COSC 4F00 Project:
Sakai Group (Group 1)

A Strategy for Design and Simulation of Nerve Signal Processing

Name:	**St. ID:**	**St. #:**
Owen Brown	ob10rp	4838488
Daniyal Muhammad	dm13hd	5528930
Edward Shin	es13pw	5517065
Chang Ding	cd12jb	5275821
Alexander Shuev	as12ga	5199393
Michael Alsbergas	ma11xr	5104112

Project Completeness - Checklist

Finished

- Signal representation and generation
- Neuron representation and generation
- Wires representation and generation
- Wire connection
- Simple signal comparison using moving window
- Basic GUI when testing
- Report Generation

Incomplete

- Neuron representation; using 2-D array instead of 3-D array to save space
- Use of important wires with duplicated needles to increase connection ratio
- Signal comparison using method described in the paper
- Minimum number of pulses in a signal
- Control of noise
- Error handling
- GUI Design

Design Document

Problem Definition

There are approximately 100 billion nerve endings in the average human brain, which acts as the command center for our bodies. Efferent and afferent nerves facilitate the communication to and from various parts of our bodies, affording humans gifts such as sight, hearing and movement. Damage to these nerves eliminates the freedoms that are enjoyed by humans. The brain may still want to communicate with a lost limb or sensory receptor, this is why 60 - 80% of individuals with amputations often encounter the sensation known as having a phantom limb[1]. What if it were possible to map these communication signals intended for the phantom limb to a prosthesis? The implications of such a technology would reach further than applications to amputees. Anyone with nerve damage causing impairment (quadriplegics for example) would be given some degree of their previous functionality back.
Let us consider the aforementioned applications of this technology. For an amputee, consider the particular case of a missing hand. A prosthesis would be required to map incoming nerve signals to computer operations in order to effectively perform the appropriate actions.

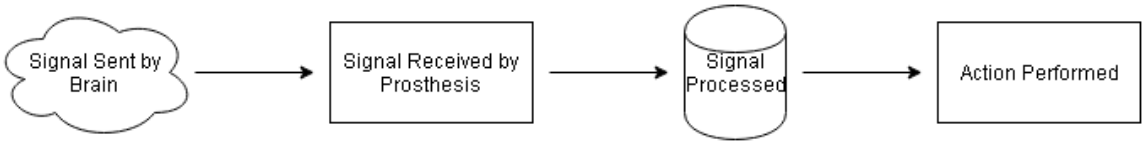


Fig. 1 - Nerve signal to prosthesis

Now consider a damaged area of the nervous system. The signal would be sent but not received by the intended target because of this obstruction (dead nerves). Consider a wire bridging the healthy nerves and affording the deliverance of such a signal.

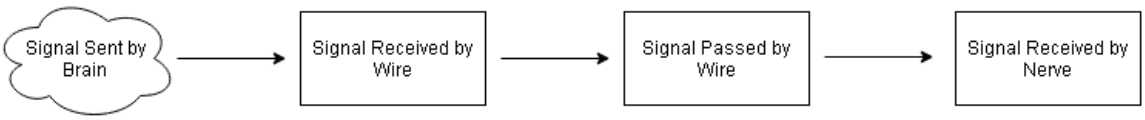


Fig. 2 -Wire bridging two healthy nerves

In both cases we consider a connection from wire to nerve, some analysis of the signal and use of this information to determine a result. These ideas will be explored further in terms of simulation and possible design challenges. In particular, we will discuss three key areas: connection representation, signal representation & connection evaluation.

[1] Sherman, R. A., Sherman, C.J. & Parker, L. (1984). "Chronic phantom and stump pain among American veterans: Results of a survey"

Connection Design

In this section we will define the simulation representation of the physical connections between the microbrush and the nerve . This is done prior to the signal matching phase.

Inputs

There are three variables that affect the calculated connection ratios.

1. Number of Neurons - The number of neurons available for connections. This in turn determines the number of axons the microbrush will be able to connect to.
2. Number of Wires - The number of wires in the microbrush used to connect to the nerve. The number of wires is usually smaller than the number of neurons.
3. Seed Value - The seed value used in the three random number generators used; one of uniform distribution and two of exponential distribution. A value of 0 will generate a random number using the system time, whereas any other value will use that specific number.

The Trials input, shown in Fig. 4, represents the number of iterations using the same parameters but different seeds. If a non-zero seed is provided, it is incremented by 1 for each trial. This GUI was designed for testing the connection representation and is not present in the final simulation code.

The connection testing program outputs a report detailing the variables used connection ratios (the number of wires from the microbrush that have successfully connected with an axon divided by the number of total axons in the nerve), its associated variables, as well as produce a text file that will provide a visual representation as to how the connections have been made. Fig. 3 is an excerpt from the output file providing a visual representation of the connection. An X denotes a successful connection to an axon, while an O represents an unconnected axon. Below is an example of how a type of connection could be made by inserting 7 wires into a nerve with 37 axons.

```
    O O O O
  O O O O O O
O O O O O O O
O O O X X O O
  O O X X X O
    O O X X O
      O O O O
```

Fig. 3 - A visual representation of a wire to neuron connection.

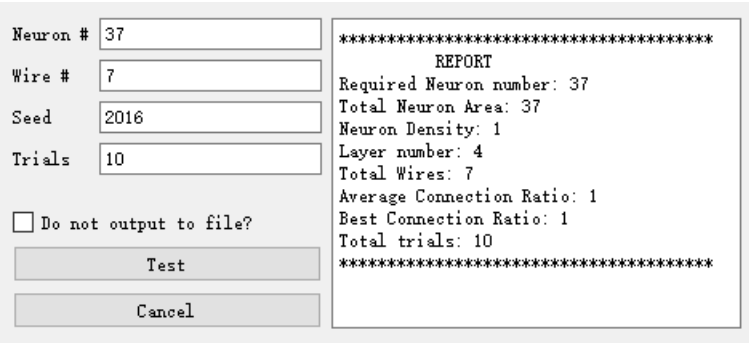


Fig. 4 - A report generated after simulation is completed.

Hexagon Representation

To simulate the interaction between the wires and the axons, the nerve is represented by a hexagonal grid. This grid is stored in the simulation as a 3D array. Fig. 5 demonstrates the coordinate system used to locate a specific hexagon within the grid.

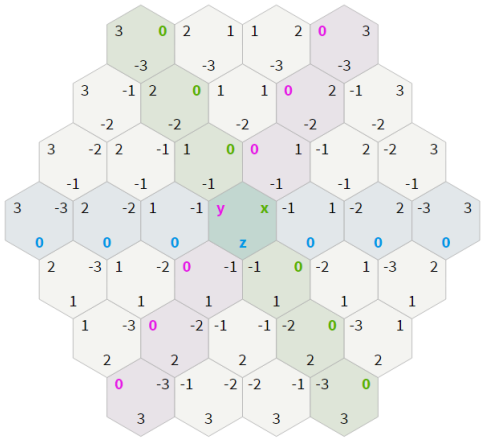


Fig. 5 - The Hexagonal Grid Coordinate System[2]

The grid is organized into layers, a group of hexagons that surround another group of hexagons. A hexagon has three coordinates: (X, Y, Z). The centre hexagon, at point (0, 0, 0), is classified as Layer 1. All surrounding hexagons of Layer 1 would be grouped into Layer 2. All surrounding hexagons of Layer 2 would be grouped into Layer 3. This pattern continues as more hexagons are introduced into the grid.

One important property of the grid is for all hexagons, the sum of its coordinates must equal to zero (i.e. X + Y + Z = 0). Another property is that for all layers, the absolute value of all hexagon coordinates will be less than the layer number. For example, the coordinates for all cells in Layer 2 are (0, 1, -1), (-1, 1, 0), (-1, 0, 1), (0, -1, 1), (-1, 1, 0), and (1, 0, -1). Every coordinate in all the hexagons in the layer has an absolute value of less than 2.

The grid is initialized layer by layer. Fig. 6 shows the code for creating a hexagonal representation of the nerve with a given number of layers, starting at the center most layer. Fig. 7 is a visual representation of how the initialization procedure works around the center point of the hexagon. The same technique is used to create the micro wire brush.

[2] Patel, A. (2016). Hexagonal Grids. Redblobgames.com. Retrieved 11 March 2016, from http://www.redblobgames.com/grids/hexagons/

```
for (int i = 0; i < result.layers; i++){
        for (int l = -i + centerPoint.coorX; l <= i + centerPoint.coorX; l++){
            for (int m = -i + centerPoint.coorY; m <= i + centerPoint.coorY; m++){
                for (int n = -i + centerPoint.coorZ; n <= i + centerPoint.coorZ; n++){
                    if (((l + m + n) == 0) && abs(l) < result.layers&& abs(m) < result.layers&& abs(n) < result.layers){
                        //if x+y+z = 0 and x, y, and z are within our range.
                        //we will create our neuron here.
                    }
                }
            }
        }
    }
```

Fig. 6 - C++ Code for creating hexagons in the hexagonal grid.

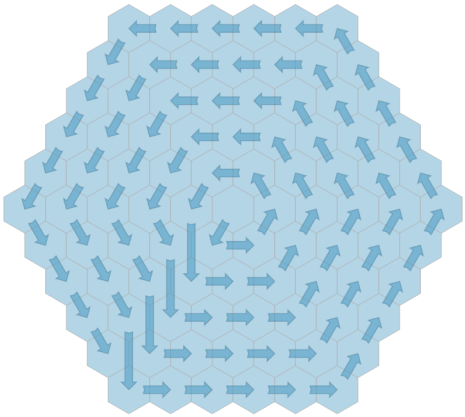


Fig. 7 - The path taken by the code to generate the hexagonal grid.[3]

It is not possible to have negative indices for an array, so an offset is used for the array index, where offset = layers – 1.
Representing the connection using a 3D array is very straightforward. However, a drawback of this design is the excess memory requirements, as the vast majority of the elements in the 3D array will not be used. For an n layer hexagon, the number of unused elements in the array would be:

$$(2n-1)^3 - 3(n-1)n - 1$$

Printing the connection

Printing or displaying a 3D array on paper or screen is challenging, so the following idea is used to convert the 3D array to a 2D array just for display:

X = (oldX + offset)
Y = (oldX - oldY) + (2 * offset)

Simulating a Connection

Using a random number generator (RNG) of Geometric distribution, the address of the neuron with which the center-most wire in the microbrush will connect is generated. This distribution is

[3] Patel, A. (2016). Hexagonal Grids. Redblobgames.com. Retrieved 11 March 2016, from http://www.redblobgames.com/grids/hexagons/

used since as modern medicine advances, surgeries are performed with the use of robotic aids, allowing surgeons to control the movements of mechanical arms while operating at a much larger scale. This allows for greater precision than the bare human hand can accomplish. For connecting the microbrush to the nerve, it can be assume that such a device will be used, allowing for the majority of connections to be made near the center of the nerve, while leaving the possibility of an imperfect initial connection.

Unconnected Wires

Wires in the microbrush have two states; they may be connected to an axon or not. One situation where a wire is disconnected is if it strikes intercellular fluid. Another is in event of an imperfect connection by the surgeon, where wires that fall outside of the nerve structure will also be considered as unconnected. The latter is handled when the connection is made by checking if the coordinate exists in the hexagonal nerve. If the coordinate does not exist, the wire is considered as having failed to connect to an axon. The first situation is simulated by providing each wire with a probability of a successful connection. It is understood that in a nerve, the small hexagon number could be 1, 7, 19, 37, 61, 91, and so on (partially completed layers are not used). If the nerve have 15 axons in a hexagonal representation with 19 elements, the neuron density is 15/19 = 0.79. Therefore, each wire will have a (1-0.79) = 0.21 chance to hit intercellular fluid.

Improvement

In using a 3D array for representing a hexagonal nerve, the following principle holds true with respect to the coordinates: x+y+z = 0. This can be simplified to z = 0-x-y, so that Z does not need to be used explicitly. This affords the ability to map the hexagonal structure to a 2D array and compute the third coordinate when necessary, resulting in a significant reduction in memory requirements for the simulation. Fig. 8 is a visual representation of this change.

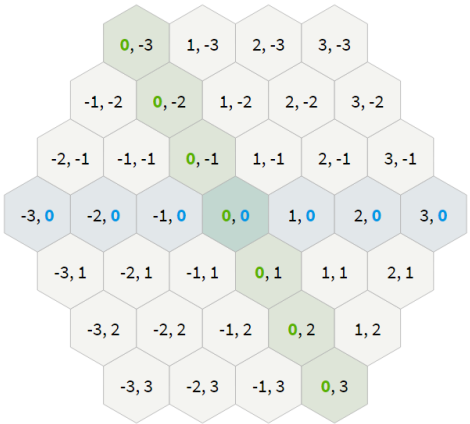


Fig. 8 - Hexagonal Mapping using X,Y only.

Signal Design

This section explores the generation and modelling of a signal within the simulation. The goal is to create, for each axon within the nerve, a signal of a user defined length (L) representing the sample time of the signal. Two sets of data are built representing the alternating pulse and

silence times for the duration of L. Combining this data, and applying a some noise factor produces the final model of a sampled signal. A noise factor is applied to the generated signal to simulate the noise from the equipment used to read the signal.

Signal Representation

As outlined in the paper *Fundamentals of Cochlear Nerve Signal Processing: Towards a New Generation of Cochlear Implant Devices* (Vlad Wojcik, W. Gregory Wojcik), a rectangular pulse signal s(t) can be represented in the following format:

$$s(t) = < t_0, t_1, t_2, t_3, \ldots t_{2n-1} >$$

In addition, consider all intervals (two distinct consecutive elements) of s(t) starting at t0:

$$s(t) = \begin{cases} 1 \text{ for } t_i \le t < t_{i+1} \text{ where i is even;} \\ 0 \text{ for } t_i \le t < t_{i+1} \text{ where i is odd.} \end{cases}$$

This can be used to map the representation of the signal to a collection object of some type and use the index to determine whether it is a pulse or silence time.

Generating Reference Signals

It is assumed that a collection of signal data for the healthy nerve is available for reference. In the real world, the initial signal patterns would be collected from the live nerve with the patient being asked to perform specific actions. This would provide signals for each neuron in the nerve to use as a reference.

This is simulated by taking a RNG (random number generator), uniformly distributed over the range 0 to 1, and producing two values; v_P and v_s. The first value, v_P, represents the mean of the pulse length distribution, and the second value, v_s, represents the mean of the silence length distribution.

Next, two distinct RNG's of exponential distribution are initialized with means v_P and v_S. These are used to create pulse and silence times in alternating fashion until they cumulatively meet or exceed the user defined signal length L.

A noise factor is then applied across the collection of interleaved pulse and silence times. To create the noise factor, we use an RNG with a normal distribution and a mean of 0. For each pulse and silence time that exists up to L, a noise factor is generated and added to the existing pulse/silence length. If the pulse or silence, once the noise is applied, has a negative length, it is considered to be "drowned out" by the noise, and the pulses or silences that surround it are merged. Using this strategy one signal is generated per axon in the nerve.

Matching Signal to Neuron

Once the connection has been established, the next step is to match each wire in the microbrush with the neuron that it has connected to. This is done by listening on each wire for a signal and comparing it to the stored reference signals. In the simulation, the signal read in by the wire will have another randomly generated noise factor applied to it, creating a similar,

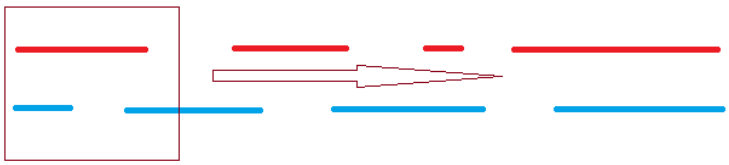
but not identical, signal to the reference signal for the neuron. The set distances are computed for this signal with all of the pre-recorded reference signals. Each wire is then paired with the neuron with which it has the smallest set difference in signal. This part is highly parallelizable as each wire on the micro-brush will require as many set difference computations as there are axons in the nerve.

Signal Comparison

Signal comparison will be conducted on two signals, represented as a zero-based collection of real numbers, where even indices are pulses and odd indices are silence. After generating the signal objects, the following patterns can be observed between the sent and received signals. Below are two examples of input and output signals where the red values represent pulse length and black values represent silence. Low levels of noise results in the received signals being similar to the original, while high levels of noise results in a significantly different signal being received from the source.
Example 1: Low levels of noise
Received: 6.79536, 0.05543, 1.670 , 5.3959, 1.86086
Sent: 6.14417, 0.1499, 1.68778, 5.47388, 1.83727
Example 2: High levels of noise
Received: 6.79536, 0.05543, 1.670 , 5.3959, 1.86086
Sent: 0.28778, 3.47388, 2.83727, 4.3215
First, the comparisons are made more consistent by ensuring that both signals being analyzed begin with a pulse at t0.
To visualise the process, the sent and received signals are shown below with the received signal in red and the sent signal in blue.

Fig. 9 - A visual look at the pulses of a sent and received signal.

A window of a user defined size (in terms of signal length) is used to compare the signals. This simulates the real time nature of the signal comparison. Unlike networking protocols, which use data packets, the human body works in real time, and using a window for signal comparison simulates real time signal comparison as opposed to waiting for the full signal to complete and then analysing the full signal. The displacement of the window, which represents the amount by which the window shifts between each comparison, is another user controlled variable. Only one window is used as we consider it as the overlap of two the two signals.



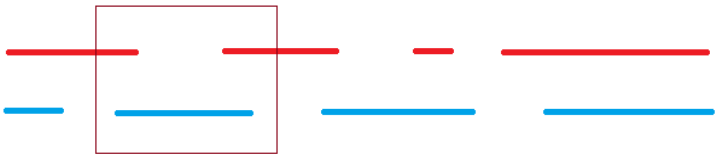


Fig. 10 - A sliding window comparing segments of one signal with another.

The distance between two signals in the window is computed as outlined in the next section. The total distance will be the sum of all the distances that are calculated in each window.

Pulse Distance

In each window the received signal and the sent signal are compared, using overlapping signal pulses as a measure of similarity. There are several possibilities for this approach.
One approach may be to take the set difference. This is calculated using the method presented in the paper *Fundamentals of Cochlear Nerve Signal Processing: Towards a New Generation of Cochlear Implant Devices* (Vlad Wojcik, W. Gregory Wojcik):

$$D(A, B) = \max \{\Delta(A, B), \Delta(B, A)\}$$

This provides a clear scoring mechanism for our signals.
Some nerves may be more important for us to connect to, thusly improving our connection if they are successfully connected to and having a negative impact if they are missed. This nerve importance factor NIF can be implemented as a number between 0 and 1, with our center nerve(s) having a NIF of 1. All surrounding positions would have a progressively smaller NIF based on their distance from the center.
Finally, to get an overall connection score the sum of all distances multiplied by the appropriate NIF would be returned. These values would be checked against a table of values built to determine whether the connection obtained is of an acceptable level.
Alternatively, the problem could be viewed from the perspective of information theory[4], attempting to find mutual information across noisy channels. Refer to Claude Elwood Shannon's work "A Mathematical Theory of Communication" for more information.

Final Analysis

Once the matching is completed, the number of wires in the microbrush paired to the neuron they were actually connected to can be computed. By running the simulation a number of times with different seed values using a combination of different values for number of wires in the microbrush, number of neurons in the nerve, signal lengths, window size, and window displacement, enough data can be collected to determine the optimal settings required for a nerve of a given size for an average correct matching of a given percent.

[4] Information Theory, https://en.wikipedia.org/wiki/Information_theory

References

[1] Patel, A. (2016). *Hexagonal Grids*. *Redblobgames.com*. Retrieved 11 March 2016, from http://www.redblobgames.com/grids/hexagons/

[2] (n.d.). Retrieved March 12, 2016, from https://en.wikipedia.org/wiki/Geometric_distribution

[3] Wojcik, V., Wojcik, G. W., & M.D. (2011). Fundamentals of cochlear nerve signal processing: Towards a new generation of cochlear implant devices.

Program Guide & Instructions

Representation of the Neurons

Classes

1/ singlePulse

- Represents the length and the state of the pulse / silence
- 1 represents pulse, 0 represents silence
- This class is located in cochlearProject.h

2/ singleSignal

- Represents a single “Signal” which may be held by the axon
- Uses a double to hold the length of the signal
- Uses vector<singlePulse> to hold the series of pulses/silences
- Pulses/Silences are generated by a random uniform real distribution engine

3/ singleNeuron

- Represents a single axon
- Uses a pointer to hold its signal data
- Uses a char to indicate its state
- X represents connected, O represents no connection, empty represents empty spot

Functions

1/ singleSignal::printSingleSignal()

- Prints one original signal and two modified signals
- Only prints the length of each pulse / silence
- Resulting order will be: pulse, silence, pulse silence…

2/ singleSignal::getPulse()

- Returns the vector that holds the pulses / silences

3/ singleNeuron:: Status getter and setter

- getStatus()
- setStatus()

Representation of a Wire

Classes

1/ singleWire

- Represents a single wire
- Uses coordinates to hold location of the wire
- Uses vector<singlePulse> to hold all pulses received from the axon

Functions

1/ singleWire:: Location getter and setter

- getCoordinate()
- setCoordinate()

2/ singleWire:: Pulses / silences getter and setter

- getPulses()
- setPulses()

3/ singleWire:: Prints the signal and location

- printReceivedSignal()
- printWireLocation()
- printSingleResultLocation()

Representation of the Microbrush

Classes

1/ wiresCollection

- Represents a collection of wires
- Uses a 1-D array to hold the wires

2/ neuronCollection

- Represents a collection of axons
- Uses a 3-D array to axons *Note: Space is wasted due to hexagon shape of nerve.
- Uses a 1-D array to hold the locations that have an axon

Functions

1/ wiresCollection::getWires(int i)

- Returns the i^{th} wire

2/ wiresCollection::printWiresLocation()

- Prints all the locations of wires

3/ wiresCollection::printWiresSignal()

- Prints the signal that the wire received

4/ neuronsCollection::convertToHex(int, int, int, int offset)

- Converts the normal index to the negative index, which represents the real location of the axon

5/ neuronsCollection::hexToMatrix(int, int, int, int offset)

- Converts the hex index to the index for printing

6/ neuronsCollection::convertToNormal(int, int, int, int offset)

- Converts the negative index to the normal index to be used in the array

7/ neuronsCollection::getResult()

- Returns the result of the connection

8/ neuronsCollection::getNeuronLocation(int i)

- Returns the location of the given axon

9/ neuronsCollection::getSingleNeuron(Coordinates)

- Returns the axon at a given location

10/ neuronsCollection::setValue(int, int, int, int offset, char)

- Sets the status of a given location

11/ neuronsCollection::printCurrentBoard()

- Prints the nerve

12/ neuronsCollection::connectWire(wiresCollection)

- Connects the wires to the nerve

Representation of the Connections

Classes

1/ Main

- Takes parameters from user
- Creates the nerve and wires
- Connects the wires
- Compares the signals

Functions

1/ main::compareSignal(vector<singlePulse>, vector<singlePulse, InputData)

- Compares two signals by creating small “windows” within the signal
- Calls compareWindows method once windows are created

2/ main::compareWindows(vector<singlePulse>, vector<singlePulse>, double startPoint, double endpoint)

- Compares two pieces of a signal from startPoint to endpoint
- Uses first pulse / silence of each piece

3/ main::isSameCoordinate(Coordinates, Coordinates)

- Returns true if two coordinates are the same

4/ main::printResult()

- Prints the report

Overhead

Classes

1/ Coordinates

- Represents the location of the axons / wires

2/ InputData

- Holds input data

3/ connectionData

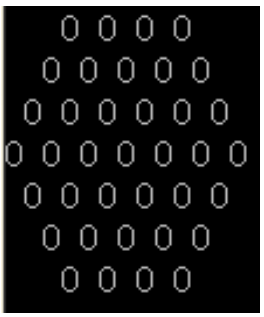
- Holds report data

Program Usage

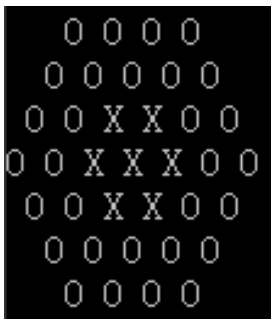
1. Import the program into Visual Studio 2013/2015
    - If you are unable to import, create a new project and copy and paste the .cpp & .h files, no external library is needed
2. Run the program
3. Enter 1 to use default parameters, otherwise, enter parameters of your own
    - Parameters needed:

i. Total axon number (int) – default 37
ii. Total wire number (int) – default 7
iii. Length of each signal (double) – default 20
iv. Seed used for random number generation (int) – default 2016
v. Window size (double) – default 5
vi. Window displacement (double) –default 4

Sample Output



Original Nerve



Connected Nerve

```
//Status of each Connected Nerve
Smallest distance for wire: (0, 0, 0); is 2.02028  It matches the Neuron: (-1, -2, 3);
Smallest distance for wire: (-1, 0, 1); is 6.39534  It matches the Neuron: (-3, 3, 0);
Smallest distance for wire: (-1, 1, 0); is 0.920352  It matches the Neuron: (-1, 1, 0);
Smallest distance for wire: (0, -1, 1); is 0.910759  It matches the Neuron: (0, -1, 1);
Smallest distance for wire: (0, 1, -1); is 0.343439  It matches the Neuron: (0, 1, -1);
Smallest distance for wire: (1, -1, 0); is 0.162099  It matches the Neuron: (1, -1, 0);
Smallest distance for wire: (1, 0, -1); is 2.52527  It matches the Neuron: (1, 0, -1);

//The number of wires that are connected and received the correct signal
Total Match is: 5


**********REPORT**********
Total Neurons: 37 Total Wires: 7
Unconnected Wire: 0 Connect Ratio: 1
Original Length of Each Signal: 20
Neurons Density (which is the connect probability): 1
Seed: 2016

Program runs: 0.158257 seconds
Press any key to continue . . .
```

Testing and Results

A battery of tests were conducted in the simulation to thoroughly demonstrate its performance and function. The purpose of these tests were not to find an optimal set of parameters nor to compare one set of parameters against another, so the tests were not wholly exhaustive. Since the implementation is incomplete, rigorous testing would serve little purpose as the results would not be representative of the design. However, even with the naive signal comparison implementation, which only compares the first pulse length in each window, the results are quite interesting.

A default set of parameters were used for each test with only the specific parameter being tested changed. The default parameters used were 60 wires in the microbrush, 1000 neurons, signal length of 100 units, a window comparison size of 25 units, and window increments of 10 units.

For each set of parameters tested, the simulation was run 20 times with different seed values to ensure a variety of random, yet repeatable, results. The results in the charts are the averages of the 20 runs.

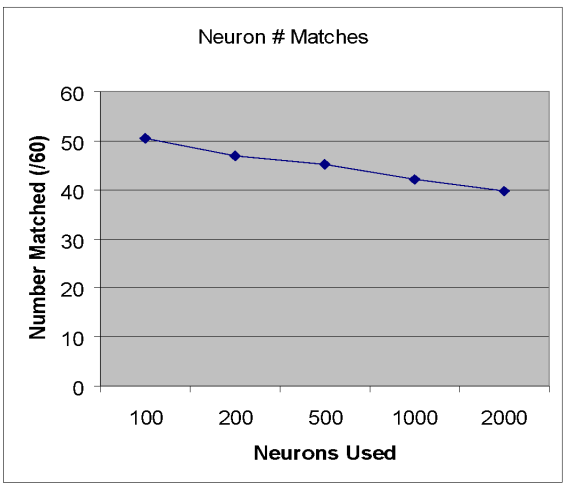


Fig. 1 - Results of Neuron Number Changes

The first parameter tested was the number of neurons in a nerve. As *Fig. 1* reveals, there was a decrease in the number of matched connections as the number of simulated neurons increases. This simply meant that the fewer neurons there are in a nerve, the more likely a match is to be found. This is logical, since the signal from the brush will be compared to more reference signals as the number of axons in the nerve increase. More reference signals means there is a higher chance that another neuron will have a similar characteristic signal, making it more challenging to find the best match. Outside of the simulation, the number of axons in a nerve will be an approximated constant depending on the nerve with which the connection will take place. Increasing the number of neurons 20 fold only yielded a 20% decrease in matches. Ultimately, resources should be spent trying to optimize other aspects of the procedure. This also demonstrates that our simulation can show meaningful results even using large numbers of neurons in a nerve.

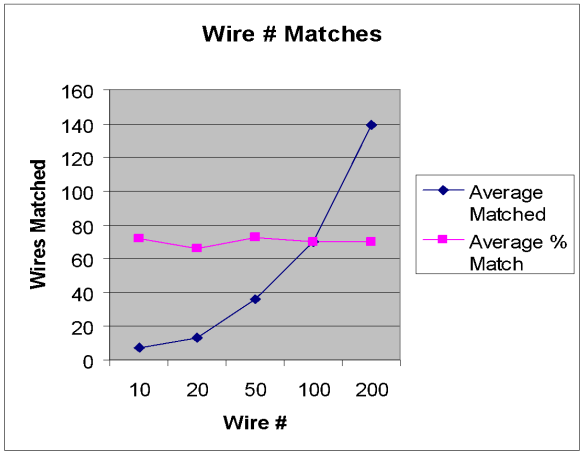


Fig. 2 - Results of Wire Number Changes

The number of wires being attached to a nerve was the next parameter to be tested. *Fig. 2* shows the outcome of the simulation with varying numbers of wires. It quite clearly shows that the more wires used in the simulation, the greater the number of positive matches made. The reason behind this is quite obvious as the more wires used, the more possible matches that can be made. For example; it's quite easy for 200 wires to get more matches than 10 wires as using 10 wires can yield a maximum of 10 matches.

However, the pink line in *Fig. 2* shows the number of matches made as a percentage of the maximum number of matches possible, i.e. the number of wires in the micro-brush. It shows there is very little change as the number of wires increase, meaning that using more wires did not make finding a matched connection more likely, it just meant that there are more attempts at matching connections.

Therefore, if the raw number of matches found is the final influence of a good connection, then increasing the number of wires used may be the most cost effective way of improving a connection. However, it does not make the connection more efficient, so other factors should be considered to optimize the procedure.

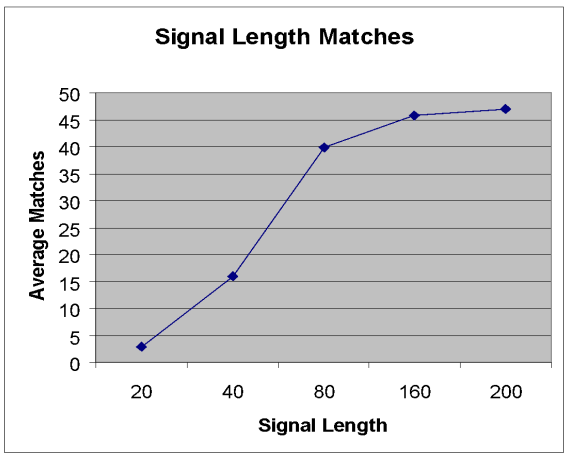


Fig. 3 - Results of Signal Length Changes

Modifying the length of a signal recorded from the nerve has yielded more promising results than the other two parameters, as shown in *Fig. 3*. It shows that as the signal length becomes shorter, the fewer the matches are accurately made. This is because, with short signals, there

is less data to compare and identify if the connection is a suitable match. The shape of the curve is also worth noting, as it appears to start converging at values higher than 160.

Given that the only significant costs to increasing this parameter is time, time to record a signal and increased computational time, thorough testing of this parameter is encouraged with a full signal comparison algorithm implemented to hone in on an optimal setting. Whether the optimal signal length varies based on the other parameters is something that will need to be tested as well.

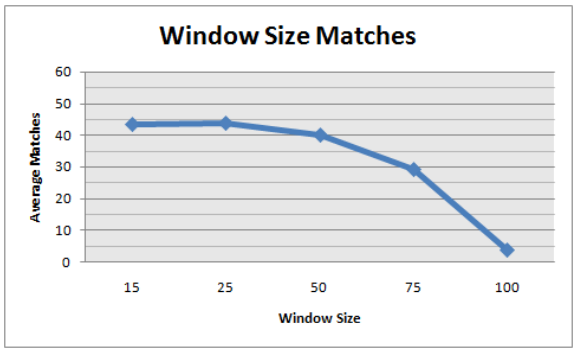


Fig. 4 - Results of Window Size Changes

Altering the size of the signal window frame, unlike the previous adjustment, yields more drastic results. As we can see in *Fig. 4*, as the window size increases, the average number of good matches reduces significantly.

The current signal comparison function is the culprit for this result. The implementation currently compares only the first pulse in a window, so regardless of how dissimilar the remainder of the window is, if the first pulse is similar, the difference value for that window will be low, i.e., good. The expected result is that, as the window size increases, the set distance calculation would become more representative of the closeness of two signals. This should result in the number of matches increasing as a result of increasing the window sizes. This is another variable that should be tested thoroughly with a complete implementation. Too large of a window will result in high computation times and too small will be inaccurate.

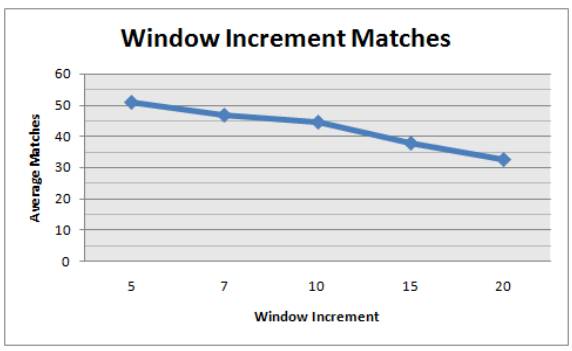


Fig. 5 - Results of Window Increment Changes

Fig. 5 shows that when the increment of the signal window frame changes, a similar declining trend is observed as was with window size, as shown in *Fig. 5*. The rate of decline is more linear rather than exponential.

With the larger window increments, although execution time decreases, we analyze fewer parts of the signal. This makes the effects of a bad connection more prevalent, thus making the overall connection a poor one.

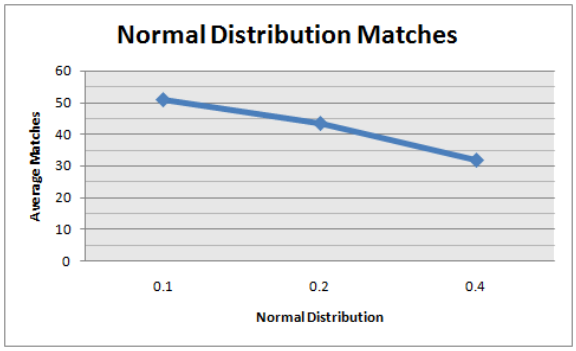


Fig. 6 - Results of Normal Distribution Changes for Noise

Increasing the standard deviation of the normal distribution used to compute the noise factor, as shown in *Fig. 6*, shows how efficient a connection is based on a range of values. The larger the standard deviation of the noise, the greater the difference between the reference signal and the same reference signal with a noise factor applied. As the signal with noise applied becomes more distant from its reference, the set difference will increase, making matching more challenging. The amount of noise in the received signal is not an absolute value and will depend on the technology used to measure the signals in the real world, so being able to simulate varying levels of noise allows for determining the levels noise which still produce acceptable results.

Conclusion

The results of testing demonstrate the need for optimization of input parameters with respect to the type of nerve the connection is made with. A larger number of wires will increase connection quality while also increasing the time required to compute the optimal outcome. Signal length shows diminishing returns after reaching a suitable length. Large window increments results in loss of information as larger sections of the signal are being disregarded with each increment and there are fewer set difference computations occurring for each signal pair being compared. The results for window size testing are unsuitable for drawing any conclusions as it highlights the limitations of the current implementation. With a full implementation, the results of this batch of testing would provide a solid base on which to build a more thorough testing platform. The goal should be to hone in on the optimal sets of values for the parameters for a particular nerve (e.g. a nerve with 30,000 axons should require a micro-brush with A wires, signal length of B units, a window size of C, and a window increment of D). The optimization will be required for a variety of nerve sizes. For each nerve size, optimal parameter sets will be required for a number of target connection ratios.