Gain Control (AGC) applied to an input signal (Cochlear Implant Hackathon (2021))

Step 2: Filter bank

The signal is then split into 15 frequency bands, in order to be distributed among the 15 stimulation channels. The input signal has a sampling frequency of 17400Hz, and the sample length is 256 samples, giving 128 linearly distributed frequency bins, The frequency resolution is 68Hz. These bins will be distributed among the 15 channels according to the Greenwood function, which gives an estimation of the frequency-to-place positioning relationship in the cochlea.

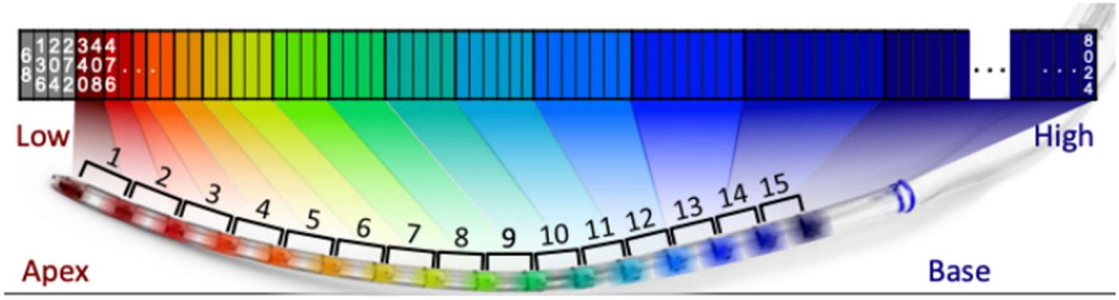

Figure 2.9 Distribution of frequency bins to channels (Cochlear Implant Hackathon (2021))

Step 3: Envelope

The signal envelope is a Hilbert transform which gives a function that describes the amplitude envelope of the signal, drawn by connecting the peaks of each oscillation in the original time signal, as shown in Figure 2.10. This envelope is used to control the final amplitudes and the output signal.

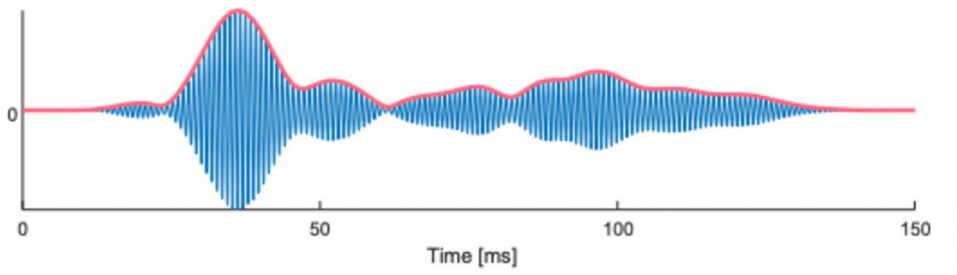

Figure 2.10 Amplitude envelope obtained by the Hilbert transform

Step 4: Frequency Estimation and Noise Reduction

For each of the 15 channels, the frequency bins allocated to that channel will be iterated over, determining which has the highest amplitude. If either of the edge bins are the highest, they will be chosen, otherwise a parabolic fit method will be used to estimate the intermediate frequency that may be present in the signal.

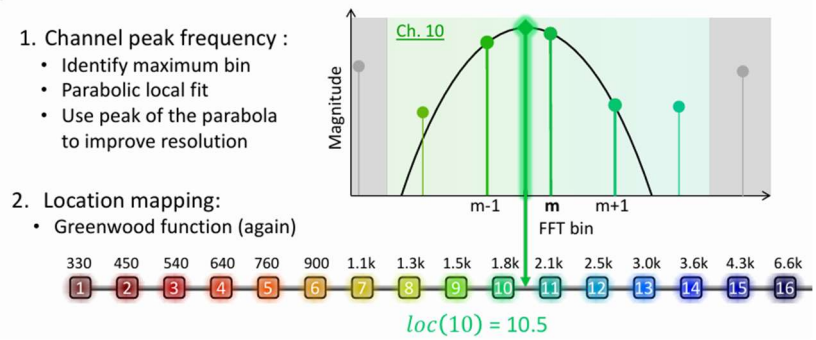

Figure 2.11 Parabolic fit method of determining the intermediate frequency in a channel

Step 5: Current Steering and Carrier Synthesis

Once a frequency is chosen for a particular channel, this frequency estimation is shifted to one of 8 discrete positions between the two electrodes associated with that channel, and then each electrode is applied a steering weight.

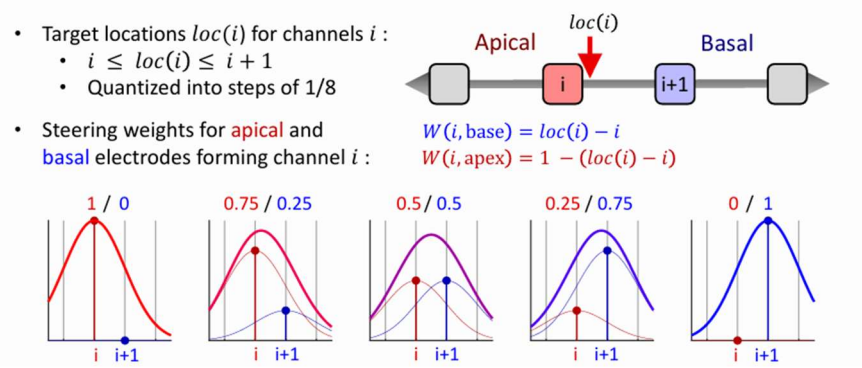

Figure 2.12 Explanation of current steering

Next a square wave pulse pattern associated with the chosen frequency is generated based on the period of the wave and the stimulation rate.

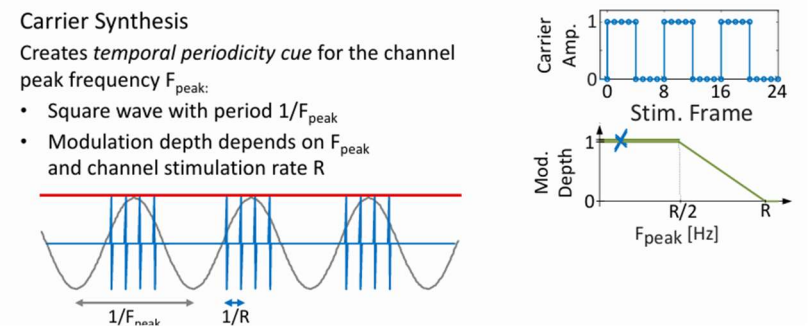

Figure 2.13 Carrier synthesis in the implant device

Step 6: Mapping

The envelopes, modulated carrier, and current steering weights are then mapped to generate a pair of amplitudes to send to each pair of electrodes during each stimulation frame.

Step 7: Current Generation

The electrode pairs are then fired in a staggered order, to prevent crosstalk between the electrodes as they stimulate the cochlear nerve cells.

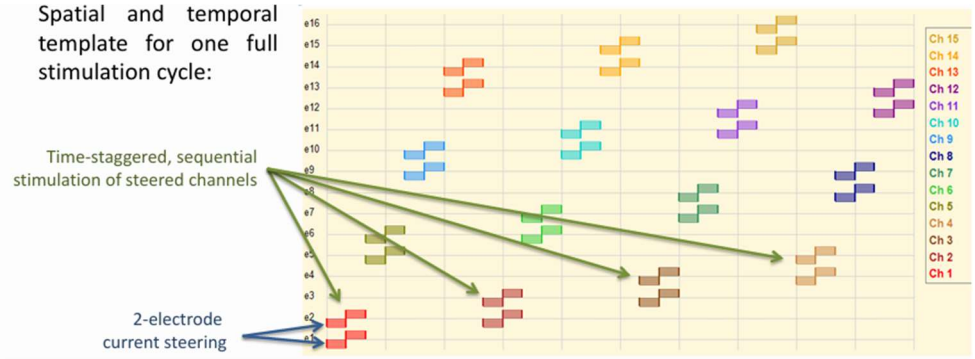

Figure 2.14 The static staggering order of electrode pair firing

Chapter 3. Methods

a. The Software

The software used for the entire project was MATLAB. The codebase was provided by Advanced Bionics in the form of an open-source Hackathon. This code is a MATLAB version of their proprietary code used in some of their implants. The code was acquired by Dr. Yoshida and was edited by a previous graduate student, Sylvia Robert (MS Southeastern Louisiana University, do I need to give something like this?), for her own thesis project. That edited code was then provided for this project.

In particular, the peak frequency finding function, specPeakLocatorFunc.m, within the codebase was edited in order to attempt different methods of frequency selection. New functions were implemented for peak finding and selection, and for electrode firing order determination. (code in Appendix A)

b. Method 1: Bin Overlap Between Adjacent Channels

i. Bin Allocation

In the HiRes algorithm (Nogueira, Litvak, Edler, Ostermann, & Büchner, 2009), the number of allocated bins per channel is static, and are determined by the linear separation distance of the electrodes, the positioning of the implant within the cochlea, and the tonotopical organization of the cochlea. The constant linear separation of the electrodes leads to a result of smaller frequency ranges in the first several channels.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Center frequency (Hz)	408	544	646	748	884	1020	1190	1427	1700	2005	2379	2821	3330	3942	6491

Table 1 Center frequencies of channels

Since a low resolution FFT (described in greater detail in Chapter 2.c) is used to determine the frequencies of each bin, the lower frequency bins are allocated a small number of bins. The first six channels are allocated less than three bins each.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n (# of bins)	2	2	1	2	2	2	3	4	4	5	6	7	8	10	55

Table 2 Number of frequency bins allocated to each channel

ii. Downfalls of the Parabolic Fit Method of Frequency Determination

The HiRes algorithm for frequency determination uses a parabolic fit method. This method first iterates over all of the frequency bins in the channel, finding the bin with the highest amplitude. Next, it determines whether the bin is on one of the 'edges', or more specifically, if it is the left-most, or right-most bin in the channel. If it determines that the highes amplitude bin is on one of the edges, it simply chooses that frequency as the one to represent on this channel in its final output. If it determines that the highest amplitude frequency bin is not on one of the edges, it then uses that highest bin and its adjacent bins to perform a parabolic fit. This parabolic fit provides an estimate of what might be the actual frequency in the incoming signal in that frequency range. Once the frequency is chosen in this manner, the current steering algorithm then determines how to activate the two electrodes for that channel. (Nogueira, Litvak, Edler, Ostermann, & Büchner, 2009)

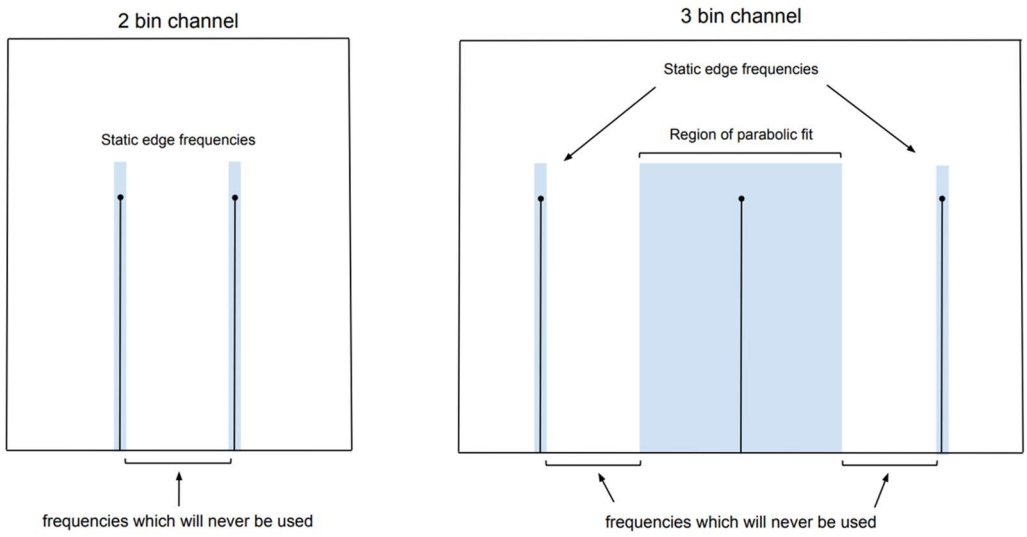

Figure 3.1 Possible frequency choices for channels with 2 and 3 bins.
Blue regions indicate frequencies that may be selected by the pre-existing algorithm.

This method has some glaring faults. As described above, it is obvious that the parabolic fit method requires three data-points. It also requires that the highest amplitude frequency bin be the center one of those three. In the case of a channel with three frequency bins, there is a one in three chance that the center bin will be the highest, so the method is only used in one in three time frames. This means that two-thirds of the time, one of two static frequencies will be chosen.

Furthermore, if the central frequency of three is chosen and the parabolic fit method is used, there are a limited number of available frequencies within the channel's total range that are choosable. This is because the parabolic fit can only choose frequencies that extend halfway to the next frequency bin. Once crossing the half way point, the neighboring bin will be higher in amplitude, and then selected outright with no parabolic fit.

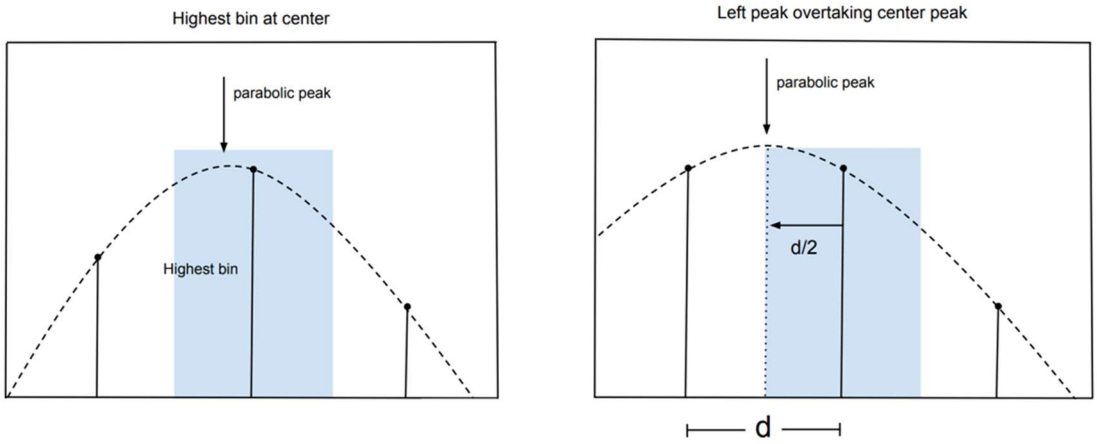

Figure 3.2 Two examples of possible parabolic fit frequency determinations. The blue region is the total range of possible parabolic fit choices when the center bin is the highest.

The first five channels have less than three frequency bins, which means that this parabolic fit method is never used at all. The next several channels have between three and five bins, so the likelihood of using the parabolic fit to find some intermediate, more accurate frequency is still quite small. One can determine the probability, P, of determining a non-static frequency in a channel with *n* allocated frequency bins as:

$$P(n) = \max\left(0\,, \frac{n-2}{n}\right)$$

The probabilities for each channel to find a close approximation of a signal frequency by using the parabolic fit is shown in the chart below.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n (# of bins)	2	2	1	2	2	2	3	4	4	5	6	7	8	10	55
P(n)	0	0	0	0	0	0	0.33	0.50	0.50	0.60	0.67	0.71	0.75	0.8	0.96

Table 3 Probability that a channel will use the parabolic fit instead of a static edge frequency

iii. Widening and Overlapping Channels

In order to allow the parabolic fit method to find intermediate frequencies in these lower channels, we must allow more bins to be considered by each channel. First, each channel's bin allocation is widened. This is done simply by lowering the lowest allocated bin number by one and increasing the highest by one.

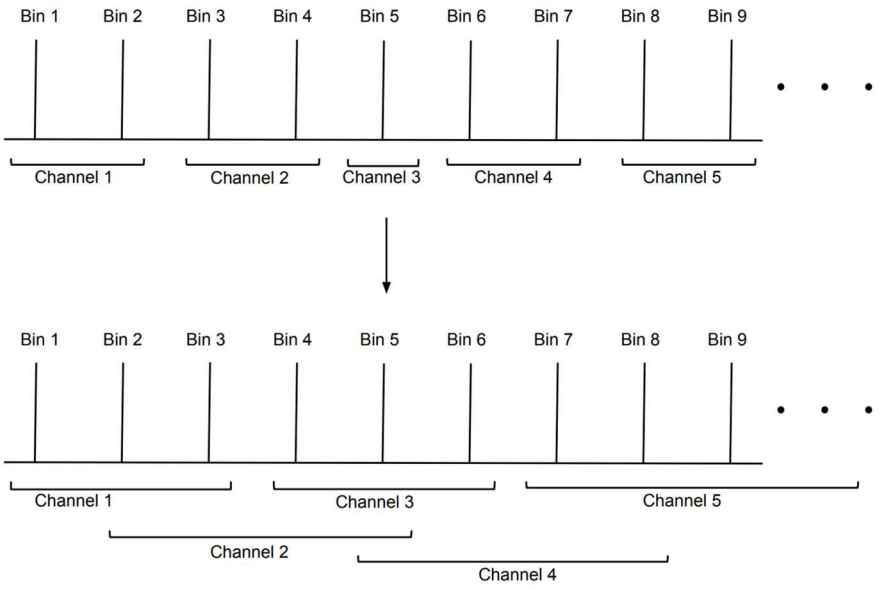

Figure 3.3 Comparison of the existing bin allocation paradigm and the overlapping bin paradigm

In this way, if an edge bin is found to be the highest amplitude bin, it can be determined that this frequency lies in the neighboring channel's range and should be ignored. The next highest bin can now be selected. For example, if Channel 1 is being considered and Bin 3 is found to be the highest in amplitude, it will be ignored and the next highest bin in Channel 1 will be used. This is because Bin 3 is within Channel 2's range of frequencies. Since Channel 1 has only 2 bins within its

By simply overlapping the bins which are allocated to each channel, the probability that a parabolic fit will be used to the intended effect increases.

Channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
n (# of bins)	4	4	3	4	4	4	5	6	6	7	8	9	10	12	57
P(n)	0.50	0.50	0.33	0.50	0.50	0.50	0.60	0.67	0.67	0.71	0.75	0.78	0.8	0.83	0.97

Table 4 Increased probability for use of the parabolic fit with overlapping channels

c. Method 2: Increased Low-Frequency Resolution and Firing Order Selection

i. Musical focus

When considering the representation of sound in a cochlear implant, the goal is most often to allow the user to understand speech. The CIS algorithm considers the wide frequency content and complexity of sounds in human speech, and many of the parameters were chosen with that in mind. While the recognition of speech is arguably the most important need for a cochlear implant user, the enjoyment of music could provide a quality of life improvement that undoubtedly many implant users would appreciate. Music can also be quite complex, and span the entire tonal range of human hearing, but it is just as often quite simple, and within a limited range of frequencies.

The method described below attempts to adjust the parameters and methodology of the CIS algorithm to account for the ways in which music differs from human speech. First, a higher order FFT can provide better frequency resolution and accuracy in the lower half of the spectrum. Second, an adjustment to the distribution of frequencies represented on the device is proposed, starting by removing the limitation of one frequency per channel. Third, in order to allow more than one frequency per channel, a dynamic electrode firing order determination method is discussed.

ii. Higher order FFT

In the original Hi-Res strategy used in the Hackathon code, an FFT is performed on input blocks of $L = 256$ samples of a windowed audio signal,

$$x_w(l) = x(l)w(l), \qquad l = 0, \dots, L-1,$$

Where $x(l)$ is the input signal and $w(l)$ is a 256-Blackman-Hanning window. The FFT is taken and only the real part is used:

$$X(n) = Re\{FFT(x_w(l))\} = \frac{1}{L}\sum_{l=0}^{L-1} x_w(l)\cos\,(2\pi\frac{n}{L}l)$$

This gives a Fourier transform with linearly spaced bins. The input sample rate used by the implant is 17400 bps, so this gives a frequency separation distance of ~68Hz. The choice of an sample length of 256 was to provide a balace between frequency resolution and temporal resolution. In human speech, a high temporal resolution is required in order to perceive most consonant sounds. In music, however, this is not necessarily required. This allows us to sacrifice temporal resolution for increased frequency resolution. By increasing the sample length used in the FFT function by four to 1024, we are able to reduced the frequency separation of the FFT bins to ~17Hz with minimal increase in computational demand.

In Western music, the most commonly used musical frequency relationship is the twelve-tone equal temperment scale. This system defines an octave as a doubling in frequency, and each octave is divided into twelve tones equally spaced on a logarithmic scale. Given some reference pitch P_a, and other pitch P_n, can be found by:

$$P_n = P_a\left(\sqrt[12]{2}\right)^{n-a}$$

Using this relationship, we can determine tonal frequency widths anywhere in the frequency spectrum. The tone known as C4 (middle C on a piano) has a frequency of 261.63Hz. This frequency lies near the center of the first channel of the cochlear implant. Using the relationship above, the frequency separation between this note and the semitone immediately above it is 15.55Hz. This suggests that in order for the implant to allow a user to discern musical tones in this range, the frequency resolution of the FFT used in the processing of the audio must be near this value.

While the parabolic fit method used in the device attempts to account for the lack of frequency resolution, it does not take into account some common musical relationships between tones. A common harmonic element used in music is the augmented chord. This is a chord, typically a triad (three notes played together within one octave), which consists of a base tone, its perfect fifth (7 semitones higher) and either the second or fourth, both of which are two semitones in frequency distance from the base or fifth, respectively. In the frequency range of the first channel, this puts the frequency separation at ~34Hz. Two tones at this separation distance would be unresolvable using the original FFT used in the HiRes algorithm. Tones at further separations would also be unresolvable if they do not fall very close to the precise frequencies of the chosen frequency bins.

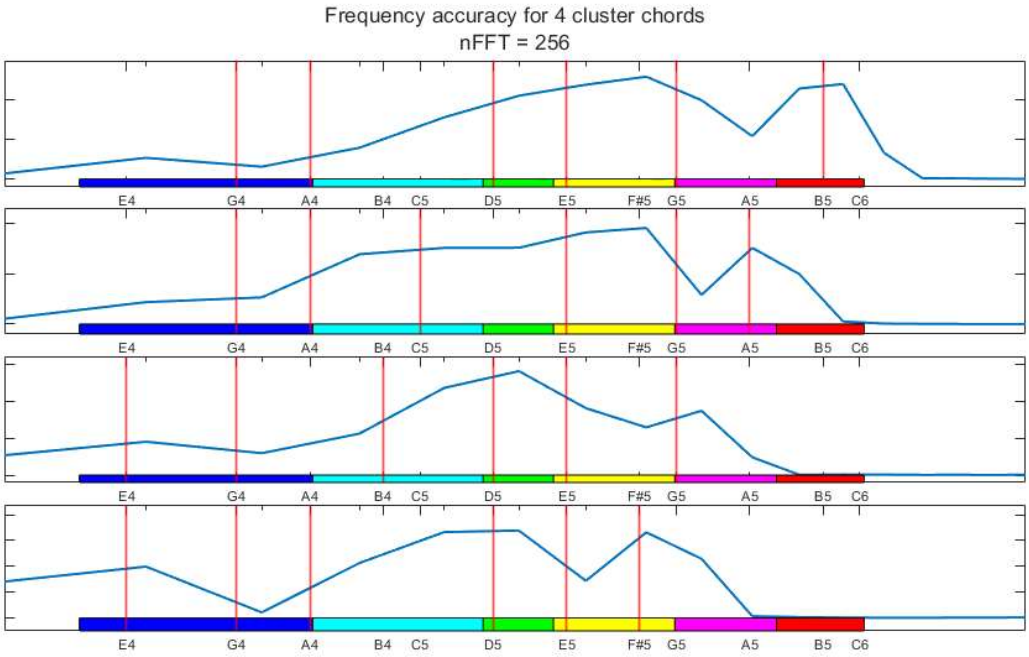

Figure 3.4 STFT of a sequence of 4 cluster chords with nFFT of 256. The 6 olored bars at the bottom indicate the first 6 channels, and the vertical red lines indicate the actual notes in the chords. The individual notes are unresolved.

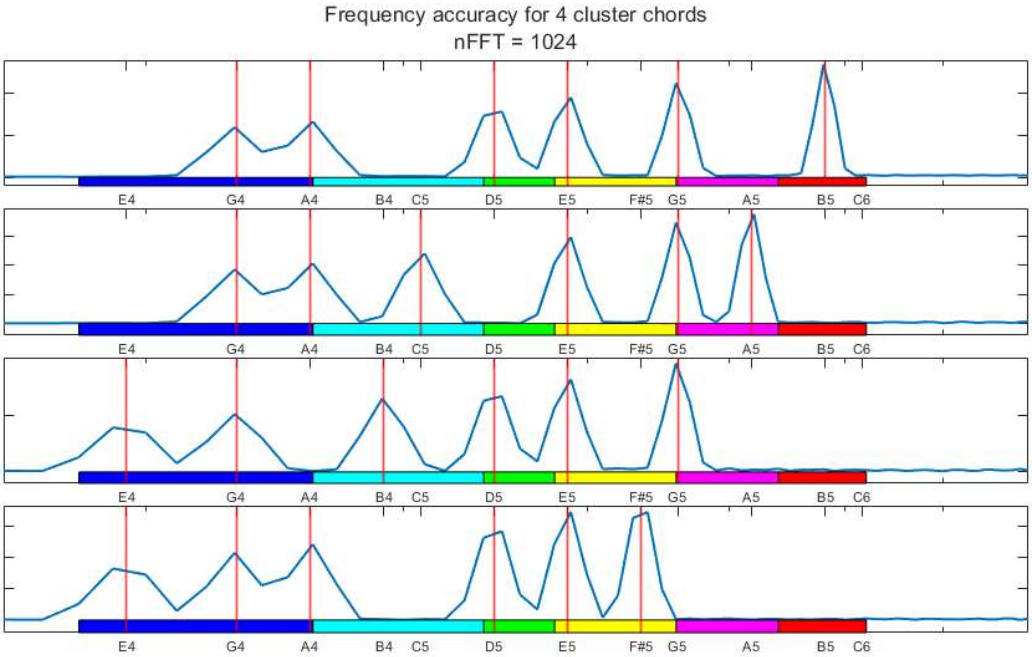

Figure 3.5 The same as Figure 3.4, but with an nFFT of 1024, giving much better frequency resolution.

With increasing frequencies, the separation distance between semitones increases, doubling at every octave. Any frequencies in the signal which are above C6 (1046.5Hz) have a frequency separation which is near the original frequency resolution of 68Hz, or higher. Depending on the computational abilities of the implant device, it may be desired to split the frequency spectrum at this point and perform a higher order STFT on the lower half, and retain the lower order one on the higher half.

iii. Peak Finding

Musical compositions often consist of chords with notes clustered close together, and the frequencies of two or more of these notes may fall within the frequency range of one channel in the cochlear implant, as seen in the example in Figures 3.4 and 3.5. The current algorithm only allows for a single tone to be played in each channel's range, and that one tone per channel be played at

all times. This choice could lead to frequencies being ignored, incorrect frequencies determined due to averaging between close together notes, and the inclusion of unimportant tonal information.

In order to allow more than one frequency to be represented per channel, an alternate method of finding peaks must be used. By searching through the magnitudes of the FFT and using prominence, relative height above neighboring points, a list of candidate peaks can be found. Next, this list will be iterated through, selecting the 15 peaks based on several criteria:

1. Prominence.

   Matlab's internal function *findpeaks()* returns several values. For each peak it finds, it also returns the prominence of that peak. This is a measure of how much higher the peak's value is over its neighbors. By setting a minimum prominence, smaller peaks which may be attributed to noise may be ignored and more prominent peaks found. A prominence value of 5.0 was selected.

2. Loudness relative to a Fletcher-Munson equal loudness contour

   A function for calculating the equal loudness contour based on the standard ISO-226 values for discrete frequencies was used, and values for the specific frequencies in question were interpolated. A loudness of 75 phons was selected as the relative loudness curve. When considering the peaks in the signal, ones which were above or nearest to this curve were considered first. Any peaks that lie well below the curve were ignored.

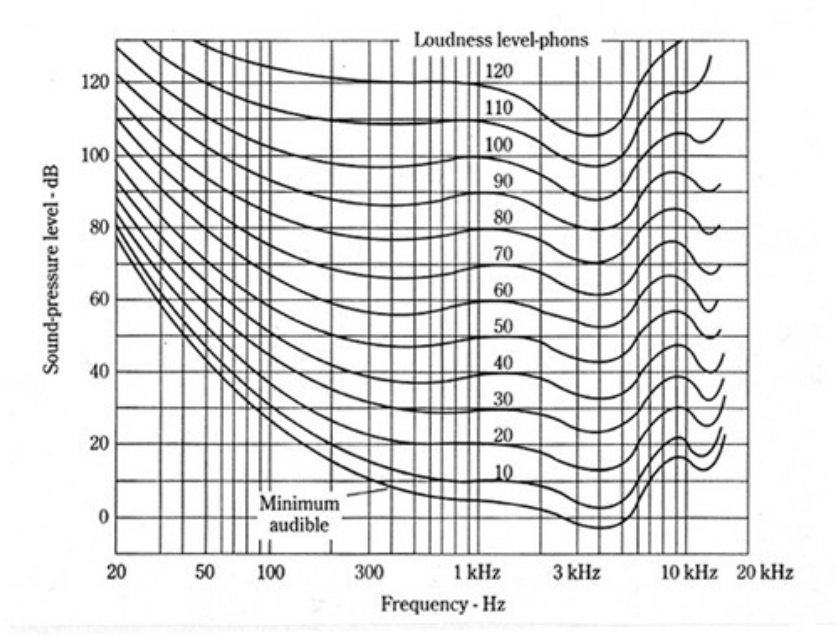

Figure 3.6 Fletcher-Munson Curve (equal loudness contour)

3. Low frequencies are favored over higher ones.

    Since most of the important melodic information in music is within the range of the dominant frequencies of the human voice, frequencies below 1000Hz were considered before higher ones.

4. Octave doubles will be removed.

    Many instruments, including the human voice, resonate significantly with overtones in integer multiples of their dominant frequency. In order to ensure that as many unique notes as possible are considered, octave multiples were suppressed, unless there was no other content.

5. Frequency repeating

In the case that there are very few frequencies present, the list of frequencies may be repeated in order to reinforce those frequencies and prevent quiet ambient noise from being included and amplified.

iv. Electrode Firing Order

In the implant, all of the frequencies are not represented simultaneously. Instead, the electrode pair associated with each channel is fired one at a time in a static, predetermined order. This firing order is shown in Figure 2.14, and listed in the Tabel 4 below:

Place in order	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th
Channel	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12
Distance to next	4	4	4	11	4	4	4	11	4	4	4	11	4	4	11

Table 5 The static order of channel electrode pair firing, and the distance between them.

This ordering assumes that there is one frequency per channel in every frame. The specific order was chosen in order to maximize the average distance between successive electrode pair firings to avoid any interference between the pulses emitted by the electrodes. This distance is counted in number of channels between each consecutive firing. This can be thought of as a linear distance, even though the implant device is curved.

In order to represent more than one frequency on some channels, there must be a new firing order determined. Since the number of frequencies is potentially different every frame, this new order must be dynamically determined. While determining this new order, the distance

between successive electrodes must still be considered. Below is an example of an array which contains the channel association of each of 15 found frequencies.

$$[1 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 3 \quad 4 \quad 6 \quad 6 \quad 6 \quad 7 \quad 8 \quad 11 \quad 12]$$

The brute force method of finding a new firing order consists of testing each possible permutation of the ordering of the 15 channels that must be fired. However, this would require testing 15! (factorial) possible permutations, which is over 1.3 trillion possible arrangements of 15 numbers. This is far too time and memory intensive to be calculated each frame.

The process of finding an optimal ordering consists of 4 steps:

1. Randomly shuffle the array.

   In order to begin the search with an increased likelihood of finding an acceptable ordering more quickly, the order of the array is first randomized.

   $$[1 \quad 8 \quad 2 \quad 7 \quad 11 \quad 1 \quad 1 \quad 6 \quad 4 \quad 1 \quad 12 \quad 2 \quad 6 \quad 6 \quad 2]$$

2. Split the array into two smaller arrays:

   By splitting the 15 member array into one 8 member array and one 7 member array, the testing of all possible permutations is reduced to 8! testings at the most.

   $$[1 \quad 8 \quad 2 \quad 7 \quad 11 \quad 1 \quad 1 \quad 6] \quad \leftarrow \quad split \quad \rightarrow \quad [4 \quad 1 \quad 12 \quad 2 \quad 6 \quad 6 \quad 2]$$

3. Find optimal orderings for each array. This is done in two parts:
   a. Iterate through permutations for each, testing for satisfaction of the distance criterion.

   $$[1 \quad 8 \quad 2 \quad 11 \quad 1 \quad 6 \quad 1 \quad 7] \qquad [6 \quad 4 \quad 1 \quad 12 \quad 3 \quad 6 \quad 2]$$

b. Once found, test the last member of the first against the first member of the second array for distance. Also test the first member of the first against the last member of the second. If these do not meet the distance requirement, as the above ordering does not with distances of 1 for each, rotate the second array until it does satisfy the requirement.

$$[1 \quad 8 \quad 2 \quad 11 \quad 1 \quad 6 \quad 1 \quad 7] \qquad [4 \quad 1 \quad 12 \quad 3 \quad 6 \quad 2 \quad 6]$$

4. Finally, concatenate the two arrays into the final array.

$$[1 \quad 8 \quad 2 \quad 11 \quad 1 \quad 6 \quad 1 \quad 7 \quad 4 \quad 1 \quad 12 \quad 3 \quad 6 \quad 2 \quad 6]$$

This final array is then used by the impulse generator to trigger the stimulation of each pair of electrodes.

Chapter 4. Results

a. Method 1: Bin Overlap

In order to see the results of widening the bin allocation on each channel, the interpolated frequencies found for each channel were plotted, and the electrodogram was comapred. 3 short audio samples were used; a short vocal sample of a voice saying “Is that chip on your shoulder chocolate?”, which will here after be referred to as the “chip clip”, a C4 major chord sustained for 1 second, and a single sine wave sweeping from C7 (2093 Hz) down to C4 (261.63 Hz). Below are the results.

Chip Clip

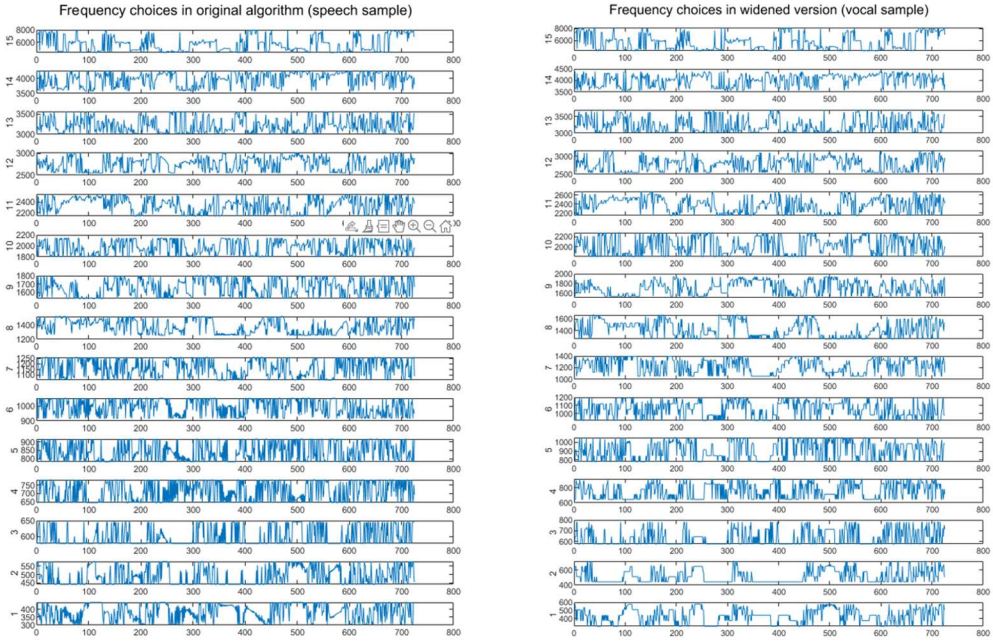

Figure 4.1 Comparison of vocal clip frequency determinations

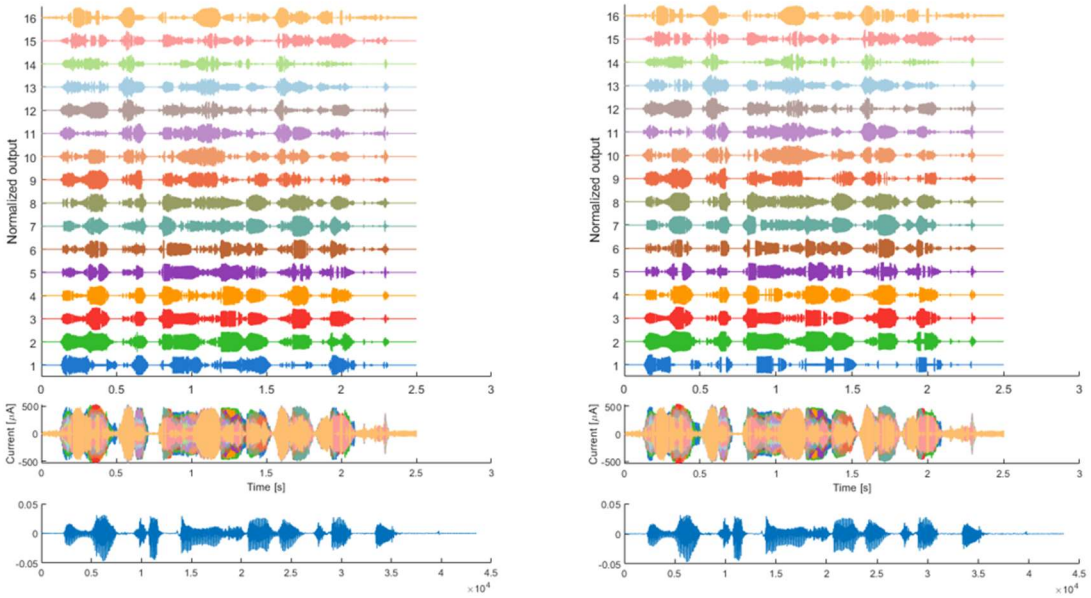

Figure 4.2 Electrodogram comparisons for the vocal sample (left: original, right: widened bin allocation)

C4 Major Chord

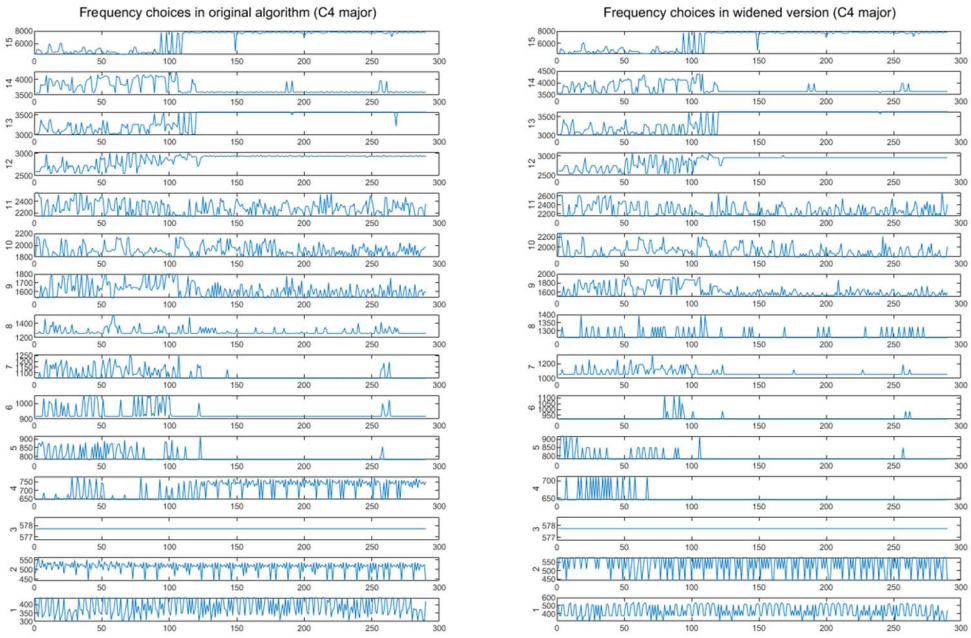

Figure 4.3 Comparison of C4 major chord frequency determinations

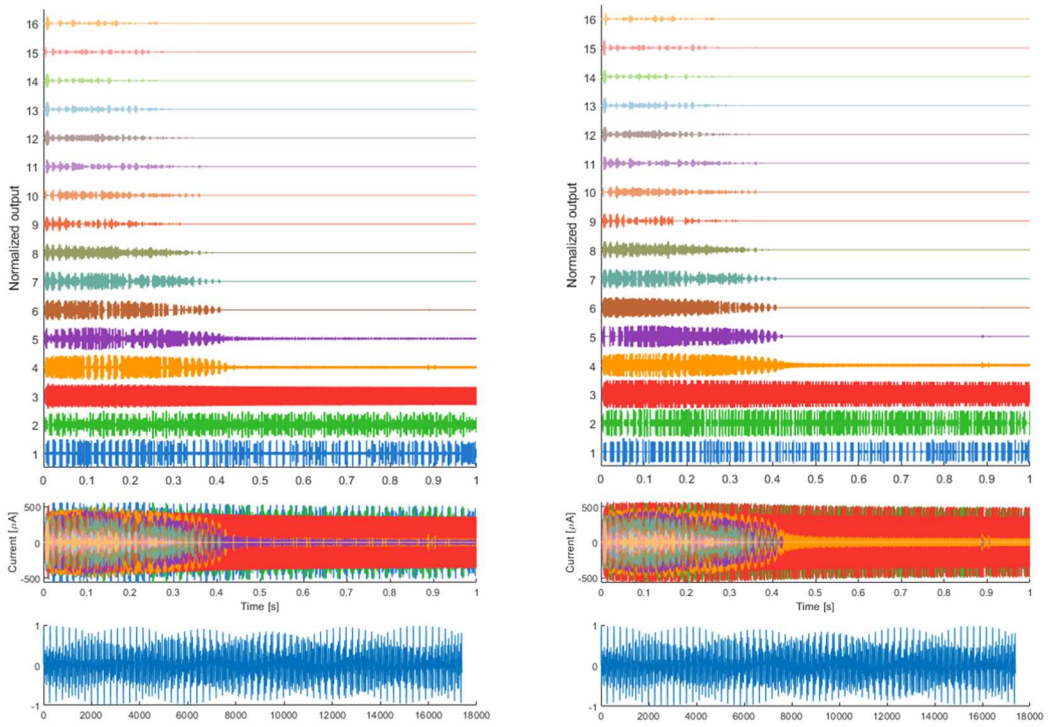

Figure 4.4 Electrodogram comparisons for the C4 major chord (left: original, right: widened bin allocation)

Sweep Down

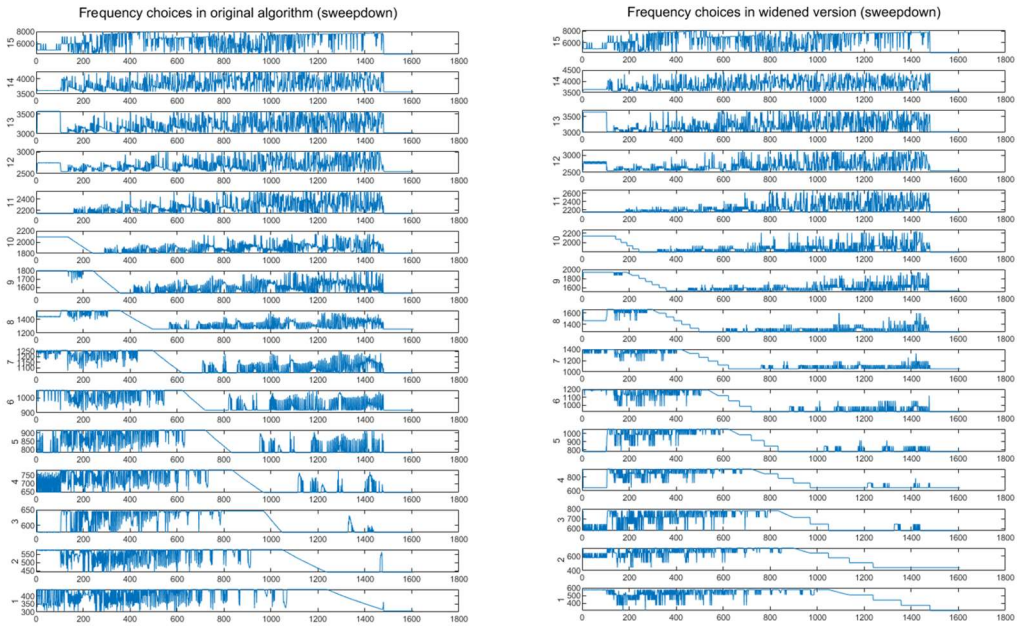

Figure 4.5 Comparison of sweep down frequency determinations

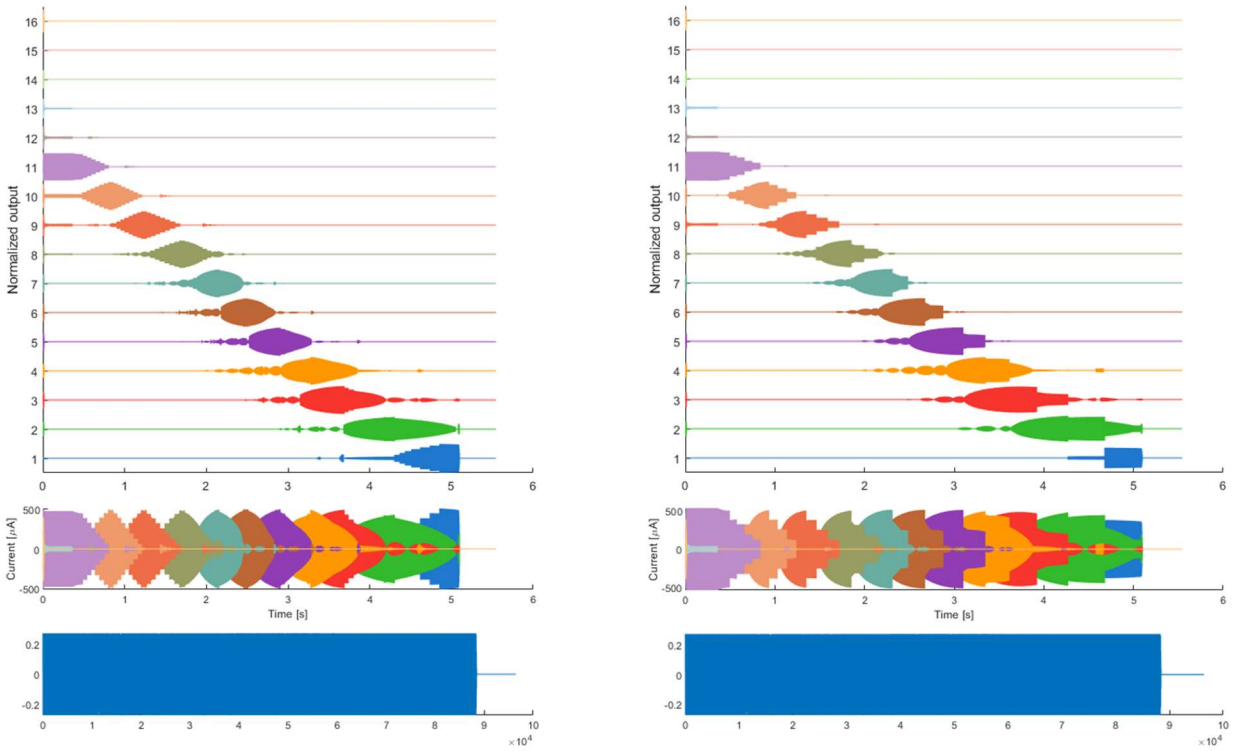

Figure 4.6 Electrodogram comparisons for the sweep down (left: original, right: widened bin allocation)

From these plots it can be seen that the widening of the bin allocation did not have the intended effect. Also the original algorithm does not behave in the way that the HiRes paper describes. For several of the frequency plots, widenening the allocations led to the code choosing discrete frequencies rather than intermediate ones throughout the range in each channel. The reason for this change is unknown, as it occurred even in the frequency range of the bins that were originally in each channel. This approach was abandoned after these intial plots were made in favor of method 2.

b. Method 2: Peak Finding and Electrode Firing Order Determination

i. Improved frequency resolution

Chip Clip

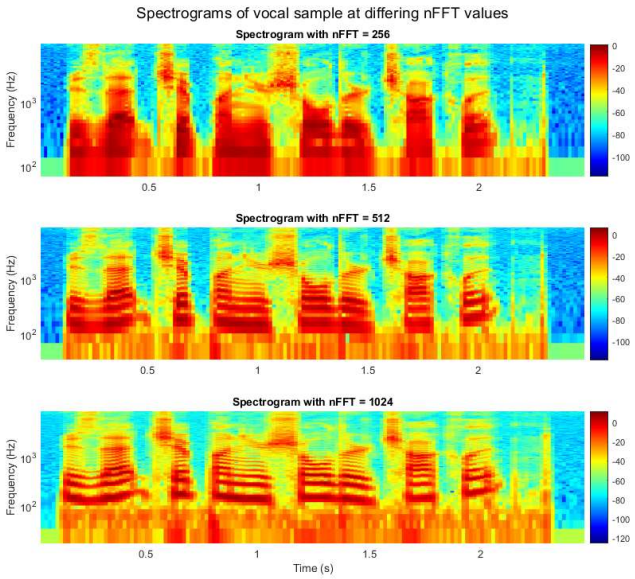

Figure 4.7 Spectrograms of the vocal sample at different frequency resolutions

With the higher nFFT values, the increase in frequency resolutions are apparent. In the original 256 plot, the tonal variation in the fundamental frequency of the voice is lost, the important variation being the rising tone at the end of the phrase indicating a question. In the other two plots, these become visible even with a single doubing of nFFT to 512.

C4 major

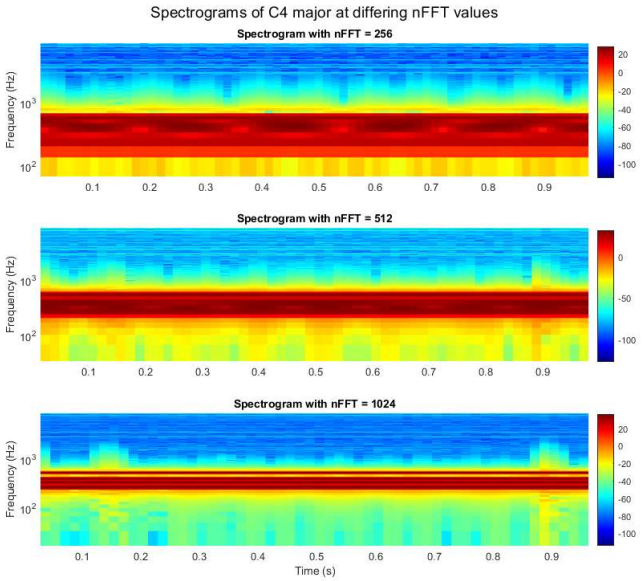

Figure 4.8 Spectrograms of the C4 major chord at different frequency resolutions

At the original resolution, the individual notes in the chord become distributed on adjacent bins, which would likely lead to only the highest one being chosen to be represented in the final signal. By increasing the nFFT value to 1024, intermediate values are found, which could allow each note to be resolved. This does confirm that increasing the resolution is crucial if the algorithm is to be expected to be used to represent musical chords in any way.

Sweep Down

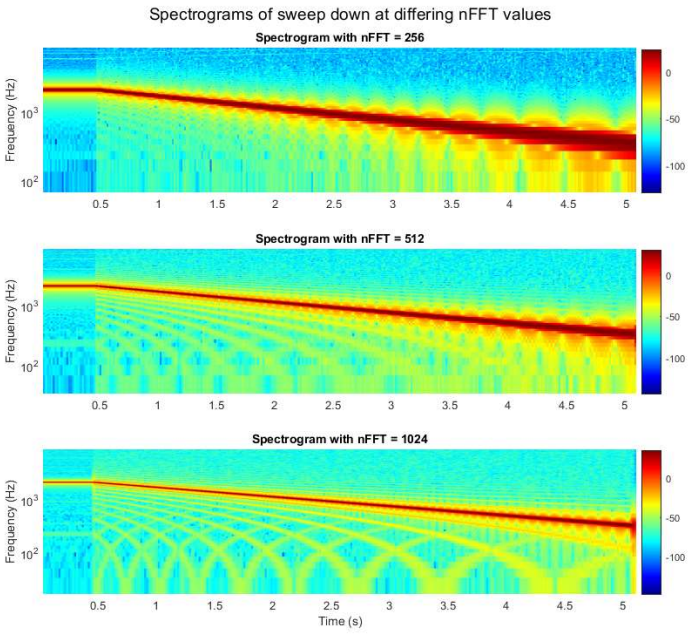

Figure 4.9 Spectrograms of the sweep down at different frequency resolutions

In these plots, one can see the downfall of the STFT. Because of the cutoffs at the edges of the analytical window for each frame, there are false frequencies detected, seen as the yellow colored sweeps down and up below the main tone. With increasing nFFT value, the overlap between successive frames is increased, allowing more averaging over more frames. This reduces the magnitude of these false frequencies, and also reduces the sweep rate.

ii. Electrode Pair Firing Order

Several starting arrangements were created in order to test the method written to determine a good firing order. Splitting the 15 channels into two groups of 8 and 7, and into three groups of 5, were tried. Each arrangement was run 100 times for each type of splitting, and the average distance

and number of times that one of the distances in an order was 1 or 0 was counted. Below are the results.

The original array

```
starting array:
     1     2     3     4     5     6     7     8     9    10    11    12    13    14    15


Amount of split arrays: 2
Results:
Average distance: 6.94
Amount that have ones or less: 63/100

Amount of split arrays: 3
Results:
Average distance: 6.3747
Amount that have ones or less: 63/100
```

Array #2

```
starting array:
     1     1     1     2     3     3     4     4     6     6     7     8     9    11    12


Amount of split arrays: 2
Results:
Average distance: 5.544
Amount that have ones or less: 72/100

Amount of split arrays: 3
Results:
Average distance: 4.9987
Amount that have ones or less: 71/100
```

Array #3

```
starting array:
     1     2     2     3     6     7     8    12    13    14    15    15    15    15    15


Amount of split arrays: 2
Results:
Average distance: 8.7973
Amount that have ones or less: 80/100

Amount of split arrays: 3
Results:
Average distance: 7.8027
Amount that have ones or less: 73/100
```

After testing these and many other arrays, the results remained fairly consistent. The average distances were between 4.8 and 9, and the average number of arrays out of 100 that had distances that are considered too small was between 60 and 80.

While the average distances seem to be in an acceptable range, the high ratio of arrays with small distances somewhere in them suggests that using this method would likely lead to a significant amount of crosstalk between electrodes, unless the function could be improved upon.

Chapter 5. Conclusion

Cochlear implants are just one of the technologies created in the attempt to bring hearing back to those who have lost it. The limitations of the device are many, and the problem of recreating sound accurately is difficult. For good reason, most teams who work on these devices focus on human speech. Interpersonal communication is a crucial part of life, and while it is not impossible to communicate as a deaf person, the loss of hearing does introduce difficulties.

I feel that the enjoyment of music is just as crucial a part of the human experience, and that more work should be done by those with a background in music in this field. The work in this paper shows that there are improvements that can be made, though they may require a complete overhaul of the approach used in these implants.

Neither of the methods proposed above were tested in cochlear implant users, but the results above could serve as a proof of concept that at minimum some incremental improvements can be made.

Appendix A

The following is the main script, which calls all the others.

```
clear


wide = false;
nft = 1024;
div = 1;
leading = false;
gain = 6.908; % originally 6.908
plotElec = false;
plotFfts = false;

%wavfile = 'sweepdown.wav';
%wavfile = 'c4major4.wav';
wavfile = 'AzBio_3_third.wav';


[y,Fs] = audioread(wavfile);

disp("original size");
disp(size(y));
disp("original Fs");
disp(Fs);


slice_size = round(Fs/div); % length of slice in data points
steps = floor(length(y)/slice_size); % number of slices (steps)

disp(append("the clip is ",num2str(length(y)/Fs)," seconds long")); %displays length
of full clip in seconds
%sound(y,Fs);
disp(size(y));



%%

[audioFinal,dFreqs] = main_call(1,div,gain,leading,plotElec,fold,wavfile, nft,wide);
if plotFfts
    figure(10);
    plot_ffts(audioFinal, 48000, y, Fs,1,1,binLocs,dFreqs,binLocArray,binNums);
end

%% play the audioFinal output sound

sound(audioFinal,48000);


%% returns frequency domain signal and frequencies for plotting
```

```
function [s, freq] = get_fft(signal,Fs)
    yfft = fft(signal);
    yfftshift = fftshift(yfft);
    n = length(yfft);
    fshift = (1:n/2)*(Fs/n);
    %yfs_half = floor(length(yfftshift)/2);
    s = yfftshift(end-length(fshift)+1:end);
    freq = fshift;
    %figure
    %plot(fshift,abs(yfftshift(yfs_half+1:end)));
end

function [psd, freq] = get_psd(signal,Fs)
    X = fft(signal);
    f = linspace(0, Fs/2, length(X)/2 + 1);
    %X_one_sided = 2 * abs(X(1:length(f))) / length(x);
    psd = (abs(X(1:length(f))).^2) / (length(signal) * Fs);
    freq = f;
end


function plot_ffts(processed, Fs1, original, Fs2,div,i,chosenB,freqs,bla,bn)

    %tiledlayout("flow");

    [sig1, freq1] = get_psd(processed,Fs1);
    [sig3, freq3] = get_psd(original,Fs2);
    [sig2, freq2] = get_fft(original,Fs2);
    spmax = max(abs(sig3));
    smax = max(abs(sig2));
    sFmax = max(abs(sig1));

    xrange = 8000;

    sig1 = sig1.*(smax/sFmax);
    sig3 = sig3.*(smax/spmax)*3;

    maximum = max(max(abs(sig1)),max(abs(sig2)))+3;

    %nexttile


    semilogx(freq1,abs(sig1),"Linewidth",1.5);
    hold on
    ylim([0,maximum]);

    xlim([200,xrange]);
    xticks([256,512,1024,2048,4096])

    yticks([]);
    title("vocoder psd");
    %nexttile
```

```
    title("original psd");


    semilogx(freq3,abs(sig3),"Linewidth",1);
    xlim([200,xrange]);
    ylim([0,maximum]);
    xticks([256,512,1024,2048,4096]);
    yticks([]);

    hold off

end



%% Global strategy parameters

function [audioStep,freqs] = main_call(i,div, gain,leading, plotElec, fold,wavfile,
nft, wide)

    if div>1
        filename = ['slice' num2str(i) '.wav'];
        folder = append(fold,num2str(div));
        if leading
            folder = append(folder,'_lead');
        end
    else
        folder = 'Demo';
        filename = wavfile;
    end

    par_strat = struct( ...
        'fs', 17400, ...
        'nFft', nft, ...
        'nHop', 20,  ...
        'nChan', 15,  ...
        'startBin', 6,  ... % ignore every bin lower than 6
        'nBinLims', [2, 2, 1, 2, 2, 2, 3, 4, 4, 5, 6, 7, 8, 10, 56], ...
        'window', 0.5*(blackman(nft) + hanning(nft)), ...
        'pulseWidth', 18, ...
        'verbose', 0, ...
        'wavFile', [folder filesep filename]...
        );
    %     'wavFile', ['Sounds' filesep 'AzBio_3sent_65dBSPL.wav']...
    %% Setting up parameter structures for each subsequent function call
    % src = ReadWavUnit(strat, 'SRC', 'Sounds\AzBio_3sent.wav');
    par_readWav = struct( ...
        'parent', par_strat, ...
        'tStartEnd', [], ...
        'iChannel', 1 ...
        );

    % mix = AudioMixerUnit(strat, 'MIX', 1, 0, 'rel');   % 1 input, no re-scaling,
assuming correct scaling in input wav file (parameters: 0 dB rel. to input)
```

```
    % (.. single input, transparent gain -> omitted ..)

    % pre = HarmonyPreemphasisUnit(strat, 'PRE');             % pre-emphasis filter
    par_pre = struct(...
       'parent', par_strat, ...
       'coeffNum', [0.7688 -1.5376 0.7688], ...  % numerator coefficients
       'coeffDenom', [1 -1.5299 0.5453]     ...  % denominator coefficients
       );

    % agc = DualLoopTdAgcUnit(strat, 'AGC');                  % AGC
    par_agc = struct(...
        'parent', par_strat, ...
        'kneePt', 4.476, ... % compression threshold [log2]
        'compRatio', 12, ... % compression ratio above knee-point (in log-log space)
[> 1] [12]
        'tauRelFast', -8 / (17400 * log(0.9901)) * 1000, ... % fast release time
const [ms] [46.21]
        'tauAttFast', -8 / (17400 * log(0.25)) * 1000, ...   % fast attack time const
[ms] [0.33]
        'tauRelSlow', -8 / (17400 * log(0.9988)) * 1000, ... % slow release time
const [ms] [382.91]
        'tauAttSlow', -8 / (17400 * log(0.9967)) * 1000, ... % slow attack time const
[ms] [139.09]
        'maxHold', 1305, ...    % max. hold counter value [int >= 0] [1305]
        'g0', 6.908, ...        % gain for levels < kneepoint [log2] [6.908; approx =
41.6dB]
        'fastThreshRel', 8, ... % relative threshold for fast loop [dB] [8]
        'cSlowInit', 0.5e-3, ...   % initial value for slow averager, 0..1, [] for
auto; default: 1
        'cFastInit', 0.5e-3, ...   % initial value for fast averager, 0..1, [] for
auto; default: 1
        'controlMode', 'naida', ... % how to use control signal, if provided on port
#2? ['naida' / 'direct'] ['naida']
        'clipMode', 'limit', ... % output clipping behavio ['none' / 'limit' /
'overflow'] ['none']
        'decFact', 8, ...       % decimation factor (i.e. frame advance)
        'envBufLen', 32, ...    % buffer (i.e. frame) length for envelope computation
        'gainBufLen', 16, ...   % buffer length for gain smoothing
        'envCoefs', [-19,  55,   153,  277,  426,  596,    784,  983,   ...  %
tapered envelope data window
                     1189, 1393, 1587, 1763,  1915, 2035,  2118, 2160,  ...
                     2160, 2118, 2035, 1915, 1763, 1587,   1393, 1189,  ...
                     983,  784,  596,  426,   277,  153,  55,   -19 ] / (2^16) ...
        );

    % wb = WinBufUnit(strat, 'WB');                           % buffering and
windowing
    par_winBuf = struct( ...
        'parent', par_strat, ...
        'bufOpt', []  ...
        );

    % fftfb = FftFilterbankUnit(strat, 'FFT');                % FFT
    par_fft = struct( ...
```

```
        'parent', par_strat, ...
        'combineDcNy', false, ...           % Combine DC and Nyquist bins into single
complex 1st bin?
        'compensateFftLength', false, ...   % Divide FFT coefficients by nFft/2?
[boolean]
        'includeNyquistBin', false ...      % Return bin #nFft/2+1 in output?
[boolean]
        );

    % env = HilbertEnvelopeUnit(strat, 'HILB');              % Hilbert envelopes
    par_hilbert = struct( ...
        'parent', par_strat, ...
        'outputOffset', 0, ...      % scalar offset added to all channel outputs;
[log2] [0] Use with caution!
        'outputLowerBound', 0, ...  % lower bound applied to output (after offset)
[log2] [0]
        'outputUpperBound', Inf ... % lower bound applied to output (after offset)
[log2] [Inf]
        );

    % engy = ChannelEnergyUnit(strat, 'ENGY', 2);            % channel energies (for
noise reduction SNR estimation); 2 inputs (to account for AGC gain)
    par_energy = struct( ...
        'parent', par_strat, ...
        'gainDomain', 'linear' ...  % domain of gain input (#2)
['linear','db','log2']
        );

    % nr = NoiseReductionUnit(strat, 'NR', 1, 'log2', false);     % noise reduction;
'log2' makes gain output commensurable with Hilbert envelopes
    par_nr = struct( ...
        'parent', par_strat,  ...
        'gainDomain', 'log2', ...   % domain of gain output on port 2 (if applicable)
['linear','db','log2'] ['linear']
        'tau_speech', 0.0258, ...   % time constant of speech estimator [s] [0.0258]
        'tau_noise', 0.219, ...     % time constant of noise estimator [s] [0.219]
        'threshHold', 3, ...        % hold threshold (onset detection criterion) [dB,
> 0] [3]
        'durHold', 1.6, ...         % hold duration (following onset) [s] [1.6]
        'maxAtt',  -12, ...         % maximum attenuation (applied for SNRs <=
snrFloor) [dB] [-12]
        'snrFloor', -2, ...         % SNR below which the attenuation is clipped [dB]
[-2]
        'snrCeil', 45, ...          % SNR above which the gain is clipped  [dB] [45]
        'snrSlope', 6.5, ...        % SNR at which gain curve is steepest  [dB] [6.5]
        'slopeFact', 0.2, ...       % factor determining the steepness of the gain
curve [> 0] [0.2]
        'noiseEstDecimation', 1, ...    % down-sampling factor (re. frame rate) for
noise estimate [int > 0] [1]
        'enableContinuous', false, ...  % save/restore states across repeated calls
of run [bool] [false]
        'initState', struct('V_s', -30 * ones(15,1), 'V_n', -30 * ones(15,1)) ...
% initial state
        );
```

```
    % gapp = ElementwiseUnit(strat, 'GAPP', 2, @plus, true);  % NR gain application:
element-by-element sum of 2 input;
    %  (no corresponding parameter struct, addition handled in the function call
section below)

    % spl = SpecPeakLocatorUnit(strat, 'SPL');                % channel peak
frequency and target location estimation
    par_peak = struct( ...
        'parent', par_strat, ...
        'binToLocMap', [zeros(1,6), 256, 640, 896, 1280, 1664, 1920, 2176, ...   % 1
x nBin vector of nominal cochlear locations for the center frequencies of each STFT
bin
                        2432, 2688, 2944, 3157, 3328, 3499, 3648, 3776, 3904, 4032,
...   % as in firmware; values from 0 .. 15 (originally in Q9 format)
                        4160, 4288, 4416, 4544, 4659, 4762, 4864, 4966, 5069, 5163,
...   % corresponding to the nominal steering location for each
                        5248, 5333, 5419, 5504, 5589, 5669, 5742, 5815, 5888, 5961,
...   % FFT bin
                        6034, 6107, 6176, 6240, 6304, 6368, 6432, 6496, 6560, 6624,
...
                        6682, 6733, 6784, 6835, 6886, 6938, 6989, 7040, 7091, 7142,
...
                        7189, 7232, 7275, 7317, 7360, 7403, 7445, 7488, 7531, 7573,
...
                        7616, 7659, 7679 * ones(1,53)] / 512 ...
        );

    % csw = CurrentSteeringWeightsUnit(strat, 'CSW');         % current steering
weights based on target location
    par_steer = struct( ...
        'parent', par_strat, ...
        'nDiscreteSteps', 9, ...   % nr. of discretization steps  [int >= 0] [9]; 0 -
> no discretization
        'steeringRange', 1.0 ...  % steering range between electrodes [0..1] [1.0]
        );

    % csynth = CarrierSynthesisUnit(strat, 'CSYNTH');         % synthesize electrode
carrier signal at FT rate (temporal fine structure)
    par_carrierSynth = struct( ...
        'parent', par_strat, ...
        'fModOn', 0.5,  ... % peak frequency up to which max. modulation depth is
applied [fraction of FT rate] [0.5]
        'fModOff', 1.0, ... % peak frequency beyond which no modulation is applied
[fraction of FT rate] [1.0]
        'maxModDepth', 1.0, ... % maximum modulation depth [0.0 .. 1.0] [1.0]
        'deltaPhaseMax', 0.5 ...% maximum phase rotation per FT frame [turns] [0.5]
        );                      % Set <= 0.5 to avoid aliasing for fPeak > FT_rate/2

    % map = F120MappingUnit(strat, 'MAP');                    % combine envelopes and
carriers, and map to stimulation current amplitude
    par_mapper = struct( ...
        'parent', par_strat, ...
        'mapM', 500 * ones(1,16), ...       % M levels [uAmp]
```

```
        'mapT', 50  * ones(1,16), ...       % T levels [uAmp]
        'mapIdr', 60 * ones(1,16), ...      % IDRs  [dB]
        'mapGain', 0 * ones(1,16), ...      % channel gains [dB]
        'mapClip', 2048 * ones(1,16), ...   % clipping level [uAmp] [2048]
        'chanToElecPair', 1:15, ...         % 1 x nChan vector defining mapping of
logical channels to electrode pairs (1 = E1/E2, ...) [] [1:nChan]
        'carrierMode', 1 ...                % carrierMode - how to apply carrier [0 -
no carrier, 1 - to input, 2 - to output] [1]
        );

    % egram = F120ElectrodogramUnit(strat, 'EGRAM', true);
    par_elgram = struct(...
        'parent', par_strat, ...
        'cathodicFirst', true, ...  %  start pulse with cathodic phase? [bool]
        'channelOrder', [1 5 9 13 2 6 10 14 3 7 11 15 4 8 12], ... % default F120
staggering order [DO NOT MODIFY]
        'enablePlot', plotElec, ...     %  plot electrodogram? [bool]
        'colorScheme',2,...
        'outputFs', 55556 ...     %  output sampling frequency [Hz]
        );

    par_validate = struct(...
        'parent',par_strat,...
        'lengthTolerance',50,...                % maximum allowable difference
between validation file and submitted data
        'saveIfSimilar',true,...                % force saving files even if result
is similar to default processing [FOR TESTING]
        'differenceThreshold',1,...             % minimum current difference across
time for each channel
        'maxSimilarChannels',8,...              % max number of similar channels
allowed before preventing save
        'elgramFs',par_elgram.outputFs,...      % match electrodogram sampling
frequency
        'outFile',''...
        );

    par_vocoder = struct(...
        'parent',par_strat,...,
        'audioFs',48000,...
        'saveAudioOutput',false,...
        'audioOutputFile','' ...
        );



%% Function calls
% ( strat.connect(block_1, block_2), ..., strat.run() )
%tic

    sig_smp_wavIn                           = readWavFunc(par_readWav); % read wav
input; assumes correct scaling of 111.6 dB SPL peak full-scale in wav input

    sig_smp_wavPre                          = tdFilterFunc(par_pre, sig_smp_wavIn); %
pre-emphasis
```

```
    [sig_smp_wavAgc, sig_smp_gainAgc]       = dualLoopTdAgcFunc(par_agc,
sig_smp_wavPre); % AGC
    sig_frm_audBuffers                      = winBufFunc(par_winBuf, sig_smp_wavAgc);
% buffering
    sig_frm_fft                             = fftFilterbankFunc(par_fft,
sig_frm_audBuffers); % STFT

    sig_frm_hilbert                         = hilbertEnvelopeFunc(par_hilbert,
sig_frm_fft); % Hilbert envelopes
    sig_frm_energy                          = channelEnergyFunc(par_energy,
sig_frm_fft, sig_smp_gainAgc); % channel energy estimates
    sig_frm_gainNr                          = noiseReductionFunc(par_nr,
sig_frm_energy); % noise reduction
    sig_frm_hilbertMod                      = sig_frm_hilbert + sig_frm_gainNr; %
apply noise reduction gains to envelopes

    %   sub-sample every third FFT input frame
    sig_3frm_fft = sig_frm_fft(:,3:3:end);
    %[sig_3frm_peakFreq, sig_3frm_peakLoc,specFreqs]   =
specPeakLocatorFunc(par_peak, sig_3frm_fft,i); % peak frequency and location
estimation
    [sig_3frm_peakFreq, sig_3frm_peakLoc,specFreqs]   = specPeakLocatorFunc(par_peak,
sig_3frm_fft, wide); % peak frequency and location estimation



    %   up-sample back to full frame rate (and add padding for skipped initial
frames)
    sig_frm_peakFreq = repelem(sig_3frm_peakFreq,1,3);
    sig_frm_peakFreq = [zeros(size(sig_frm_peakFreq, 1),2), sig_frm_peakFreq];
    sig_frm_peakFreq = sig_frm_peakFreq(:,1:size(sig_frm_fft, 2));
    sig_frm_peakLoc = repelem(sig_3frm_peakLoc,1,3);
    sig_frm_peakLoc = [zeros(size(sig_frm_peakLoc, 1),2), sig_frm_peakLoc];
    sig_frm_peakLoc = sig_frm_peakLoc(:,1:size(sig_frm_fft, 2));

    sig_frm_steerWeights                    = currentSteeringWeightsFunc(par_steer,
sig_frm_peakLoc);  % current steering, based on peak location
    [sig_ft_carrier, sig_ft_idxFtToFrm]     = carrierSynthesisFunc(par_carrierSynth,
sig_frm_peakFreq); % carrier synthesis, based on peak frequencies
    sig_ft_ampWords                         = f120MappingFunc(par_mapper,
sig_ft_carrier, ... % combine envelepes, carrier and current steering weights,
compute current outputs
                                                sig_frm_hilbertMod,
sig_frm_steerWeights, sig_ft_idxFtToFrm);

    elgram = f120ElectrodogramFunc(par_elgram, sig_ft_ampWords, sig_smp_wavIn);  %
plot electrogram
    %saved = validateOutputFunc(par_validate,elgram);

    [audioStep,audioFs] = vocoderFunc(par_vocoder,elgram);
    freqs=specFreqs;
end
%sound(audioOut,audioFs);
```

```
This script finds the electrode firing order.

chans = [1,2,2,3,6,7,8,12,13,14,15,15,15,15,15];

disp("starting array:");
disp(chans);
firingorder(chans,2,100);
firingorder(chans,3,100);


function firingorder(chans,split,amount)
avgs = zeros(amount,1);
has_ones = 0;
for k = 1:amount
    hasd = false;
    if length(chans)==15
        chanIndeces = randperm(15);
        randchans = chans(chanIndeces);
        %disp("randomized order");
        %disp(randchans);

    if split == 3

        chans1 = randchans(1:5);
        chans2 = randchans(6:10);
        chans3 = randchans(11:15);
        [ord_chans1, d1] = maximizeDistance(chans1);
        [ord_chans2, d2] = maximizeDistance(chans2);
        [ord_chans3, d3] = maximizeDistance(chans3);
%         disp(ord_chans1);
%         disp(ord_chans2);
%         disp(ord_chans3);
    else

        chans1 = randchans(1:8);
        chans2 = randchans(9:15);
        [ord_chans1, d1] = maximizeDistance(chans1);
        [ord_chans2, d2] = maximizeDistance(chans2);
%         disp(ord_chans1);
%         disp(ord_chans2);
    end

%         ord_chans = [ord_chans1 ord_chans2];

        good2 = false;

        while ~good2
            if abs(ord_chans2(1)-ord_chans1(5)) < 3
                ord_chans2 = circshift(ord_chans2,1);
            else
                good2 = true;
            end
            avgs(k) = (d1+d2)/15;
        end
```

```
        if split == 3
            good3 = false;
            while ~good3
                if abs(ord_chans3(1)-ord_chans2(5)) < 3
                    ord_chans3 = circshift(ord_chans3,1);
                else
                    good3 = true;
                end
            end
            ord_chans = [ord_chans1 ord_chans2 ord_chans3];
            avgs(k) = (d1+d2+d3)/15;
        end
        ord_chans = [ord_chans1 ord_chans2];

        %disp("maximized distance order:");
        %disp(ord_chans);
        %disp("distance: " + (d1+d2) + " , average distance: " + ((d1+d2)/15));


        distances = zeros(1,15);

        for i = 1:length(ord_chans)
            j = i+1;
            if i == length(ord_chans)
                j = 1;
            end
            distances(i) = abs(ord_chans(i)-ord_chans(j));
            if distances(i) < 2
                hasd = true;
            end

        end
%         disp("consecutive distances:");
%         disp(distances)
        if hasd
            has_ones = has_ones +1;
        end
    end
end
avg_tot = 0;
for k = 1:amount
    avg_tot = avg_tot+avgs(k);
end

fprintf("\n");
disp("Amount of split arrays: "+split);
disp("Results:");
disp("Average distance: " + avg_tot/amount)
disp("Amount that have ones or less: " + has_ones+"/100");;
end
```

```
%%
function [orderedArray,distance] = maximizeDistance(array)
    % Initialize variables
    n = length(array);
    maxDistance = -1; % Initialize maxDistance to a small value
    bestPermutation = array; % Initialize bestPermutation to the original array
    i = 1;
    thisperm = array;
    while i < factorial(n)
        thisperm = nextperm(thisperm);
        %disp(thisperm);
        thispermplus = [thisperm thisperm(1)];
        currentDistance = sum(abs(diff(thispermplus)));
        if (currentDistance > maxDistance)
            bestPermutation = thisperm;
            maxDistance = currentDistance;
            %disp(currentDistance);
            %disp(thisperm);
        end
        i = i+1;
    end

    distance = maxDistance;
    orderedArray = bestPermutation;
end
```

References

Brodie, A., Smith, B., & Ray, J. (2018). The impact of rehabilitation on quality of life after hearing loss: a systematic review. *Eur Arch Otorhinolaryngol 275*, 2435-2440. doi:10.1007/s00405-018-5100-7

Brownell, W. (1997). How the ear works - nature's solutions for listening. *The Volta Review 99(5)*, 9-28.

Dorman, M., & Parkin, J. (2015). The role of the Utah Artificial Ear project in the development of the modern cochlear implant. *Cochlear Implants Int.* doi:10.1179/1467010015Z.000000000246

Eshraghi, A., Nazarian, R., Telischi, F., Rajguru, S., Truy, E., & C., G. (2012). The Cohlear Implant: Historical Aspects and Future Prospects. *Anat Rec. 295*, 1967-1980. doi:10.1002/ar.22580

Nogueira, W., Litvak, L., Edler, B., Ostermann, J., & Büchner, A. (2009). Signal Processing Strategies for Cochlear Implants Using Current Steering. *EURASIP J. Adv. Sig. Proc.*

Vita

The author was born in Metairie, LA in 1981. After obtaining his undergraduate degree in General Physics from University of New Orleans in 2020, he began a Master of Arts in Teaching (MAT) program at the same university, but before completing it, he returned to the physics department to pursue a Master of Science in Applied Physics (MSAP). Under advisement from Dr. Kendal Leftwich of UNO, he joined the research team of Dr. Yoshida and Dr. Coleman of Southeastern Lousiana University in 2023.