Master's Thesis

Artificial Neural Network for Neural Signal Processing

Author:
Alexandros Elio Vlissidis

Supervisor:
Professor John Taylor

Student Number: 129154107
Assessor: Dr. Paulo Rocha

May 3, 2017

Declaration

"I certify that I have read and understood the entry in the Student Handbook for the Department of Electronic and Electrical Engineering on Cheating and Plagiarism and that all material in this assignment is my own work, except where I have indicated with appropriate references. An electronic copy of this work has been uploaded to Moodle to allow submission to Turnitin for Plagiarism Detection."

Word Count: 10,157

Abstract

Velocity Selective Recording (VSR) has been proposed as a technique of extracting more information from implantable neural interfaces. Recent research has shown that a *Time Delay Neural Network* (TDNN) has the potential to outperform traditional VSR techniques, but requires a time consuming optimisation process. This paper describes a series of analyses performed in order to understand the operation of this model in VSR, in order to eliminate the need for optimisation and potentially improve its performance for low *Signal-to-Noise Ratio* (SNR) inputs.

This report focuses on time response analysis of this model, which revealed that it converges to a highly sophisticated version of a traditional method, called *delay-and-sum*. The key to the performance of the model lies in the timings of the neural signals from multiple channels, creating constructive and destructive interference patterns. It was also shown that the reason that the model fails with real data extracted from a rat, is that these recordings suffer from DC distortion and neural pulse interference, which severely affect the performance of this specific model.

Finally, the theories formed during the analysis of the results were validated by designing FIR filters to simulate the neural signal processing that the TDNN accomplishes. The designs did not reach the level of performance that the TDNN achieved, but important steps were made towards the final goal and future work is proposed.

Acknowledgements

I would like to thank Professor John Taylor and Dr. Ben Metcalfe who have co-supervised this project and have provided unlimited support by ensuring I have all the needed materials and literature, as well as by being always present when I needed help or advice.

Contents

Acronyms

ANN *Artificial Neural Network.*

AP *Action Potential.*

API *Application Programming Interface.*

AWGN *Additive White Gaussian Noise.*

CNS *Central Nervous System.*

EMG *Electromyogram.*

ENG *Electroneurogram.*

FES *Functional Electrical Stimulation.*

FIR *Finite Impulse Response.*

MEC *Multi-Electrode Cuff.*

PNS *Peripheral Nervous System.*

SNR *Signal-to-Noise Ratio.*

SPU *Signal Processing Unit.*

TDNN *Time Delay Neural Network.*

TMAP *Trans-Membrane Action Potential.*

VSR *Velocity Selective Recording.*

1 Introduction

The human body's nervous system is divided in the *Central Nervous System* (CNS) and the *Peripheral Nervous System* (PNS). The CNS consists of the brain and the spinal cord, which are responsible for controlling the rest of the body. The PNS connects the CNS to the limbs and organs via nerve fibres. The signals from the CNS to the PNS are called *efferent* and the opposite are called *afferent* signals. Various medical conditions, such as spinal cord injuries [1] or strokes, can damage the CNS, while leaving the PNS intact. This causes patients to lose some ability to control bodily functions. However, an opportunity is presented as in most cases the PNS is left in functional condition.

Since 1957, neuroprosthetic devices have managed to re-instill the ability to use the PNS, and thus recover some motor and sensory bodily functions. This has been achieved via the electrical stimulation of particular nerves in the body. This technique is called *Functional Electrical Stimulation* (FES) and has applications in treating neurogenic incontinence, which is a life threatening disease [2]. However, long term FES device installation can cause decrease of excitability or even cell death, as the nerves get less sensitive to the stimulant signals [9][15].

If there was a method to monitor the signals travelling through the nerves, then FES could be applied only when it is required. The *Electroneurogram* (ENG) signal has the potential to provide this information. The ENG provides a measurement of the electrical stimulation of a nerve, which consists of multiple axons that carry the electrical signals (*Action Potential* (AP)) in the body. Using the ENG, one can detect when an unwanted body function is about to happen and stimulate a specific nerve to avoid that. One example of this application is in cases of seizures. The Vagus nerve is stimulated minutes before the seizure happens [5]. Using a feedback control signal this stimulation can be detected and action can be taken in the form of subsequent electrical nerve stimulation or release of chemical substances, avoiding the seizure incident. Hence, the FES technique becomes a closed loop control system with the aid of a feedback signal provided by the ENG.

As mentioned above the ENG provides the AP response from a nerve, i.e. a group of axons. Each nerve can contain from hundreds to thousands of axons. Because each axons is responsible for different body functions, it is essential to identify which of the axons is responsible for the stimulation. Traditionally, spike sorting and pulse shape clustering techniques have been used [7][8]. However, these suffer from DC distortion and cannot be used in real-time spike analysis, as they are time consuming due to their complexity. This has lead to a method called *Velocity Selective Recording* (VSR), which transforms compound APs to the velocity domain and thus, detecting the velocity of neural pulses. Taking advantage of the linear relationship between the axon diameter and the velocity of propagation of the AP, the stimulated axon can be identified. By detecting the velocity of an AP, its diameter can be evaluated. Mapping the detected axon diameter to known human anatomy allows for the identification of the specific axon and thus determine its function.

[1] About 1,000 cases of spinal cord injury annually in the UK, leading to an annual cost of £1 billion for the NHS. [2]

This project is an extension of research done by professor John Taylor et al [1], which presents an improved method of performing VSR using a *Time-Delay Neural Network* (TDNN), which is a type of *Artificial Neural Network* (ANN). This new model is worth investigating as it promises to solve real world issues of FES, due to its excellent performance in VSR, and its potential for real-time implementation. The aim of this project is to attempt to understand why the ANN is performing better than traditional methods, and more importantly, what is it trying to achieve during its optimisation process. The answer to these questions will allow for direct design of a signal processing model which eliminates the training process and potentially increases the velocity selectivity of the model in low SNR situations.

2 Background

This is an adaptation and extension of the Literature Review [12], which was submitted by the same author on the 24th of February 2017.

2.1 Human Nervous System

As mentioned in the Introduction the human nervous system is divided in the CNS and PNS, as shown in Figure 1.

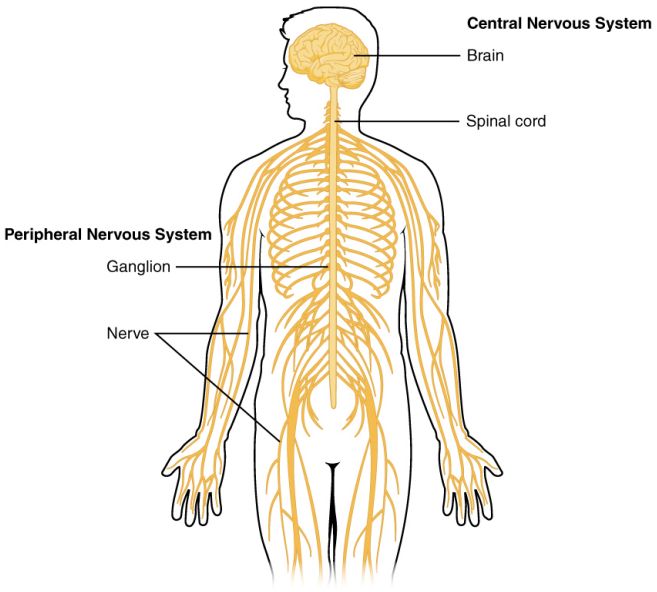


Figure 1: CNS in yellow and PNS in orange. Taken from [3].

The CNS contains the brain and spinal cord, while the PNS consists of long nerves that travel to the limbs of the body. These nerves are bundles of axons, which transmit the electrical pulses and connect the PNS with the CNS, across the body. These axons propagate *afferent* (sensory) and *efferent* (motor) signals in the form of an AP, which is essentially an electrochemical wave causing the release of neurotransmitters at the junctions (synapses). A typical AP pulse is shown in Figure 2 below.

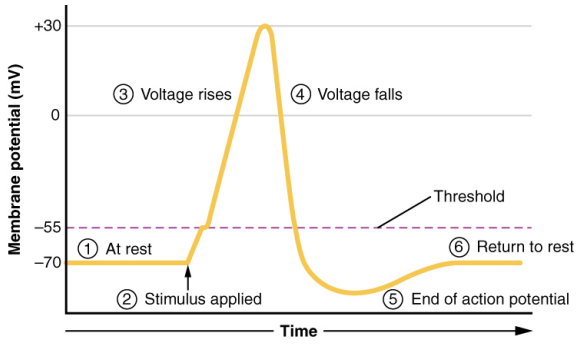


Figure 2: Action potential plot. Taken from [4].

This pulse is caused by the rapid rise and the decrease of the membrane potential of

the biological cells inside the neurons, hence this is also called *Trans-Membrane Action Potential* (TMAP). The membrane potential is the potential difference between the inside and outside of the biological cell in an organism. In this way the body can use these cells to carry signals, thus generating a pulse like the one illustrated in Figure 2.

The velocity of propagation of these APs, is directly proportional to the diameter of each axon and can range from 10 to 120 m/s in humans. This is the only feature in a bundle of axons that can distinguish an individual one, from the rest of the population, which can range from the hundreds to the thousands.

2.2 Multi-Electrode Cuff

The ENG signal is a small voltage signal ($\approx 1 - 10\mu V$) that is produced by electrodes that record the electrical activity of the human nerves. These signals include noise and interference. The interference is caused by the *Electromyogram* (EMG) signal, which is an electrical signal caused by the movement of the muscles, and is much larger in magnitude that the ENG ($\approx 1mV$). The noise is random and can be modelled as *Additive White Gaussian Noise* (AWGN)

The standard method for recording the ENG is by installing an implanted nerve cuff with integrated electrodes. A more sophisticated method is the *Multi-Electrode Cuff* (MEC) [1], which uses multiple (10 - 15) equally spaced (2 - 3mm) electrodes, to measure the propagating AP along the nerve. Typically a tripolar electrode system is employed to remove the EMG interference. A tripolar electrode is shown below in Figure 3.

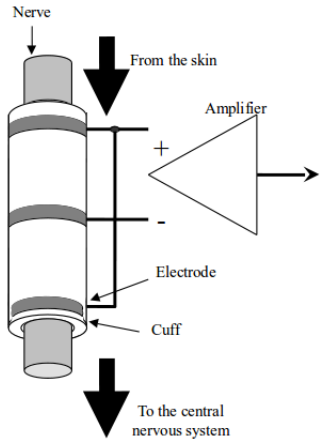


Figure 3: Tripolar Electrode. Taken from [2].

A tripolar AP signal can be generated by the following equation.

$$V_o(t) = V_1(t) - 2V_2(t) + V_3(t) \tag{1}$$

Where $V_1(t)$, $V_2(t)$ and $V_3(t)$ are the AP pulses recorded by the electrodes on the nerve. $V_o(t)$ is the output signal from the configuration shown in Figure 3. Other configurations are also possible. A bipolar signal only uses two electrodes and is given by the following equation.

$$V_o(t) = V_1(t) - V_2(t) \tag{2}$$

Similarly, a unipolar signal only uses one electrode and is the simplest of all configurations.

$$V_o(t) = V_1(t) \tag{3}$$

However, the unipolar AP does not include as much information as the other two. The reason for this is that the tripolar and bipolar configurations, have some velocity information inherently in the signal. Because the signal is captured by more than one electrode, the velocity of the AP affects the shape of the recorded signal. In addition, the unipolar signal suffers from DC distortion. Thus, any noise added to the signal is going to severely affect the result.

2.3 Velocity Selective Recording

As mentioned above, VSR transforms an AP to the velocity domain and manages to detect its velocity. Traditionally, a delay-and-sum method is used. This technique uses a sequence of electrodes described in Section 2.2 to capture the ENG signal as it travels through the nerve fibre, creating a channel of recordings for each electrode. It also uses two rows of differential amplifiers to obtain the tripolar ENG signals, using the method described in Figure 3. This is illustrated below.

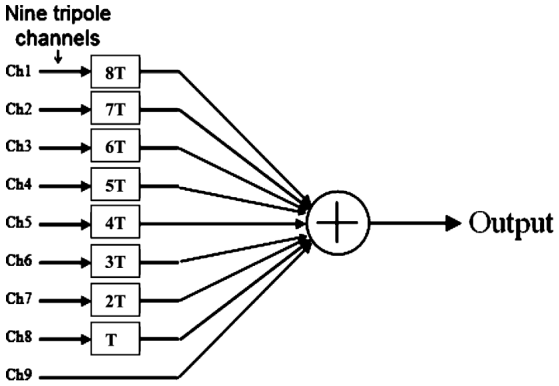


Figure 4: Delay-and-Sum *Signal Processing Unit* (SPU). Taken from [1].

The velocity to be detected is called the *matched velocity*. Each data channel is delayed by an amount T, which corresponds to the time needed for a pulse of the matched velocity v to travel the inter-electrode spacing d, in order to counteract the naturally occurring delays and align all the pulses with the last channel. As a result the amplitude of the pulse of the matched velocity gets maximised after the summation, while all the other amplitudes get suppressed with respect to the matched one. Then amplitude is maximum when:

$$T = d/v \tag{4}$$

The first issue with this approach is that it has low velocity selectivity due to the fact that the cuff length is only about $2-3cm$. However, using a bandpass filter at the output of the system can increase selectivity [10]. In addition, at high AP propagation velocities the inter-channel delays become very small, which puts a limit in the velocity resolution of the system. The overall design of a delay-and-sum method with a MEC in tripolar configuration is shown below in Figure 5.

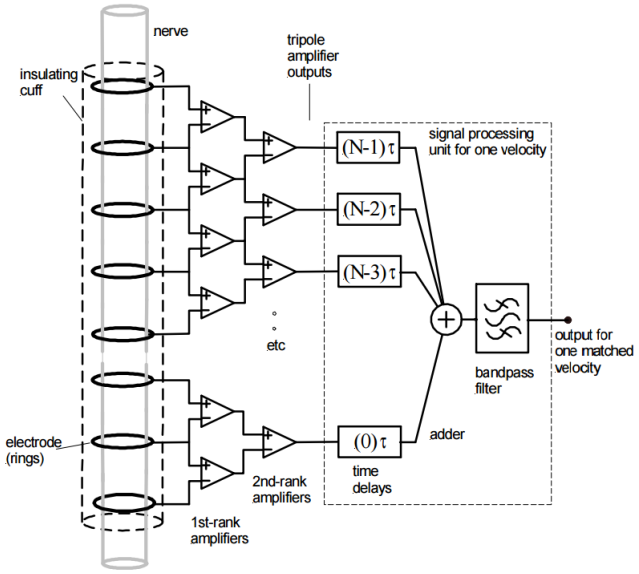


Figure 5: MEC connected to a tripolar array with a bandpass filter at the output. Taken from [11].

2.4 Tuning Curves

Velocity selectivity is a complicated process, hence we need a standard method to assess the performance of a system, in order to compare the operation of different methods. The two common metrics used are the *velocity selectivity*, which is measured via the *quality factor* Q_v [13], and *velocity resolution* R_v [11]. These are defined by the following equations.

$$Q_v = \frac{v_0}{v_{3+} - v_{3-}} \tag{5}$$

$$R_v = \frac{\Delta v}{v} \tag{6}$$

Where v_{3+} and v_{3-} are the velocities at which the normalised response drops to $1/\sqrt{2}$, either side of the matched velocity v_0. Δv is the velocity step and v is the propagation velocity. These metrics give rise to the concept of the *tuning curve*[1], also called an *Intrinsic Velocity Spectrum* (IVS) plot [10]. This is a plot of the maximum of the magnitude of the response of a VSR system across a range of input APs of different velocities. Hence, it contains the velocity spectral behaviour of the SPU, similar to the more traditional frequency spectrum of a system. This plot can give a visual representation of the performance of a system, while also providing information about the shape of the response, which cannot be provided by the velocity selectivity or resolution values. An ideal IVS plot would have a maximum peak at the matched velocity and zero everywhere else. However, in reality we get a smooth decay around the matched velocity.

Having established the standard of the tuning curve and some performance metrics, it is important to analyse the tuning curve of the delay-and-sum method across a range of velocities of 10 to 120 m/s. Here the matched velocities selected are 20, 50 and 90 m/s as they constitute a representative sample of the range.

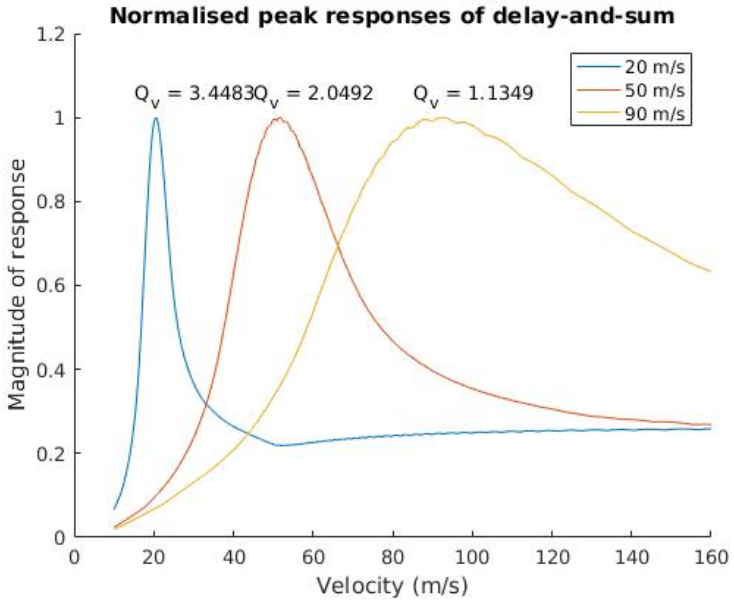


Figure 6: Delay-and-sum method tuning curve for 20, 50 and 90 m/s respectively.

It can be seen from Figure 6 that the delay-and-sum method is able to perform well in low velocities but its performance drops at higher velocities. This is the reason serious research has been performed into discovering new methods of performing VSR.

2.5 Time-Delay Neural Network

In order to approach the two main problems with the standard delay-and-sum approach of VSR, a non-linear model was used[1], called a TDNN. This type of ANN has shown promising potential in VSR as it has shown vastly superior performance than traditional methods. This is a special type of feed-forward ANN, introduced in 1991 [6], which defines a new type of synapse that operates as an *Finite Impulse Response* (FIR) filter in digital filter theory and has been used in speech recognition and time series analysis. The output of the synapse is dependent on both the current and previous inputs and can be modelled by the following equation.

$$\hat{y}(t) = \sum_{j=0}^{M} b_j.x(t-j) \tag{7}$$

where $\hat{y}(t)$ is the output vector at time t, $x(t-j)$ is the delayed input and b_j, for $j = 0, 1, 2, \ldots, M$, are constant weights. A TDNN has one such synapse per channel. After the calculation of the output of each channel, they are all summed together and the bias is applied. The number of weights M, namely the depth of the first layer is defined by the minimum velocity to be examined. For example, if the minimum velocity is 10 m/s and the inter-electrode spacing d is 1mm, using Equation 4, the inter-channel delay is $100\mu s$. For a nine channel system this corresponds to a total delay of $800\mu s$. For a sampling period of $10\mu s$, the depth required is 80 stages. Hence, the size of the first layer is going to be an $m \times n$ matrix, where m is the number of channels of the ANN, and n is the depth of the first layer defined by the minimum velocity to be considered. The goal of the training process is to optimise the values of this matrix. Using this model,

each channel from the MEC can be a time series input to an FIR filter, whose outputs are combined by a summation of the signals, adjusted by a bias value and a hyperbolic tangent transfer function described by the following equation.

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{8}$$

This function is also called the *squashing function* as it suppresses really high valued inputs due to its sigmoid shape shown in Figure 7.

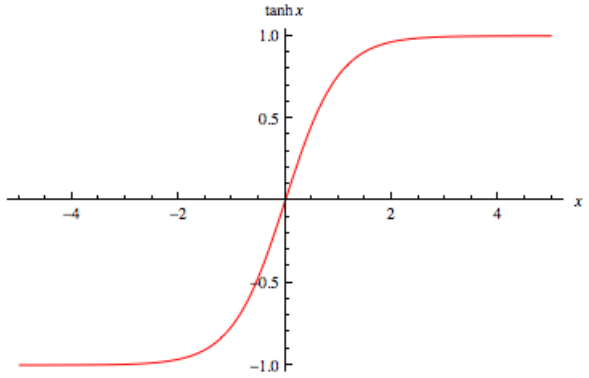


Figure 7: Hyperbolic Tangent Function. Taken from [18].

Hence, the equation of the first layer becomes:

$$z = tanh((\sum_{i=0}^{N} \hat{y}_i(t)) + b) \tag{9}$$

Where z is the output of the first layer, $i = 0, 1, \ldots, N$, where N is the number of FIR channels, b is the bias and $\hat{y}_i(t)$ is the output the i^{th} FIR filter described by Equation 7.

The model proposed in previous research on VSR methods [1] uses a second layer of one neuron with a linear transfer function. The signal processing model using this ANN is shown in Figure 8 and Figure 9 below.

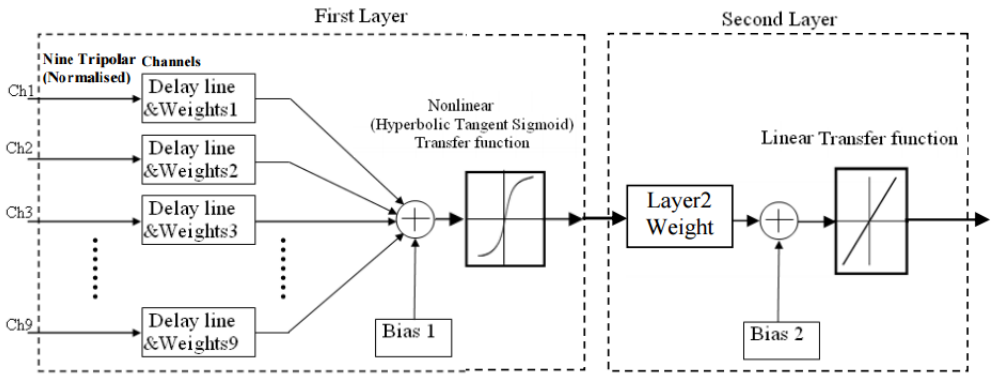


Figure 8: TDNN method of performing VSR. Taken from [1].

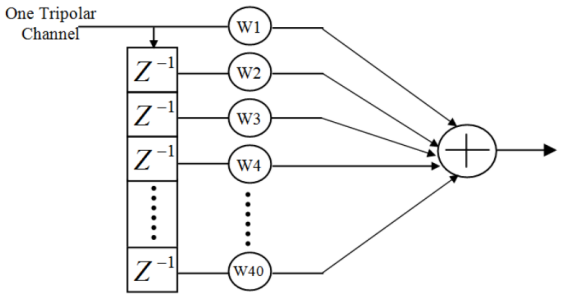


Figure 9: Delay line with adaptive weights. Taken from [1]

The first layer of this neural network architecture is similar to an array of FIR filters. Hence, it has all the key elements that characterise such a filter, such as linear phase response and unconditional stability, due to their finite length and non-recursion.

This new model has shown to maintain high velocity selectivity and resolution at high propagation velocities, which are two weak points for the traditional delay-and-sum method. The tuning curve for a TDNN can be seen below.

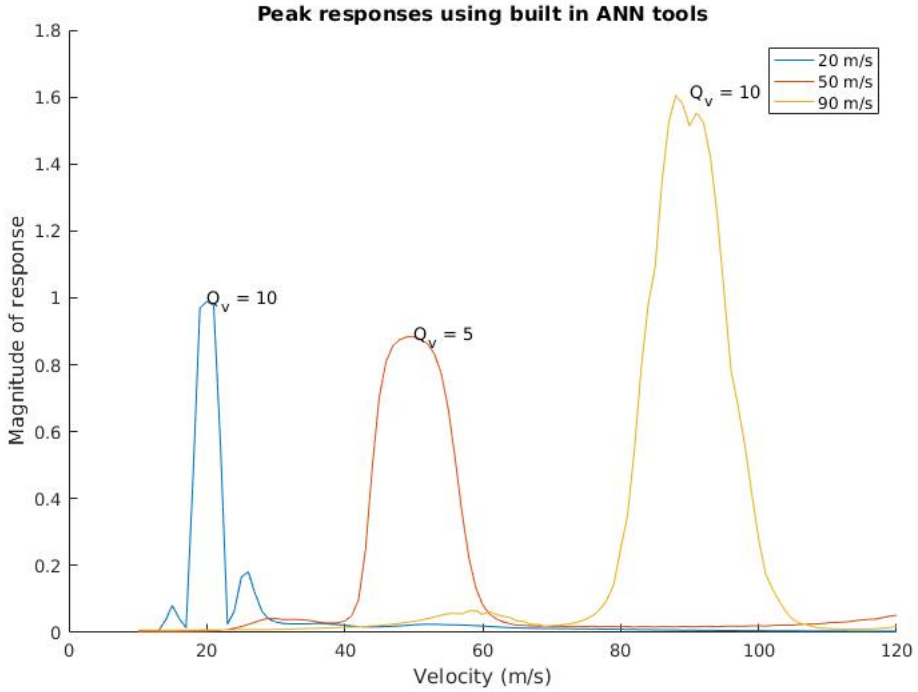


Figure 10: Tuning curve for TDNN with 9 channel tripolar configuration and 120 stages.

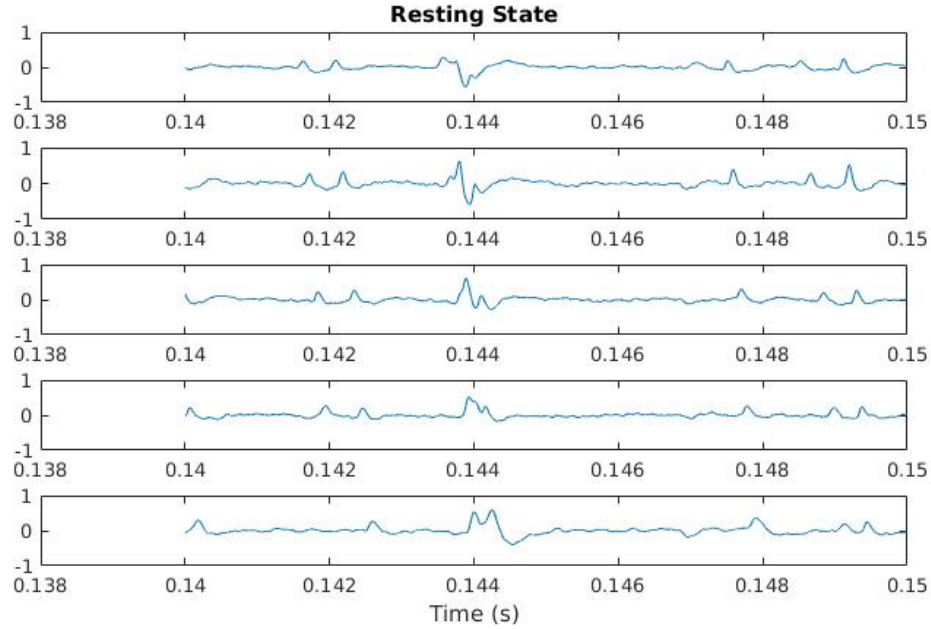
If this tuning curve is compared with the one illustrated in Figure 6, it can be seen that this SPU model has much better performance that is constant across the whole velocity spectrum, which is a very desired characteristic in VSR. In addition, this model can be run in real-time, an advantage over spike sorting and AP shape clustering techniques [7], while maintaining robustness to noise. The challenges faced with this method are that one neural net can only classify one velocity, and that the optimisation achieved by the training process is not fully understood yet. This project aims to answer these

questions with the hope that a clearer understanding of the model will also improve its performance in low SNR situations.

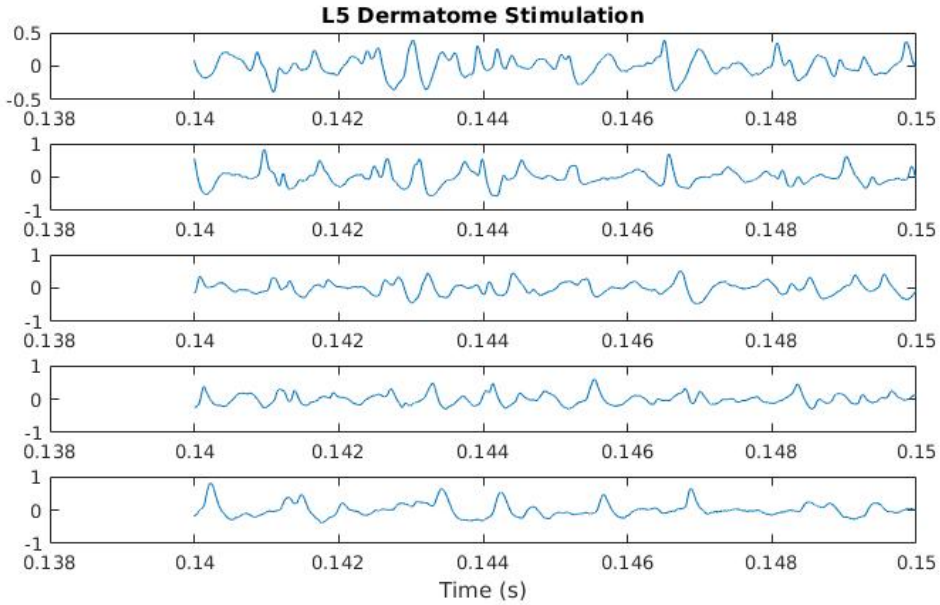
2.6 Physiological ENG in Rat

One of the issues with this TDNN is that it was not able to perform VSR with real data that was extracted from a rat [16]. This section is going to overview the background of the generation and setup used to extract this data, as well as have a sneak peak at the actual shape of the recordings.

The AP velocities found in rats range from 5 to 30 m/s. The surgical procedure followed to extract the signals, exposed the spinal cord of the rat, which allowed the dissection of one of the fascicles on which the MEC was fitted on. This fascicle was $100\mu m$ in diameter and the MEC was arranged in an isolated bipolar configuration. This means that from 10 electrodes 5 bipolar channels were generated. Hence the ENG signal generated is described by Equation 2. The total length of the MEC was $5mm$ and the inter-channel spacing was $1mm$, which yields an inter-electrode spacing of $0.5mm$. Each of the bipolar channels was sampled for $250\mu s$ for $2.5s$ at a sampling rate of $f_s = 500kHz$. With this setup, two sets of data were recorded. One dataset included ENG signals of the rat in resting state. The other dataset included recordings from naturally evoked stimulation from cutaneous skin sensation. This was achieved by lightly stroking the L5 dermatome. This is an area of the skin that is controlled by a single nerve fibre.



(a) Resting state ENG recordings.



(b) Stimulated ENG recordings.

Figure 11: Real rat ENG recordings.

It can be seen from the rat recordings that in the stimulated case there are a lot more APs travelling though the nerve, which leads to AP overlap and more events of constructing or destructive interference. However, in the resting state recordings there are more distinguished APs with less interference, which makes the signal processing much easier and shows promising potential.

2.7 Centroid Filter

In mathematics the centroid is the geometric centre of a shape. This is the arithmetic mean of the positions of all the points in the shape. Using this property it has been proposed that a centroid filter can provide a more robust and effective method of aligning APs [17]. By considering an AP as a two dimensional polygon, then the centroid of the AP will be the centre in both time and amplitude. Because the centroid is a function of all the samples in the AP, it has the potential to be more resilient to DC distortions.

By considering the AP shape as a polygon, the time axis can be labelled with x and the magnitude axis with y. Then the centroid along each axis can be labelled as C_x and C_y, respectively. Taking the first moment of area along the x axis and dividing by the total shape area, gives the centroid along the x axis.

$$C_x = \frac{1}{A}\int_{t_1}^{t_2} xf(x)dx \tag{10}$$

Where A is the total area of the polygon, $t_1 - t_2$ is the width and $f(x)$ is the function that describes the AP, i.e. Equation 16. Now we consider the convolution of $f(x)$ with another function $h(x)$:

$$y(x) = (h * f)(x) = \int_{-\infty}^{\infty} h(x-\lambda)f(\lambda)d\lambda \tag{11}$$

If this is performed at the origin, where $x = 0$, then Equation 11 reduces to:

$$y(0) = \int_{-\infty}^{\infty} h(\lambda)f(\lambda)d\lambda \tag{12}$$

Now this can be realised with an FIR filter if the impulse response is made to be:

$$h(\lambda) = -k\lambda \tag{13}$$

Considering the general form of the discrete FIR filter output $y[n]$, for a length L with input $x[n]$ and coefficients b_i:

$$y[n] = b_0.x[n] + b_1.x[n-1] + \cdots + b_L.x[n-L] = \sum_{j=0}^{L} b_j.x(t-j) \tag{14}$$

In order to implement Equation 12 as a discrete FIR filter the filter coefficients must be set according to:

$$b_i = \frac{-2i}{L} + 1 = mi + 1 \tag{15}$$

The effect that an FIR implementation of the centroid filter has on a pulse, is illustrated below.

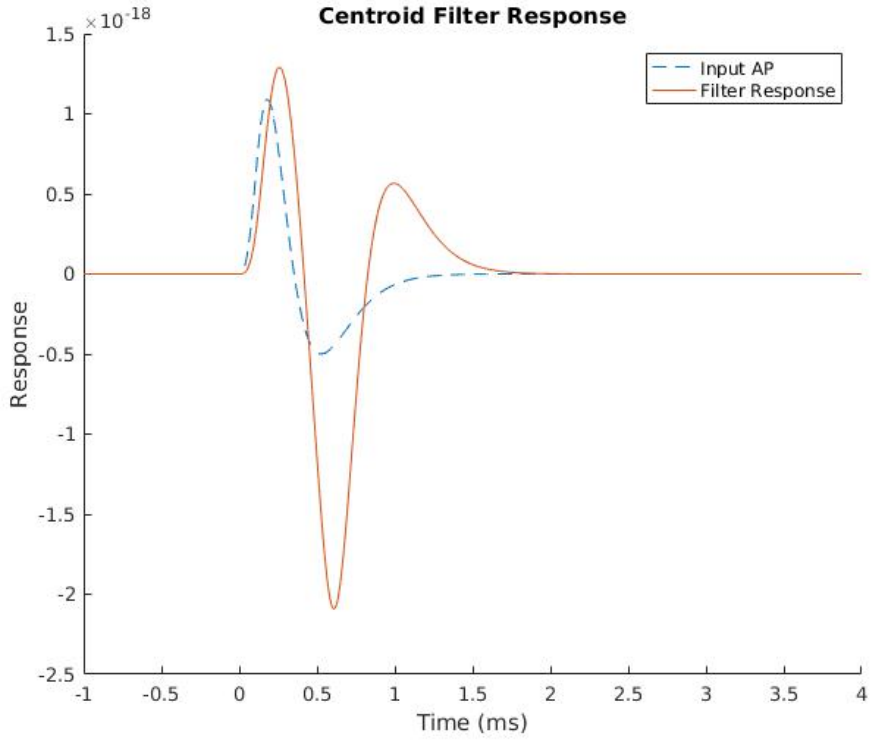


Figure 12: Effect of a centroid filter on an input AP pulse.

Hence, using this theory an FIR filter can be used that can more efficiently perform AP alignment in the presence of noise. It is already easy to see how this can improve upon the traditional delay-and-sum method.

3 Aims and Objectives

The main aim of this project was to provide an deeper understanding of the way with which the ANN has managed to increase VSR performance. Up until now, the output of the ANN was observed and compared with the traditional delay-and-sum method. This project disassembled the ANN into pieces and analysed the function of each of its blocks. In addition, this model has failed to perform when it was given real data extracted from mammals. This was another point of interest of the project in understanding why it was failing.

- Understand literature and general theory behind ENG signal generation, velocity selective recording and time delay neural networks.
- Understand MATLAB code generated for design and testing of the ANN, in order to edit it and break down ANN function.
- Understand why the system fails to do VSR with real data.
- Understand overall function and optimisation of ANN and why it improves upon delay-and-sum.
- Design FIR filters that can perform VSR like the ANN does to validate its operation.

4 Methods

This section is going to describe the methods that were used to reach the aims and objectives of the project, such as the ENG signal simulation, training process of the ANN and the methods used to investigate the performance of the model. This builds upon previous research [1] and existing MATLAB models.

4.1 Action Potential Simulation

In order to train the ANN some generation of simulated data is needed. The first step for this is to model the AP pulse as accurately as possible. The input to the MEC is, in fact, a TMAP pulse, which can be modelled by a function $V_m(t)$ [14] described by the following equation:

$$V_m(t) = \begin{cases} v^2 A t^n e^{-Bt} & \text{for } t \geq 0 \\ 0 & \text{for } t < 0 \end{cases} \tag{16}$$

Where A, B and n are constants and v is the AP velocity. Hence, the higher conduction velocities have larger signal amplitudes due to this v^2 term. The values of these constants determine the shape of the AP in order to approximate the ones found in mammals and to control the duration to $1ms$. In this project the TMAP function used was the one that was established in [10]. The parameters used in Equation 16 are $A = 4.08 \times 10^{-3}$, $B = 1.5 \times 10^4$ and $n = 1$. This was easily implemented in MATLAB, with a sampling frequency of $f_s = 100kHz$, an electrode spacing of $1mm$, which resulted in the following TMAP approximation for a tripolar MEC configuration.

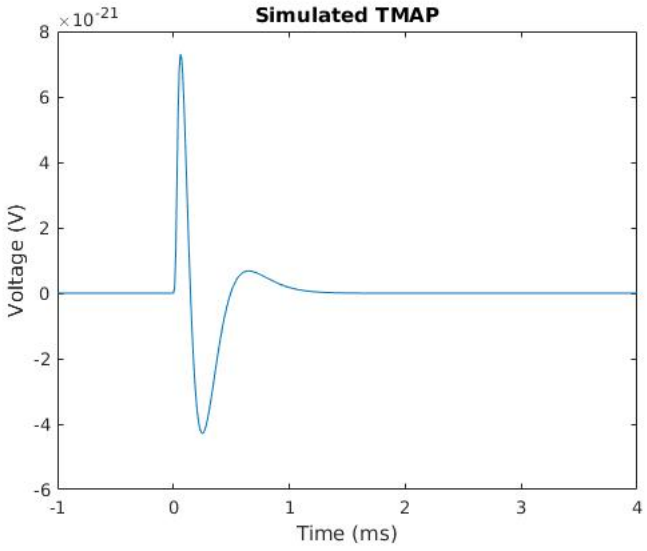


Figure 13: Tripolar TMAP simulated in MATLAB for a velocity of 50 m/s.

Figure 13 is a simulation of a single tripolar channel. The MEC usually has multiple channels. This has been simulated in MATLAB, to provide a multi-channel signal that approximates the operation of the actual MEC. This has been achieved by time shifting the pulses in order to simulate the time the AP takes to travel the inter-channel distance.

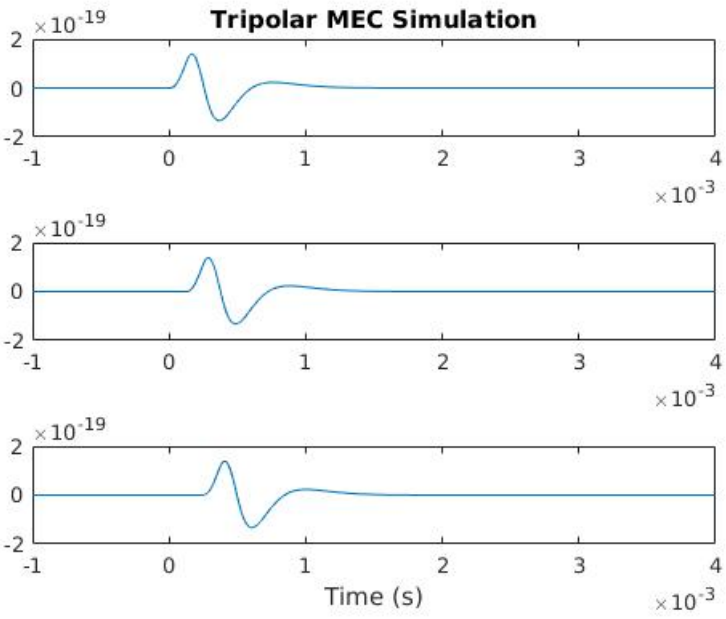


Figure 14: Three channel tripolar MEC signal of 50 m/s velocity.

Each of the above plots is the signal captured by each channel of the MEC. It can be seen that the AP arrives at each channel at different points in time as it travels the length of the device.

As mentioned before the signal captured by the MEC will suffer some additive noise. In order to improve our simulation of the actual AP captured and consequently improve our training process, a parameter has been added to the model to simulate AWGN noise, as mentioned in Section 2.2. A tripolar AP with additive Gaussian noise with an amplitude of $2e^{-21}$ is shown below.

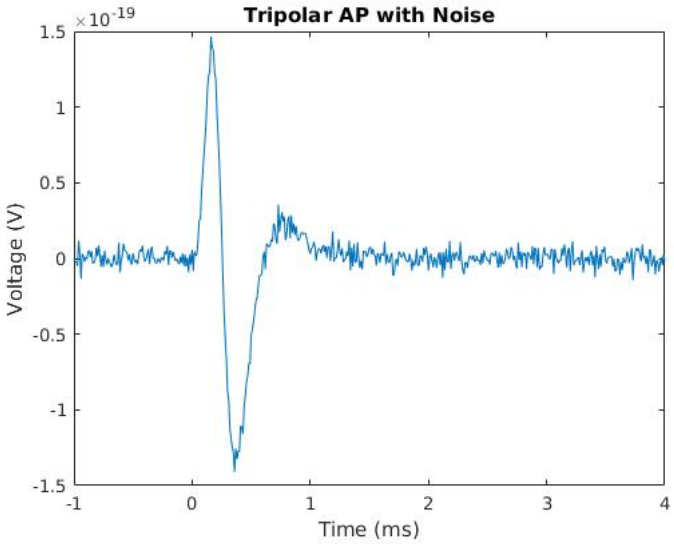


Figure 15: Tripolar TMAP with noise simulation.

Other parameters that can be tunes are the inter-electrode distance d, the sampling frequency fs, the number of electrodes and the MEC configuration (unipolar, bipolar or tripolar).

4.2 Artificial Neural Network Design

In order to design a neural network like the TDNN described in Section 2.5, MATLAB's Neural Network Toolbox was used. This can be done using the `distdelaynet` function, which generally creates a distributed feed-forward neural network, but can be modified to design a TDNN. The minimum velocity this project is considering is 20 m/s, the signals are sampled at 100kHz and the inter-electrode distance is 3mm. Repeating the calculation done in Section 2.5 results in a required depth of 135 in order for the SPU to be able to operate at the slowest velocity. However, by increasing the number of weights of the first layer the ANN has more parameters to optimize. This can result in over-training to the simulated data. Hence, the depth of the first layer used was 120 stages. This architecture is shown below.

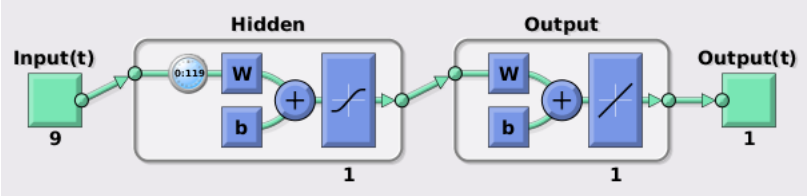


Figure 16: ANN design.

Due to the fact that FIR filters cannot have a different impulse response for different inputs, one ANN is trained for each matched velocity. This TDNN is trained using a supervised algorithm, which requires the generation of the training input data along with the desired outputs. The challenge here is to decide what the ideal output would be for each input. It has been shown [1] that a rectangular output pulse is the most effective. The output pulse amplitude and width are derived from the point where the input AP's amplitude has fallen below 3dB.

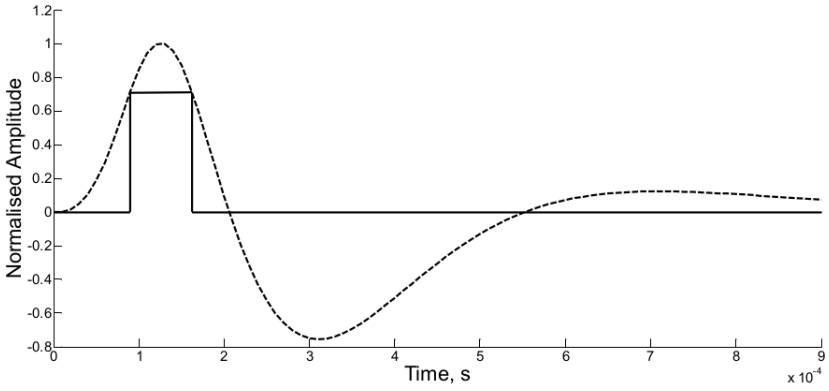


Figure 17: Tripolar AP overlayed with target output pulse. Taken from [1].

In this way the output encapsulated information about the delay, amplitude and duration of the AP. An example of a training sequence (with simulated noise) for a 20 m/s matched

velocity, with a nine channel tripolar configuration is shown below.

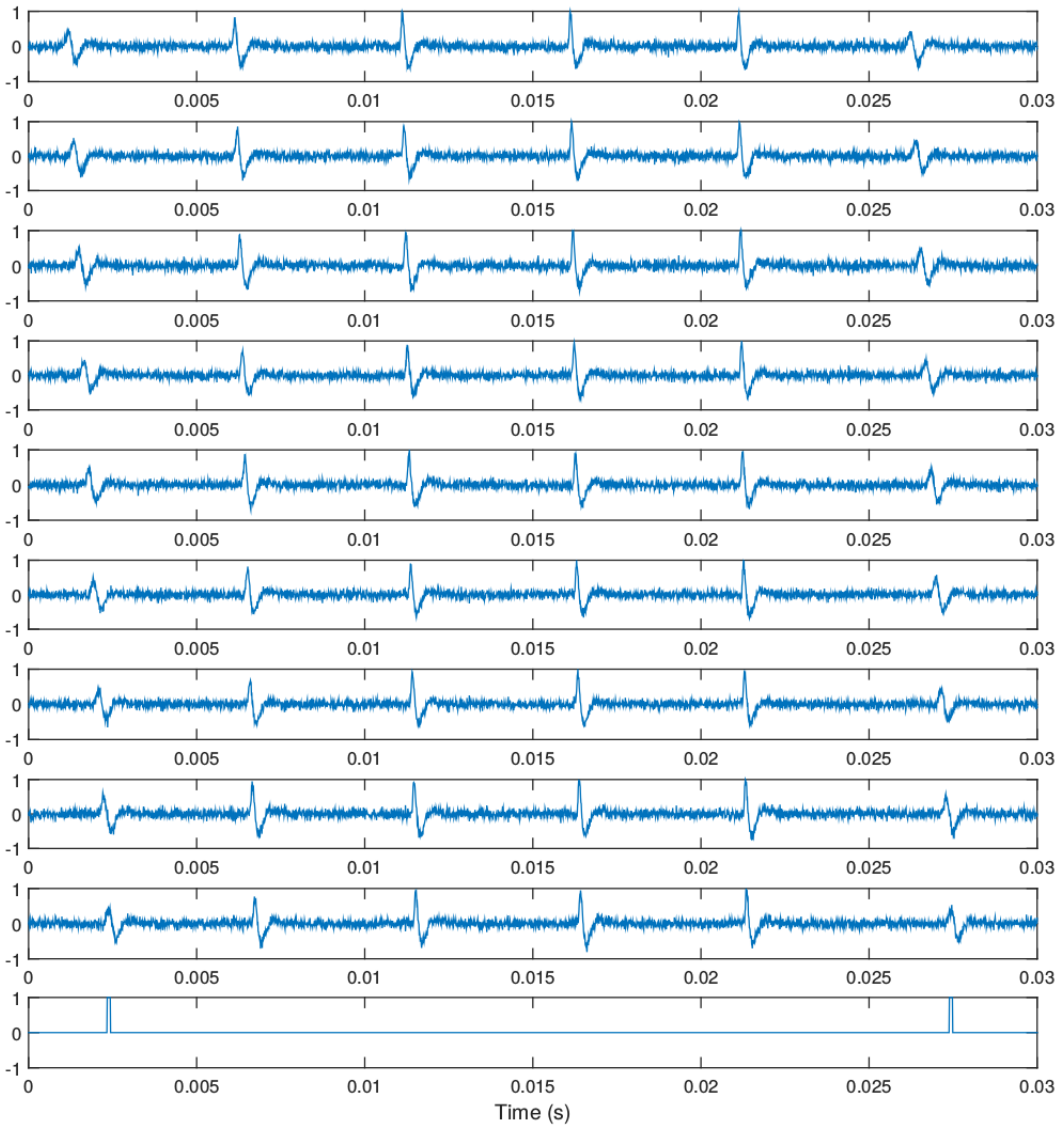


Figure 18: Training sequence of APs with simulated noise for 9 channels. The velocities are 20, 40, 60, 80, 100 and 20 m/s, respectively. The last plot is the target output signal.

MATLAB's default and most effective (for TDNNs) training option is `trainlm` that uses the *Levenberg-Marquardt* backpropagation algorithm, which is used for non-linear least squares problems [19]. It is a combination of the *Gauss-Newton* and *gradient descent* methods. The gradient descent method, is a general minimisation technique, which tries to update the model's parameters in order to follow the "downhill" direction with respect to the goal of the problem. Usually, it makes use of the error metric, which it tries to minimise by following the negative slope using its derivative. The term backpropagation refers to the method used to update the weights of the ANN, from the output back to the first layer by calculating the partial derivatives of each of the parameters with respect to the output.

In the Gauss-Newton algorithm the sum of the square errors is minimised by assum-

ing that the solution is a quadratic function, whose minimum it tries to find. Its is typically much faster than the gradient descent method. The Levenberg-Marquadt operates like a gradient descent algorithm when it is far away from the optimal solution and switched to Gauss-Newton when it is close, in order to accelerate the convergence. Training a TDNN with the Levenberg-Marquardt algorithm is a complicated process and has been evaluated in other research [20], so this report is not going into detail on this technique.

MATLAB's training tool allows for various parameters to define the end of the training process:

- Maximum number of epochs has been reached.
- Performance goal has been met.
- Number of validation steps has been met.
- Gradient has dropped below the minimum value.
- The damping term μ has exceeded its maximum value.

Figure 19: MATLAB's training tool.

5 Results and Analysis

This section is going to present the results of the investigation of this ANN and how it is able to improve upon existing VSR methods, such as delay-and-sum. In previous research the ANN that was investigated was had a nine channel tripolar input. This configuration although effective is too complicated to process and understand. For example, the following figures illustrate the time response and weight patterns of each of the nine channels.

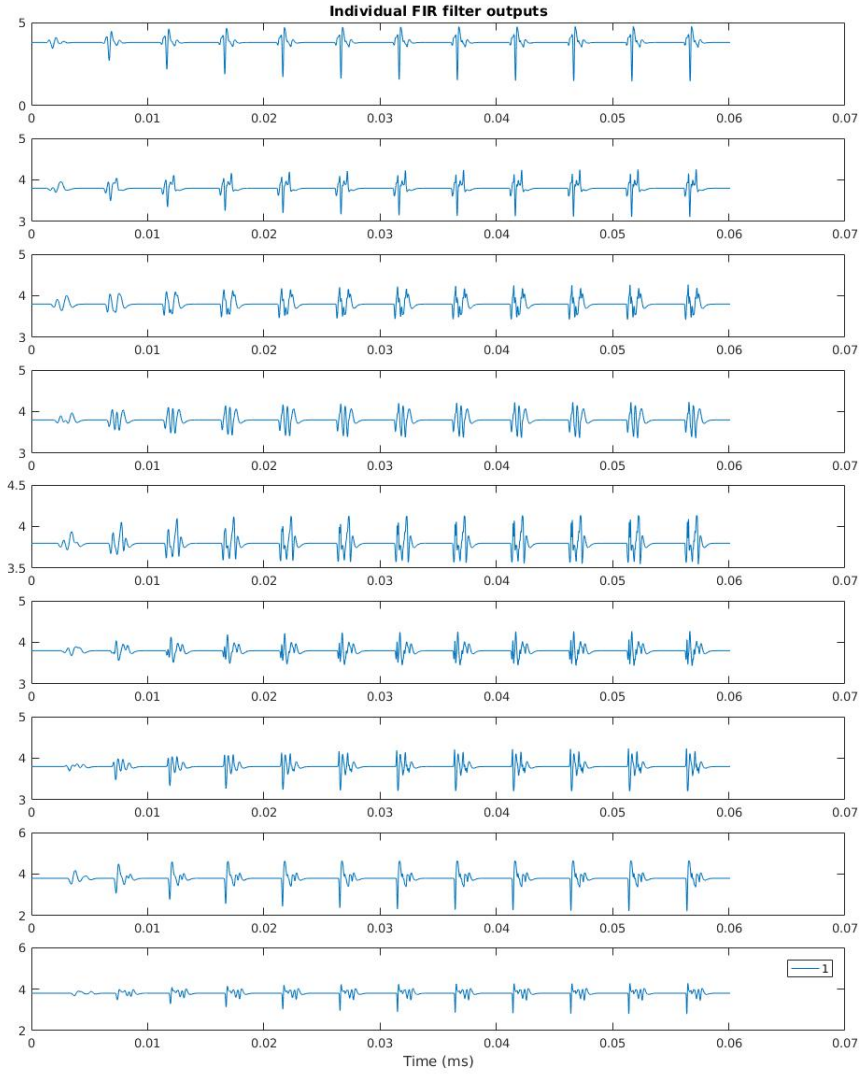


Figure 20: Time response of ANN.

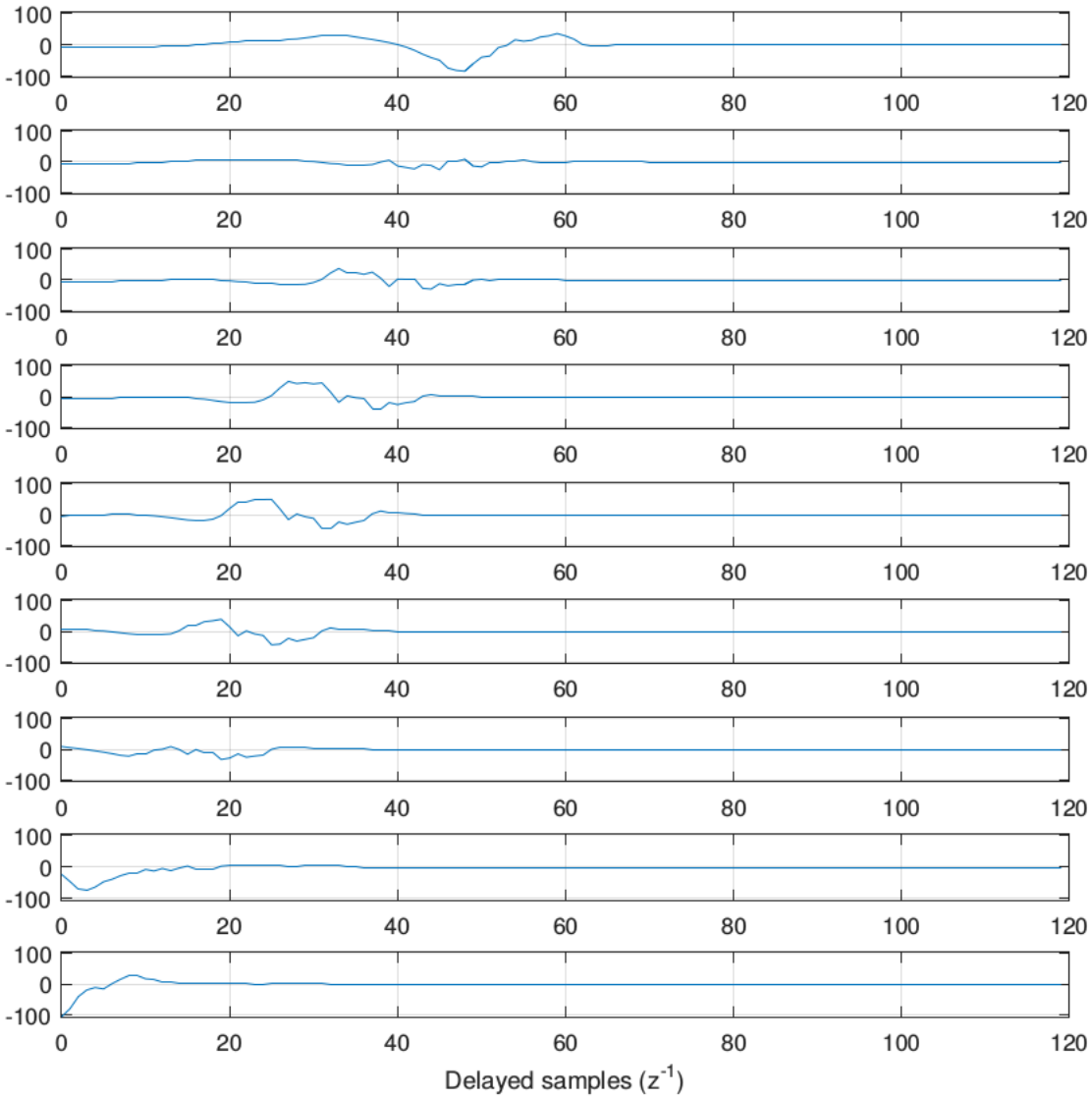


Figure 21: Weight pattern of ANN.

It can easily be seen from the above illustrations that the analysis of such a system is quite complicated, especially in the time domain case. Previous research [21] has performed frequency domain analysis of the weight patterns and frequency and phase analysis of the FIR filters of the TDNN. These analyses did not yield any breakthroughs in understanding the operation of the model, thus this project focused on time response analysis. In order to simplify the problem, the ANN under investigation will have three isolated bipolar channels (described by Equation 2), have a depth of 120 stages, an inter-electrode spacing of $3mm$, with the signal sampled at $100kHz$. The ANN trained with this configuration yielded the following tuning curve.

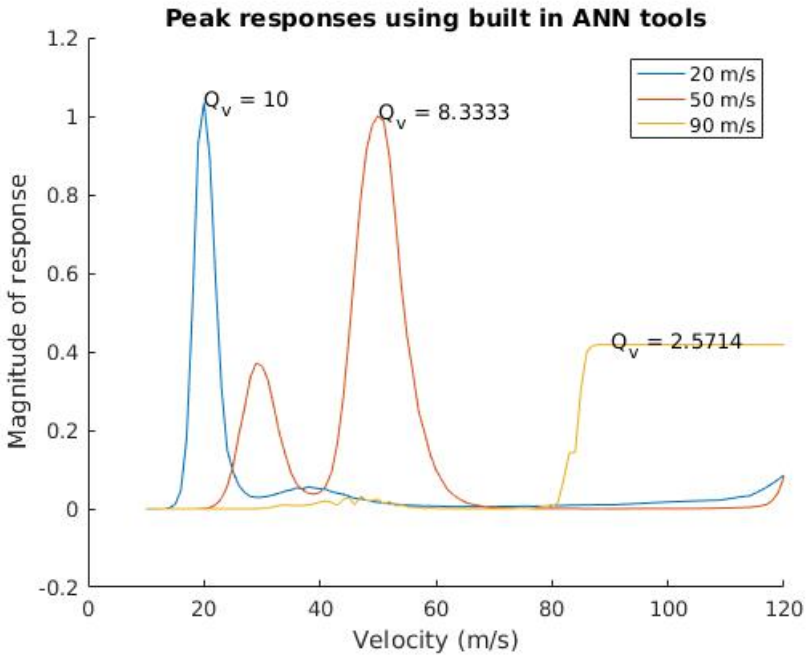
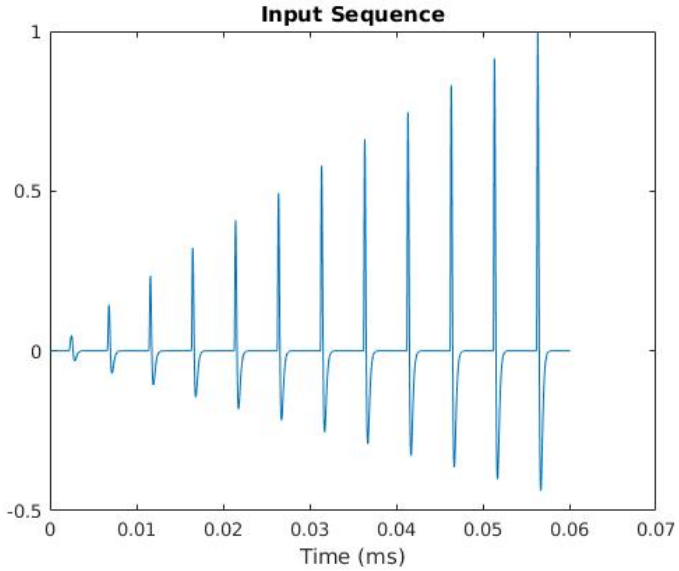


Figure 22: Tuning curve for ANN under analysis.

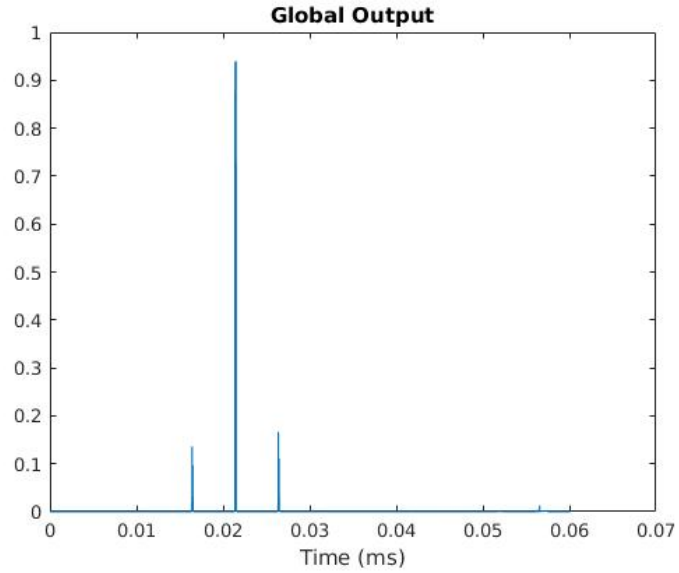
This simplified model can perform VSR efficiently in low velocities, as shown by the Q_v values for the 20 and 50 m/s curves, but struggles at 90 m/s. This is because higher velocities are more difficult to distinguish with less channels as the spread of the pulse, in time, across the device is more narrow.

5.1 Complete Model

In order to understand the operation of the TDNN, the most obvious way is to examine the time response of this SPU. An ANN is like a black box that makes a specific mapping of inputs to outputs based on the training dataset. Hence, it seems wise to look at the shape of the output, when specific inputs are applied. The ANN output when the input is a sequence of APs that range from 10 to 120 m/s in steps of 10 is illustrated below.



(a) Input AP sequence. AP velocities range from 10 to 120 m/s in steps of 10.



(b) Output of the ANN for a matched velocity of 50 m/s.

Figure 23: Complete ANN model time analysis.

It can be seen from the graph that the AP at the matched velocity (the fifth pulse from the left), is nearly the single peak at the output with only its neighbouring APs having a comparable, but significantly smaller, magnitude responses. All the other pulses are almost perfectly suppressed. This illustrates the excellent performance of the ANN model. The same result occurs for other matched velocities. These figures have been included in the Appendix if further investigation is needed.

5.2 Half Model

The result of the model's performance seems impressive and it is necessary to analyse its internal structure in order to understand its function. A logical place to start the

investigation is after the first layer of the neural net. This includes the FIR filters, the summation, the bias, and the sigmoid function described by Equation 8. The MATLAB toolbox provides a useful *Application Programming Interface* (API) that allows full control over the ANN model. Hence, it is possible to connect the output of the first layer to the final output of the model, bypassing the output layer.

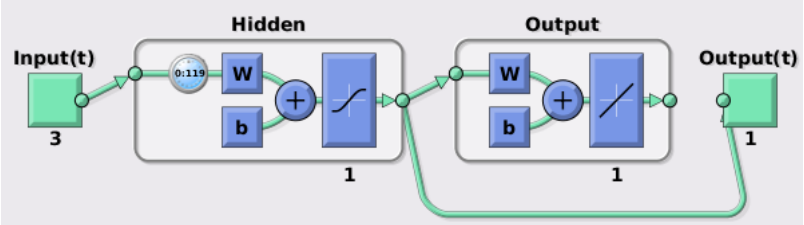


Figure 24: Half model architecture.

It is interesting to see that even with half of the model connected to the output and only three channels, it is able to perform VSR in an effective manner.

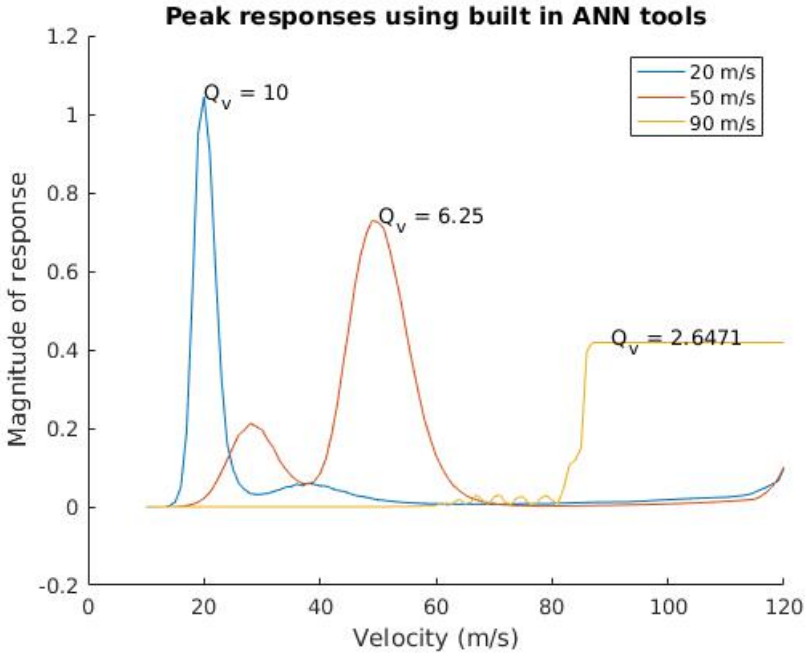
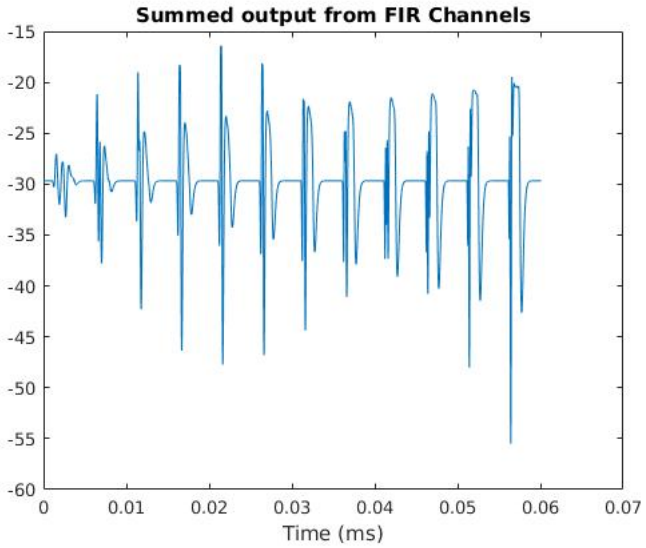


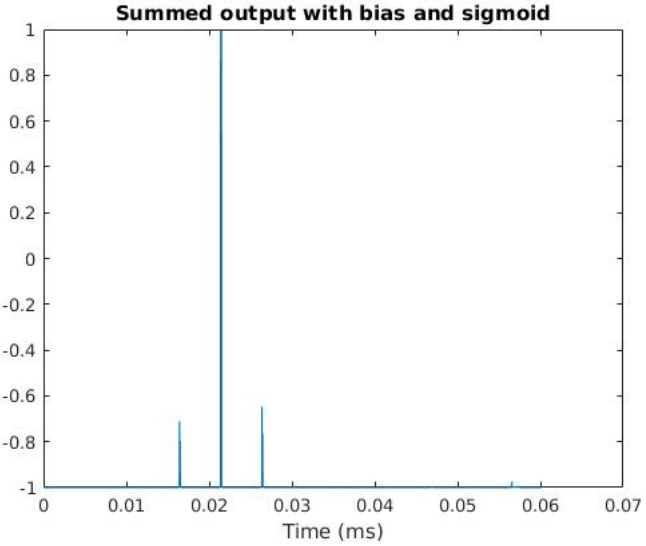
Figure 25: Tuning curve for half model.

The tuning curve in Figure 6 shows that it performs worse than the complete model. So that means that the output layer is in fact, functional.

For the same input shown in Figure 23a, the output of the half model is shown below in Figure 26b.



(a) Time response of half model before sigmoid function for matched velocity of 50 m/s.



(b) Half model output for 50 m/s matched velocity.

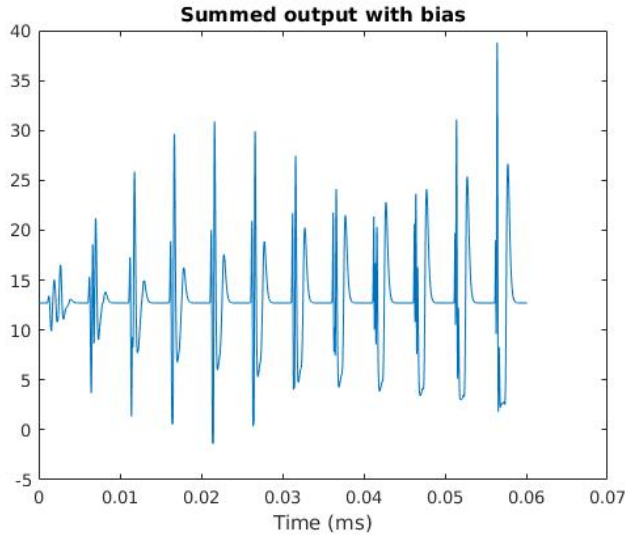
Figure 26: Half model impulse response.

From Figure 26b above, it can be seen that the output of the half model resembles the global output. The only difference is that the half model output is shifted by approximately 1 unit downwards relatively to the global output. However, the general AP resolution remains the same. Hence, it can already be suggested that the intelligence of this model is hidden in the first layer.

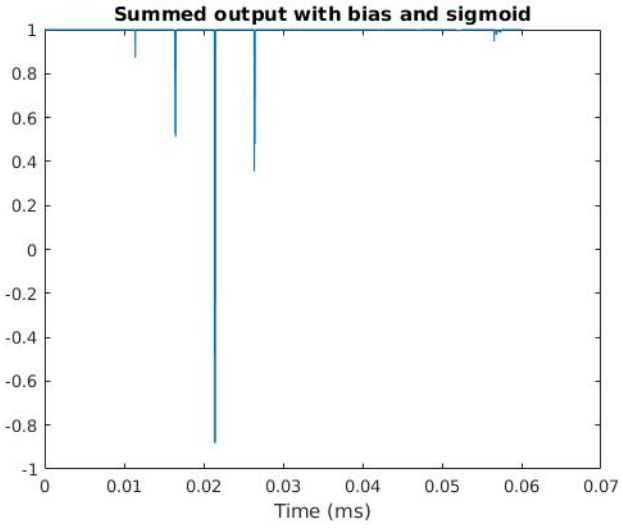
The next natural step would be to examine the effect of the hyperbolic tangent function. This requires to look at the time response before the function, which can be done by replacing the activation function with a linear one. By looking at the time response of the filter before and after the sigmoid function, some interesting conclusions can be drawn. From Figure 26 it can be seen that the model uses the bias value to optimally adjust the DC level of the response in a position, which allows the sigmoid to suppress

the non-matched APs and expand the matched one. This is why this sigmoid is also called the *squashing function*.

A previously unimagined ability of the half model was also discovered. It has been observed that in some cases the FIR filters fail to perform VSR in the positive phase of the APs. Namely, the positive peak of the matched velocity AP, did not exceed the rest. In fact, if one looks at the time response in Figure 27a, it is obvious that the whole time response is inverted from the one in Figure 26a. In this case the model was able to perform VSR by using the negative peaks. In such cases the negative peak of the matched velocity pulse is much smaller than the rest. Hence, by adjusting the bias value it is able to shift the response to an optimum value where the sigmoid function is able to suppress all the other peaks and just maintain a large negative peak at the matched velocity.



(a) Time response of half model before sigmoid function.



(b) Half model inverted output.

Figure 27: Half model inverted effect for a matched velocity of 50 m/s.

Since the optimisation process failed to distinguish the matched pulse in the positive direction, it shifted the whole response to an optimal position where the only the negative peak of the matched pulse is expanded. After this, the output layer weight is negative to invert the output and produce a result similar to the one shown in Figure 23b. This is the reason that when the half model was considered and the output layer was bypassed, the tuning curve showed slightly worse results (see Figure 25). The model was not able to do this distinguishing of the APs in the negative direction.

5.3 FIR Stage Analysis

Going one step further, the MATLAB API allows us to just look at the output from our FIR filters. This can be done by replacing the activation function with a linear one, so that any input to it is just copied verbatim to the output, and by removing the bias of the first layer. The setup for this is shown below.

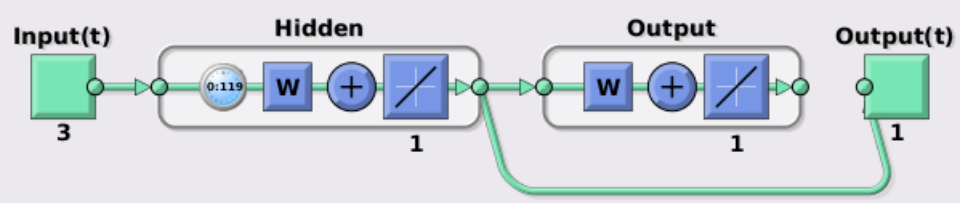


Figure 28: FIR filter architecture.

Running the optimisation algorithm for the architecture illustrated in Figure 28 above, yields the following tuning curve.

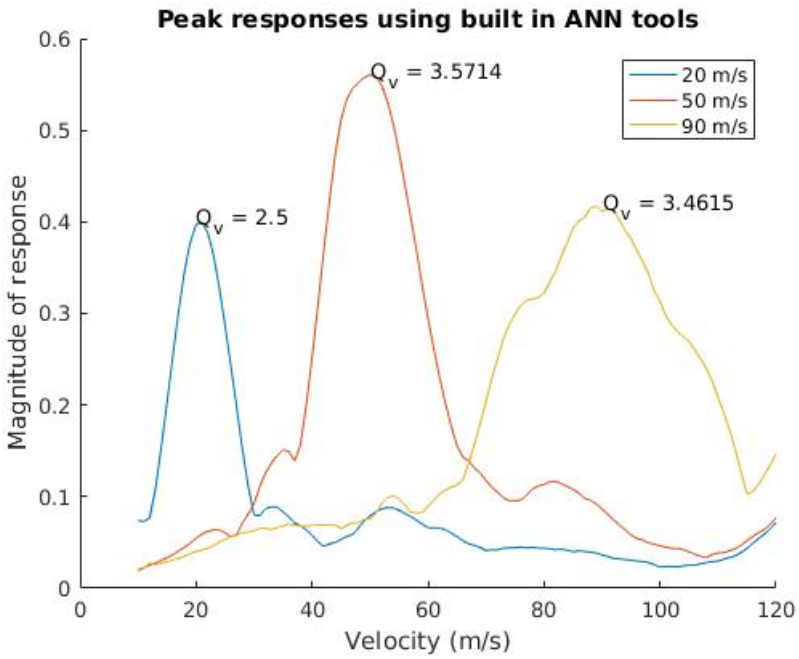


Figure 29: FIR filter tuning curve.

By comparing the tuning curve shown in Figure 29 with the corresponding one for the delay-and-sum method, shown in Figure 6, one can deduce that just the FIR filters of

the TDNN, are capable of better VSR performance than normal delay-and-sum. This means that instead of just aligning the pulses they also do some clever processing to increase the selectivity as indicated by the Q_v values. In fact, the FIR stage maintains a constant $Q_v \approx 3$ across the velocity range. A very desirable feature in VSR.

In order to extract the time response of the model from the FIR filters it was necessary to create a multi-channel input (see Figure 14), as normal, and then successively set all channels, but one, to zero and record the output. What this achieves is to get the response of each FIR filter separately. After all the time responses are gathered, they can all be added together to obtain the full time response. The MATLAB code for this can be found in the Appendix of this report. The examined velocity here is for 50 m/s. The rest are included in the Appendix.

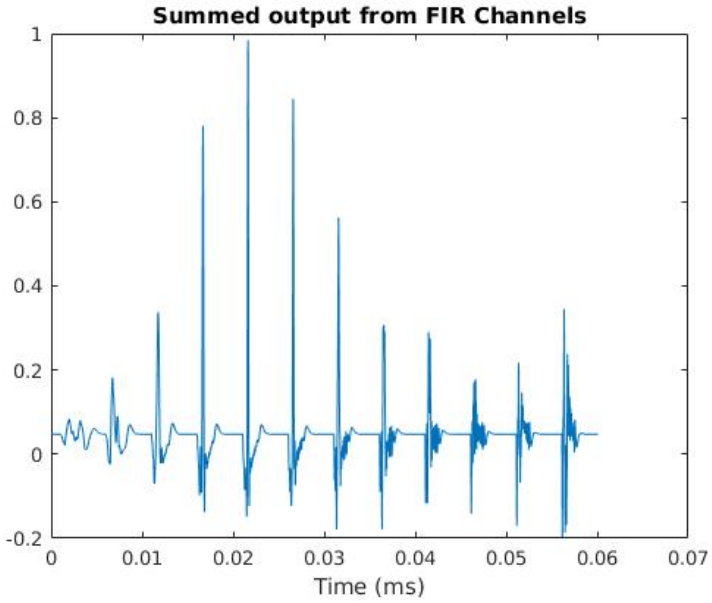


Figure 30: Time response of FIR stage for a matched velocity of 50 m/s.

The FIR filters are capable enough to achieve a high selectivity as indicated by the $Q_v \approx 3.6$. To understand how these filters are able to extend the performance of the delay-and-sum method and have the time response shown in Figure 30, the individual filter time responses must be analysed. These can be obtained by the same method as described above, with the source code included in the Appendix.

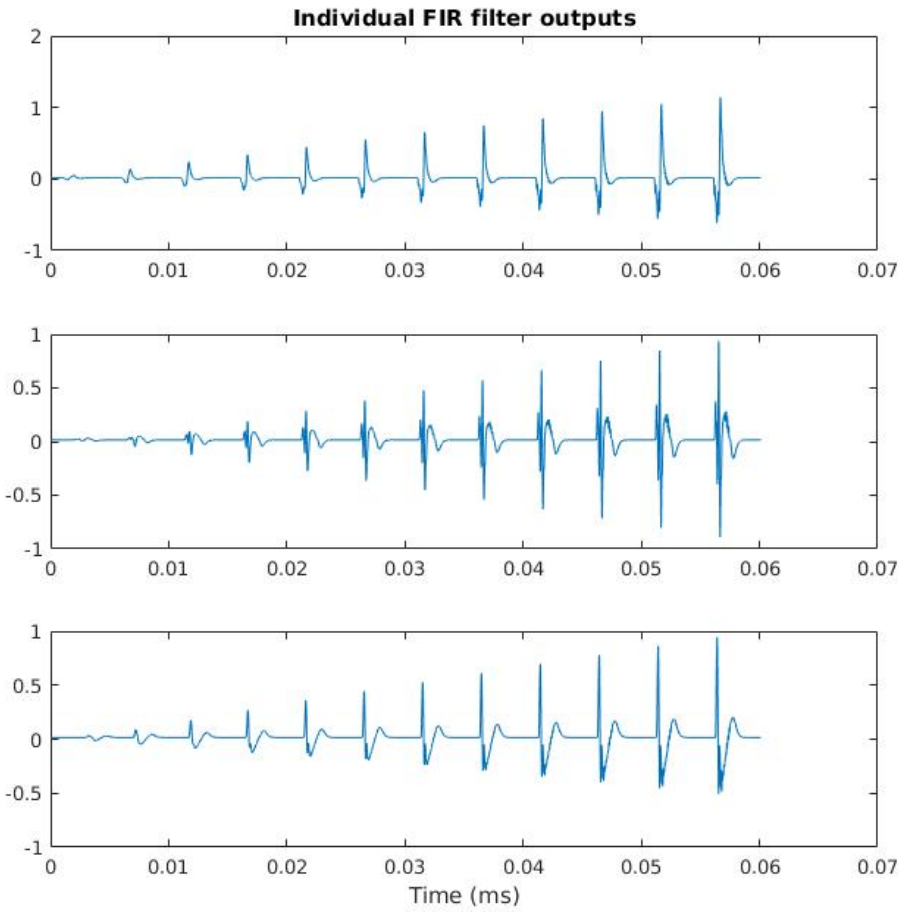


Figure 31: Individual FIR channels time responses for a matched velocity of 50 m/s.

It can be seen from Figure 31 above, that with a three channel configuration, all the pulses are aligned, which hints some delay-and-sum operation, and the middle channel has its peak in the negative direction, while channels 1 and 3 are in phase. This result was obtained for repeated optimisation processes and for all matched velocities. These are included in the Appendix. Namely, channel 2 always had its peak in the negative direction and the other two in the positive. The repetition of this pattern confirms that there must be a specific advantage that this configuration offers over just aligning the pulses and summing them together. An important characteristic to be looked at is the fact the although the input pulses are bipolar, as shown in Figure 23a, the pulses after the FIR filters have changed their shape and have more peaks, especially in the case of channel 2. This will be discussed in detail later on in the report. Diving deeper into this phenomenon the overlaid time responses of the channels must be looked at individually.

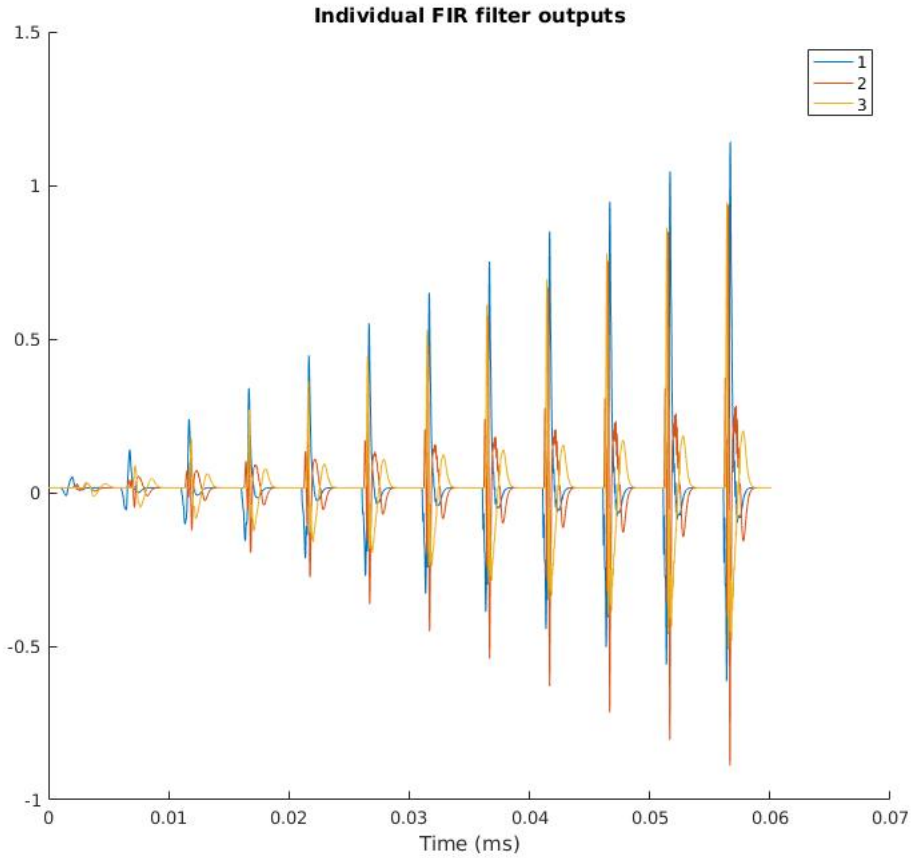


Figure 32: Overlaid FIR time responses for a 50 m/s matched velocity.

It seems that the channels 1 and 3 have similar phase and their peaks interfere constructively, and the pulses from channel 2 are opposite phase and destructively interfere, creating the time response shown in Figure 30. However, the key of the model's performance lies in the timings of these pulses. Next, the report is going to examine the interaction of these pulses.

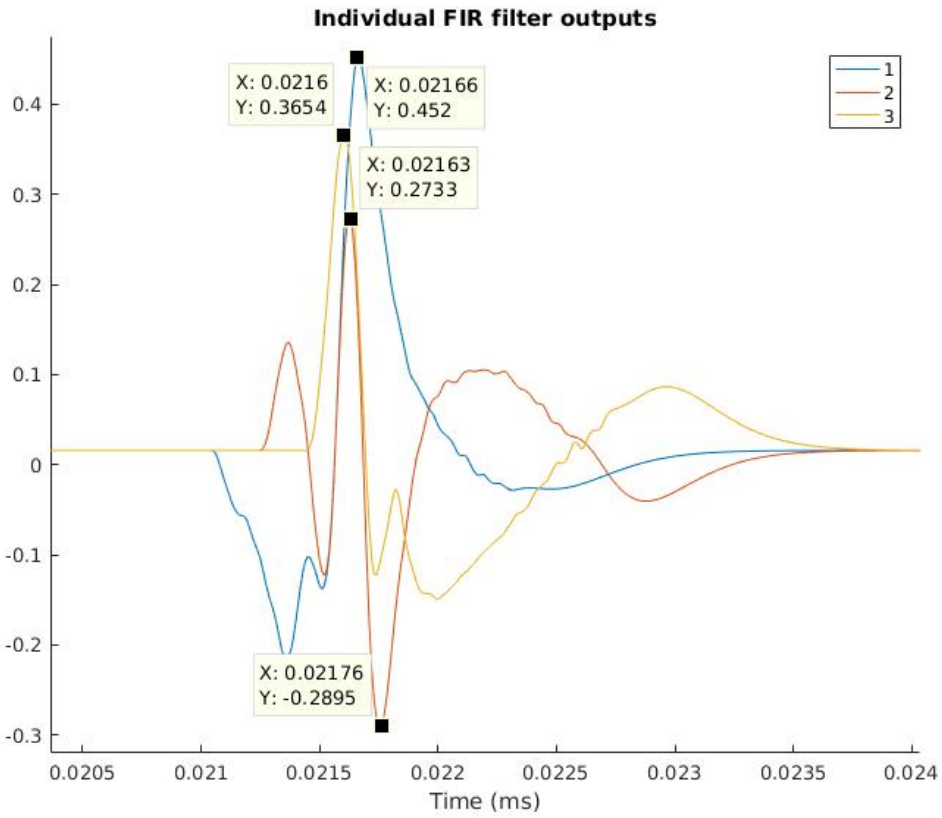


Figure 33: Timing of pulses on matched velocity of 50 m/s.

It is surprising to see that the neural net has not optimised the filter coefficients to perform delay-and-sum at the matched velocity, as previously thought. The timings of the pulses on Figure 33 show that they are not synchronised, but have a spread of $0.06\mu s$, and channel two is right in the middle of the other two. In addition, it can be noticed that the large negative peak of the middle channel does not coincide with the peaks of the other positive channels. Hence, no destructive interference occurs at the matched velocity. Surprisingly, the pulses are almost perfectly aligned at a different velocity than the matched. In fact, Figure 34 shows that the pulses are much better aligned at a velocity of 40 m/s.

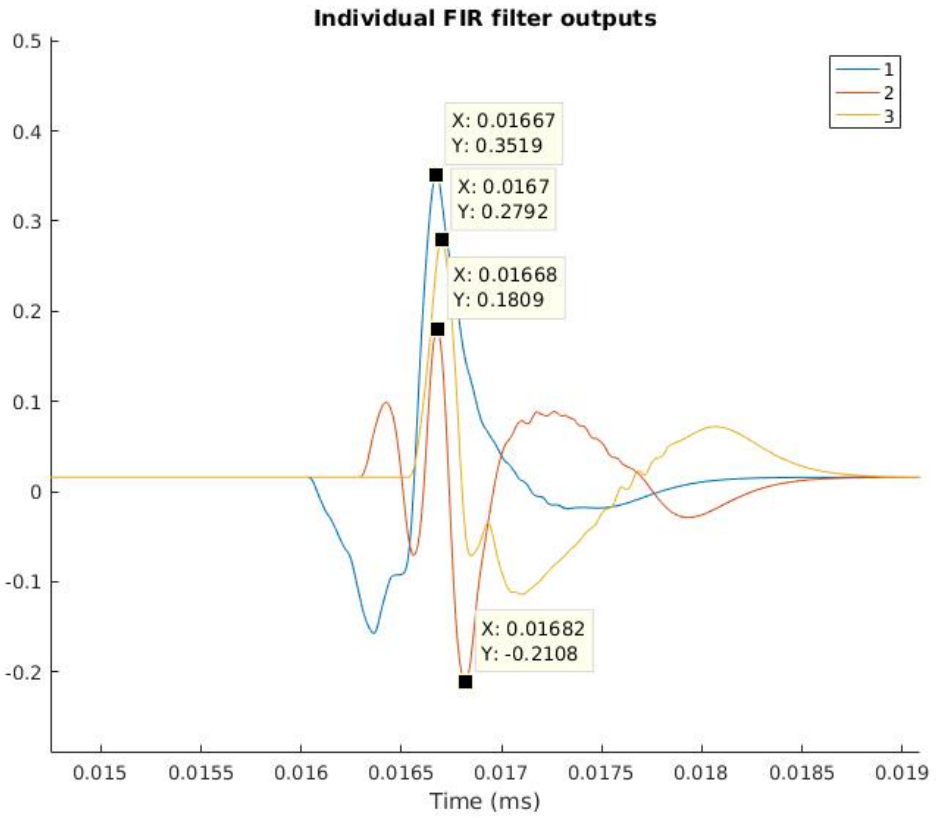


Figure 34: Timings of pulses at non-matched velocity of 40 m/s.

It is clear from Figure 34 that the pulses are much less spread out than in the matched velocity pulse, with a spread of $0.03\mu s$. Still the negative peak of the middle channel is non-relevant and causes no destructive interference with the positive peaks of the system. A theory that explains the fact that the pulses are aligned at 40 instead of 50 m/s, is that the v^2 term in Equation 16 is dominant. Namely, even if the pulses were perfectly aligned at 50 m/s, the pulses at higher velocities have large enough amplitudes so as to yield a higher output because the sum of their misaligned pulses is still larger than the sum of the aligned pulses of a lower velocity. For this reason the system optimises the filters to better align the pulses to a velocity lower than the matched one, which places the pulses of higher velocities further apart and thus interfere less constructively. This theory is going to be confirmed in the Validation section of this report (see Section 6).

Looking at one of the high velocities of 110 m/s, reveals another interesting optimisation with which the system manages to suppress high velocity amplitudes. Figure 35 shows that the spread of the peaks of the pulses increases to $3.25\mu s$, with the middle channel still appearing in the middle, as expected from the way that FIR filters work.

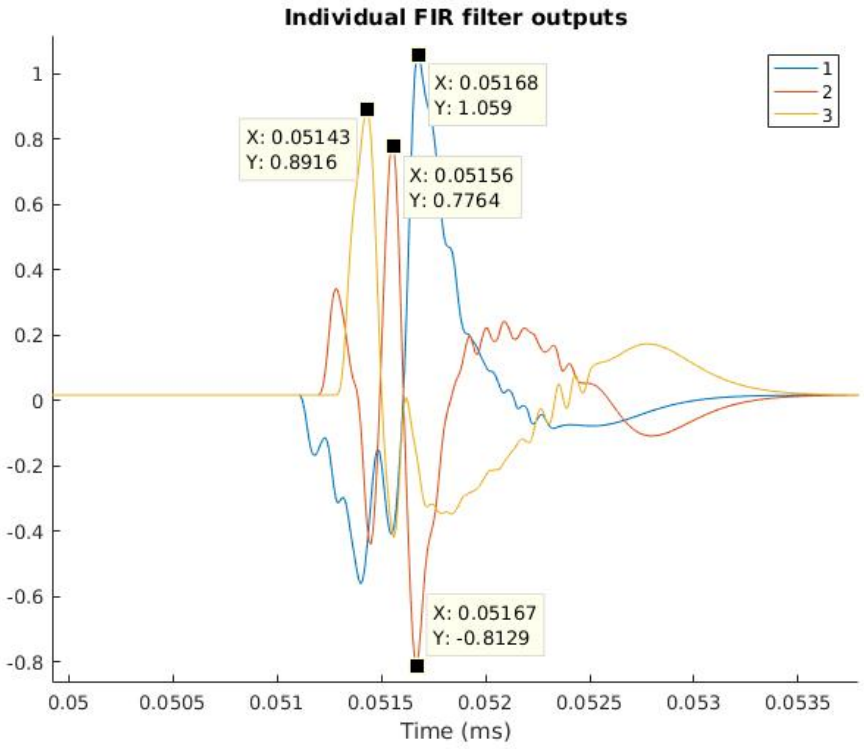


Figure 35: Timings of pulses at a velocity of 110 m/s.

However, it can be seen that the negative peak of channel 2 is almost aligned with the positive peak of the first channel. This creates a destructive interference pattern that suppresses the response of the system for high velocity APs.

Since channels 1 and 3 seem to have a similar response it is logical to try and sum the two signals and compare it with channel 2 in order to confirm the theory about the timings.

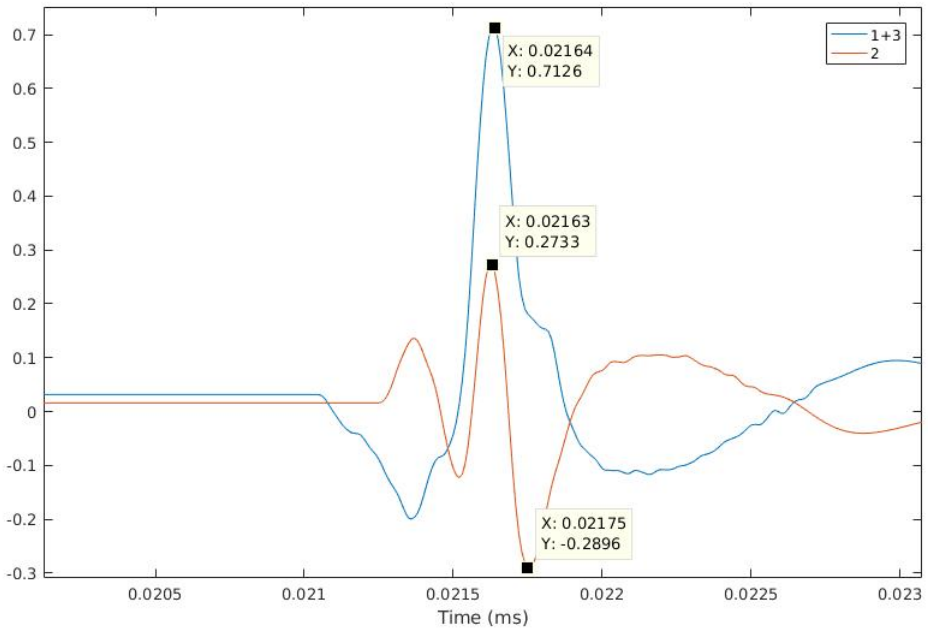


Figure 36: Sum of channels 1 and 3 overlaid with response from channel 2 for matched velocity of 50 m/s.

This figure shows that the positive peak from the sum of channels 1 and 3 is far away, in time, from the negative peak of channel 2, so no destructive interference occurs. In contrast, looking at a non-matched velocity below it is clear that the negative peak is aligned with the positive to suppress high velocity pulses.

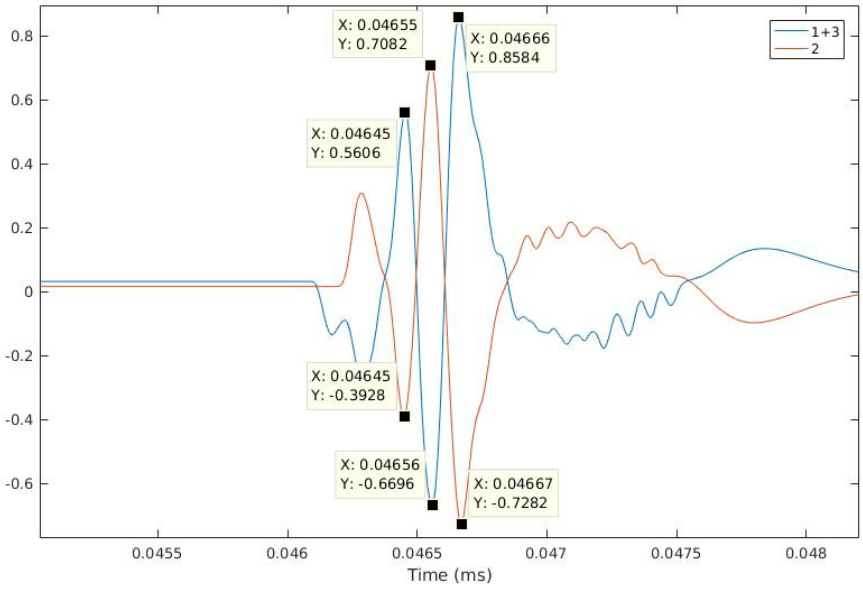


Figure 37: Sum of channels 1 and 3 overlaid with response from channel 2 for velocity of 100 m/s.

Lastly, it is interesting to examine the overall shape of the envelope produced by the sum of channels 1 and 3.

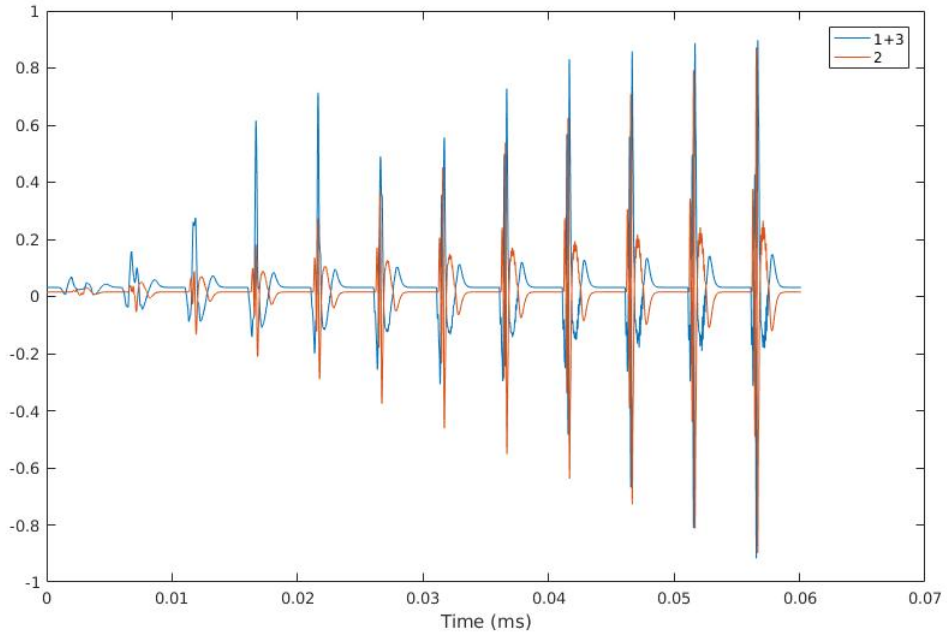


Figure 38: Sum of channels 1 and 3 overlaid with response from channel 2.

The graph shows that the two channels perform delay-and-sum. However, higher velocity pulses have higher magnitudes due to the v^2 term. Hence, the other channel is used to improve that result via the method described above to produce the response shown in Figure 30.

5.4 Real Rat Data Testing

One of the aims of this project was to investigate the reason the TDNN model failed to perform VSR with real data. The dataset available for this project was a series of ENG recordings from a rat. The process of extraction and the setup configuration have been described in Section 2.6. In order to perform this testing process the model was trained with simulated isolated bipolar data with a noise level of $2 \times 10^{-22} V$ in order to simulate some DC distortion.

The configuration described in Section 2.6, dictates a total length of the device is $5mm$. In rats the AP velocities range from 5 to 40 m/s, hence for a minimum velocity of 5 m/s the total device delay will be $1ms$. The sampling rate was $500kSa/s$, so the sampling period was $2\mu s$. However, to make the problem more computable the recordings were sub-sampled to $100kSa/s$, which corresponds to a sampling period of $10\mu s$. This means that the model requires 100 stages in order to be able to select the minimum velocity in the spectrum. The matched velocities chosen were 10, 20 and 30 m/s. The tuning curve of this setup is shown below.

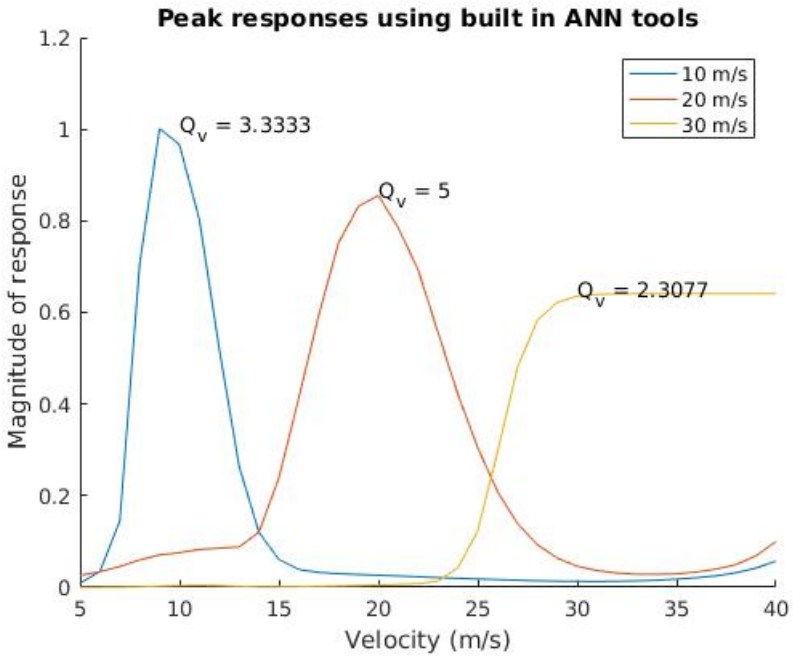


Figure 39: Tuning curve achieved for real rat data setup.

The model is not able to reach the performance of previous configurations because the inter-electrode spacing is too short, which makes VSR challenging the the pulses become too narrow. Now that the model has been trained with simulated data and noise it is logical to test it with real data inputs from the rat. The dataset chosen for this was the resting state recordings, as these had much better SNR than the stimulated one. A subset of the recordings were chosen as inputs due to their favourable SNR and distribution of pulses, which are shown below.

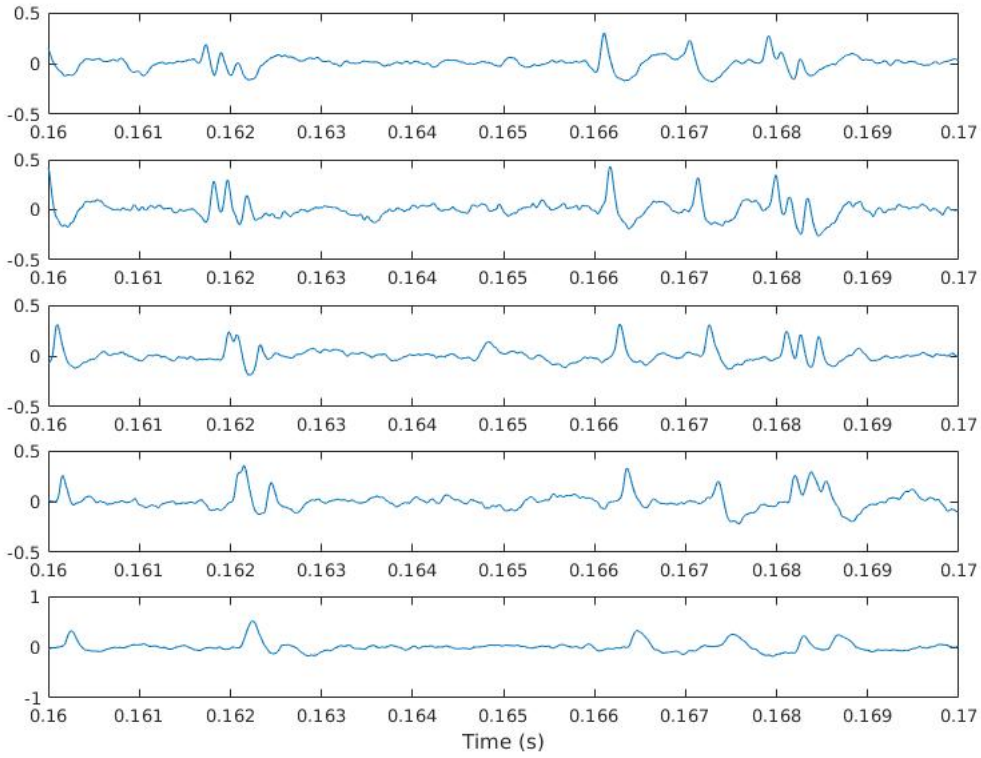


Figure 40: Subset of recordings chosen for testing.

The pulses in the input have been classified in the range of 7 to 12 m/s by measuring their time distribution and knowing the device length. The results are shown in Figure 41 below.

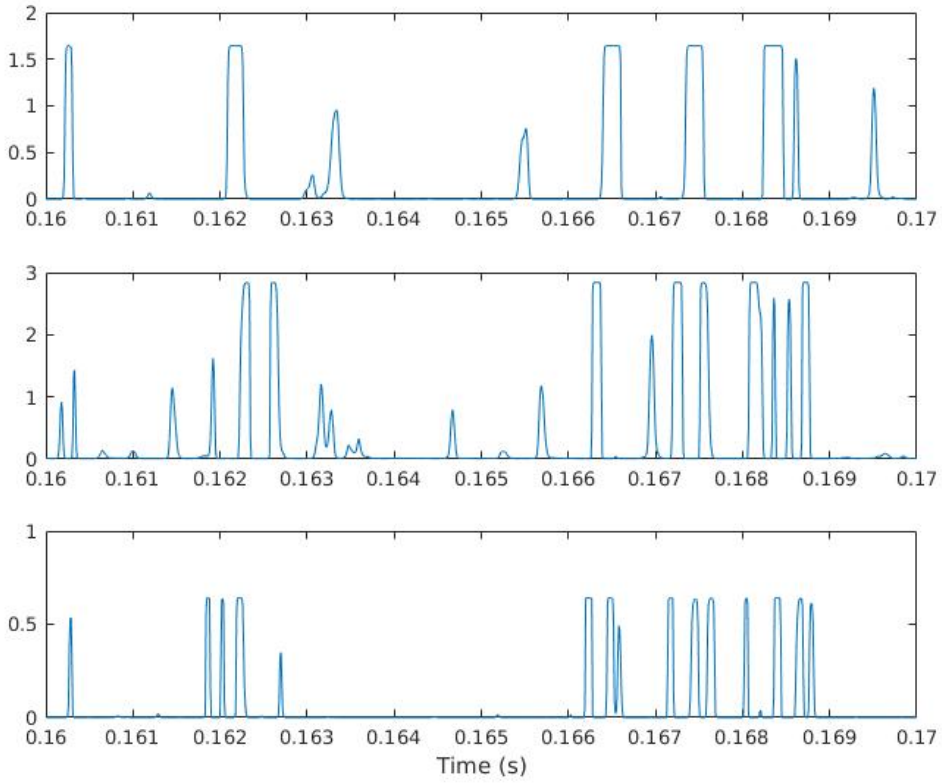


Figure 41: Output of trained model for real rat data in resting state.

It is clear that the models are behaving erratically. The ANNs are all firing at the same time and in theory if they were fully optimised they shouldn't produce any output since the pulses are not of matched velocity, especially with the models trained for 20 and 30 m/s. Hence, this test verifies the result obtained in previous research.

By having analysed the model's performance and operation in previous sections an explanation of the reasons that the model fails can be drawn. The ANN makes some clever timing arrangements between the channels in order to maximize the amplitude of the matched velocity. However, the real data suffer from DC distortions and interference from EMG signals (see section 2.2). These will not only shift the peaks of the pulses but also will shift the whole signal in the positive of negative direction for short periods of time. This will ruing the model's use of the sigmoid function. This stage depends on fine tuning the bias value in order to position the signal at an optimum value for suppression of the undesired pulses. DC distortion and other types of noise will severely affect the model. In addition to that, the whole setup of the recordings is far from ideal. The inter-channel spacings were not in fact, $1mm$ but varies from 0.8 to 1.2. This effect imposes serious challenges for the model.

6 Validation

By analysing the results from the simulations, a theory on how the TDNN manages to achieve such high selectivity Q_v values across the velocity range was formed. This section is going to try and validate the operation of the FIR stage of the model by designing a set of FIR filters (one for each channel) in MATLAB. Since the model under investigation had three filters with length of 120 stages, the design will need to set 120 coefficients for each of the three filters that are going to be implemented using the `filter` command in MATLAB.

The model to be designed needs to perform delay-and-sum on filters 1 and 3, and the other filter needs to align its negative peak with the positive peak of the sum of the other two for every non-matched velocity, as shown in Figure 37. In order to perform delay-and-sum, for each channel the number of delay samples needs to be calculated. The FIR filter then will be of the form:

$$y[n] = x[n - d] \tag{17}$$

Where $y[n]$ is the output, $x[n]$ is the input and d is the number of delay samples applied. Hence all the coefficients of the FIR filter will be zero except the one at position d, which will be 1. This will delay the signal by d samples. Each channel will be delayed by a different amount. For example, if there are three channels, all the channels need to be aligned with channel three. Hence, channel one will be delayed by d, but channel two by $2d$, assuming the channels are equidistant. The equation that describes d is the following:

$$d_i = (i - N) \times \frac{2.l}{v} \times f_s \tag{18}$$

Where d_i is the delay for channel i, N is the total number of channels, l is the inter-electrode spacing, f_s is the sampling frequency and v is the matched velocity. In Section 5.3 it was mentioned that only channels one and three do traditional delay-and-sum. Hence, only two of our filters were designed for this operation. The other filter is going to invert the signal by using a negative weight (-1) as a filter coefficient and is going to have a different delay, which is going to be explained later on in the report.

These weighted FIR filters also have the ability to scale the signal. This design scales channel one by a small amount of 0.5 (minor positive), channel three by 1.8 (dominant positive) and channel two by -1.8. These values are based on the ratios between the channels. A dominant positive channel and a similarly scaled negative channel are needed, in order for them to cancel out at large velocities, which will result in the response of channel 1 (minor). Simultaneously, at lower velocities destructive interference between them is not going to occur as effectively, and the dominant positive channel combined with the minor positive channel are going to have the maximum response, overcoming the v^2 term. Hence, the intelligence of the system relies on the ratios of the weights and not the values themselves.

In addition, in Figure 34 it was shown that the optimisation process for the ANN aligned the pulses much better for the 40 m/s velocity pulse for a matched velocity of 50 m/s. Thus, in the designed model, the delay d is calculated for 40 m/s and channels 1 and 3 are scaled by 0.5 and 1.8 factors, respectively. Using Equation 18, this translates to a

weight of 0.5 at position 30 for channel 1 and a 1.8 at position 1 (no delay) for channel 3. All the other coefficients for both filters are zero. The source code for the design and simulation of this system is included in the Appendix. For the same input used for the previous models (shown in Figure 23a), the summed output of the designed channels two and three is shown below.

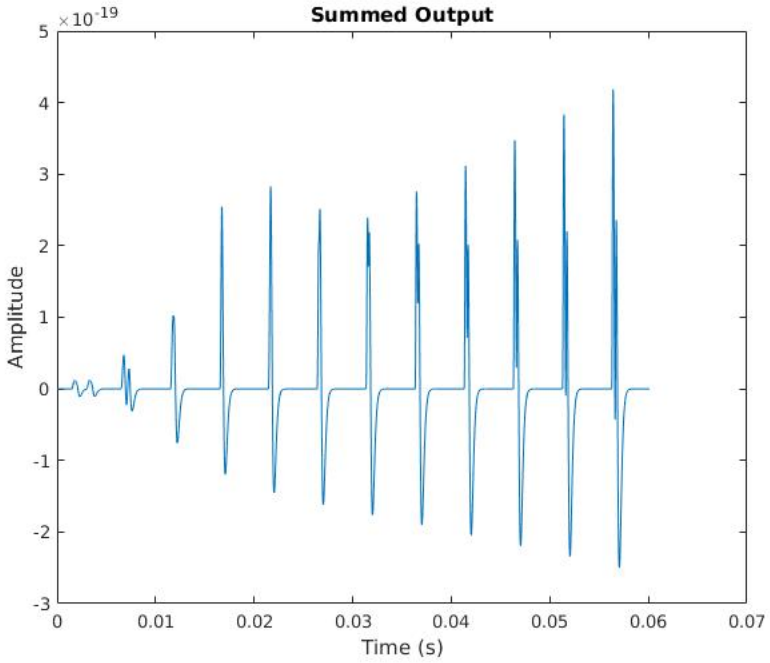


Figure 42: Summed output of designed filters 1 and 3.

The matched AP is the fifth pulses from the beginning. It can be seen that a similar envelope as in Figure 38 is achieved, with just the two channels performing delay-and-sum. This confirms the theory about the v^2 term, derived in Section 5.3. However, as mentioned above channel two has a different function. It has a different delay which causes it to align much more closely with higher velocities and not as well at the matched one (see Figure 35), while being inverted in order to cause destructive interference. Using Equation 18 channel 2 was designed for a delay of 100 m/s. The overlaid time responses of all the channels is shown in Figure 43 below for a matched velocity of 50 m/s, which occurs at $0.02162s$.

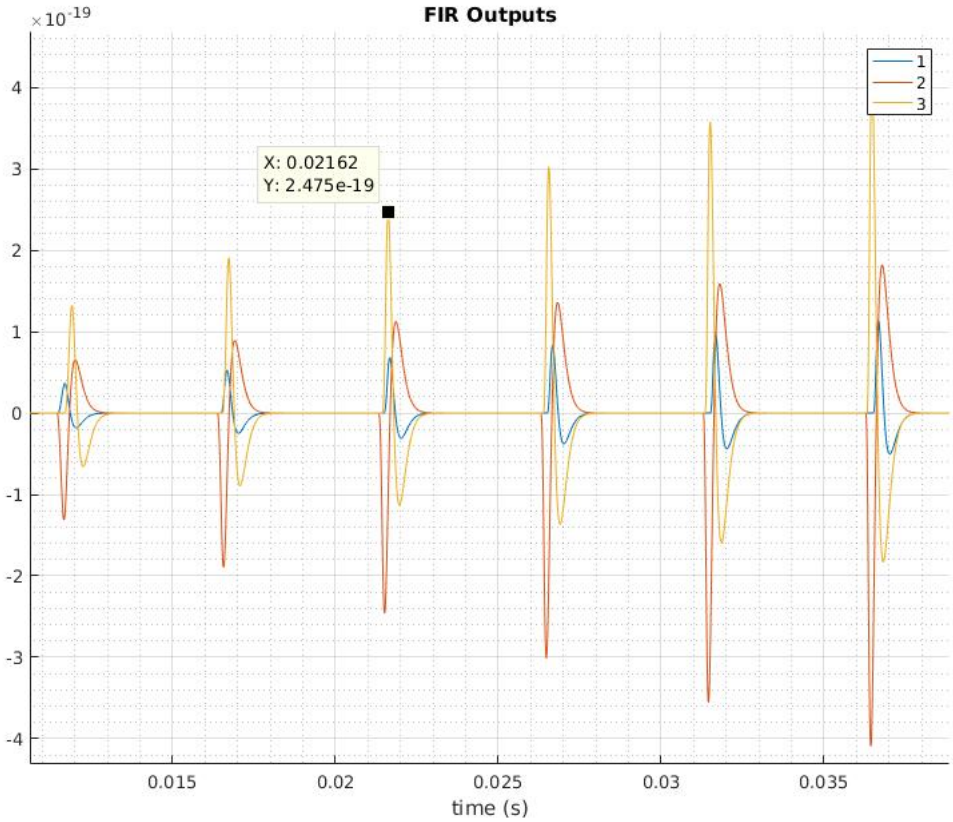


Figure 43: Overlaid designed responses of FIR filters. Filters 1 and 3 perform delay-and-sum and channel 2 is delayed for a velocity of 100m/s and is inverted.

It is clear that channels 1 and 3 (in blue and yellow respectively) align at 40 m/s and the inverted channel 2 (in orange) aligns perfectly with the AP of 100 m/s velocity. Now, summing the response from all the channels together yields the following time response.

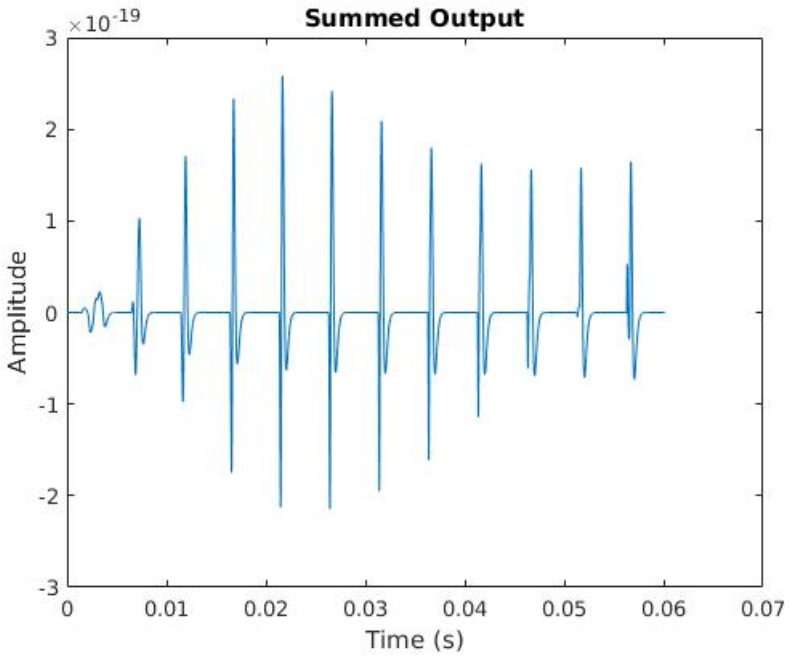


Figure 44: Time response of designed model.

This response is similar to the one shown in Figure 30, and is clear that the matched AP with velocity of 50 m/s at 0.02162 is selected. Having designed the system for this time response it is necessary to examine the tuning curves of this specific system for various velocities in the velocity spectrum (10 to 120 m/s). Since in the previous analyses the samples used were 20, 50 and 90 m/s for matched velocities, this is going to follow the same pattern.

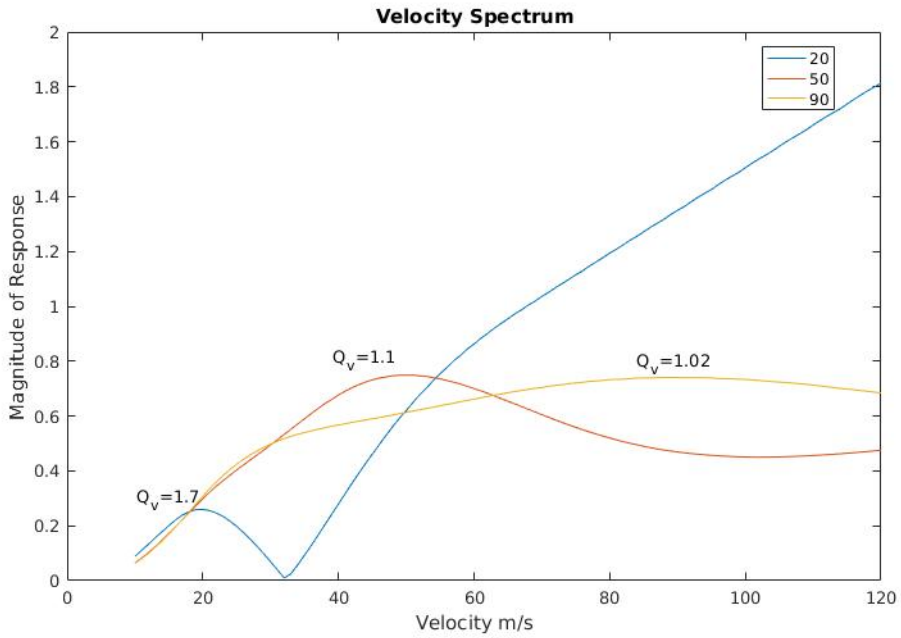


Figure 45: Tuning curve for designed system.

It can be seen that the selectivity of the system is not ideal. Although, it can select the matched velocity in every case, i.e. the maximum peak occurs at the matched velocity, the performance does not reach the one achieved by the optimised FIR filters of the ANN. A proposed theory in order to improve the performance of the system is based on an observation of the optimised FIR filter time response. Figure 31 showed that although the input APs were bipolar, the resulting pulses after the filters had more than one peaks. This effect inspired the use of the centroid filter described in Section 2.7. Figure 12 illustrates the effect of a centroid filter on an AP. The response of the filter resembles the response of the trained FIR filters. In addition, the use of the centroid is also based on the fact that it makes the pulses more narrow which can increase the selectivity, as destructive and constructive interference happens more abruptly. The centroid FIR filter was implemented in MATLAB and was chained to the delay FIR filters. The time responses of each channel for a sequence of APs ranging from 10 to 120 m/s, after the centroid and delay filters is shown in the Figure below.

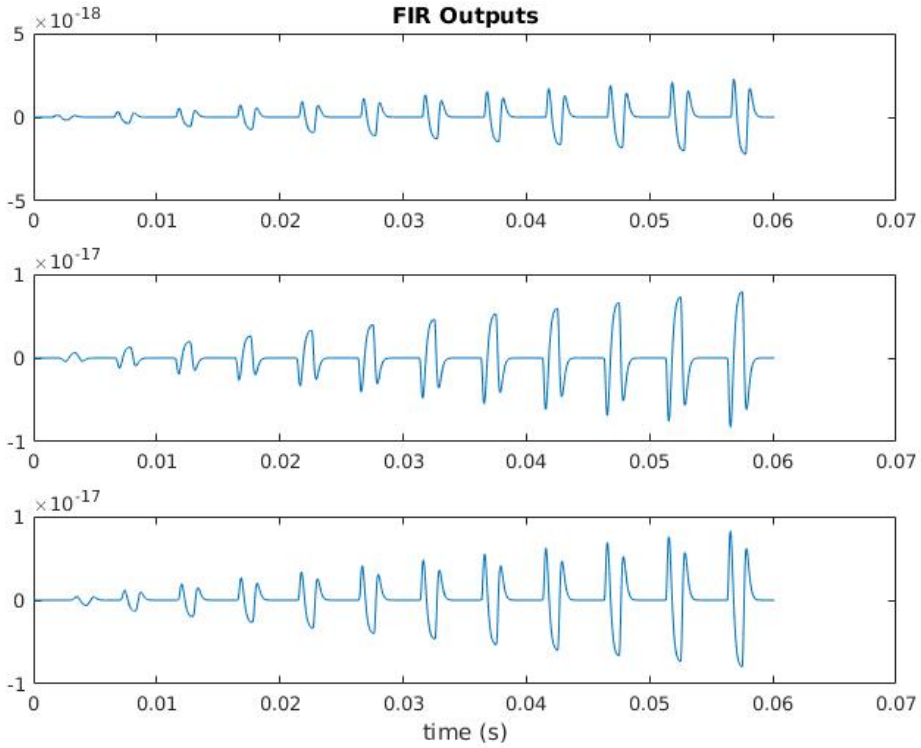


Figure 46: Time responses of channels with centroid filter for a matched velocity of 50 m/s.

This confirms the effect of the centroid filter proposed in Section 2.7 and combines it with the delay filters. The summed output of these models is shown below.

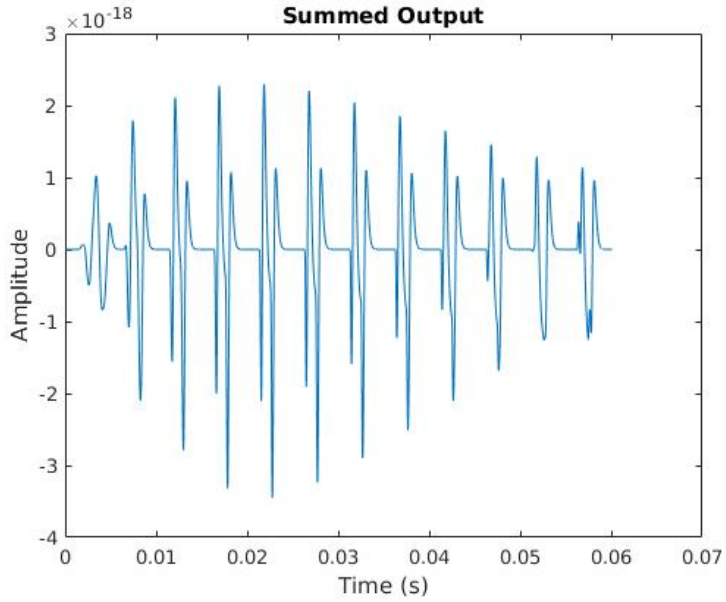


Figure 47: Time responses of complete system with three centroid filters and three delay filters for a matched velocity of 50m/s.

Now in order to confirm the assumption that the centroid is going to increase the selectivity of the system, the tuning curve of this model needs to be examined for velocities of 20, 50 and 90 m/s.

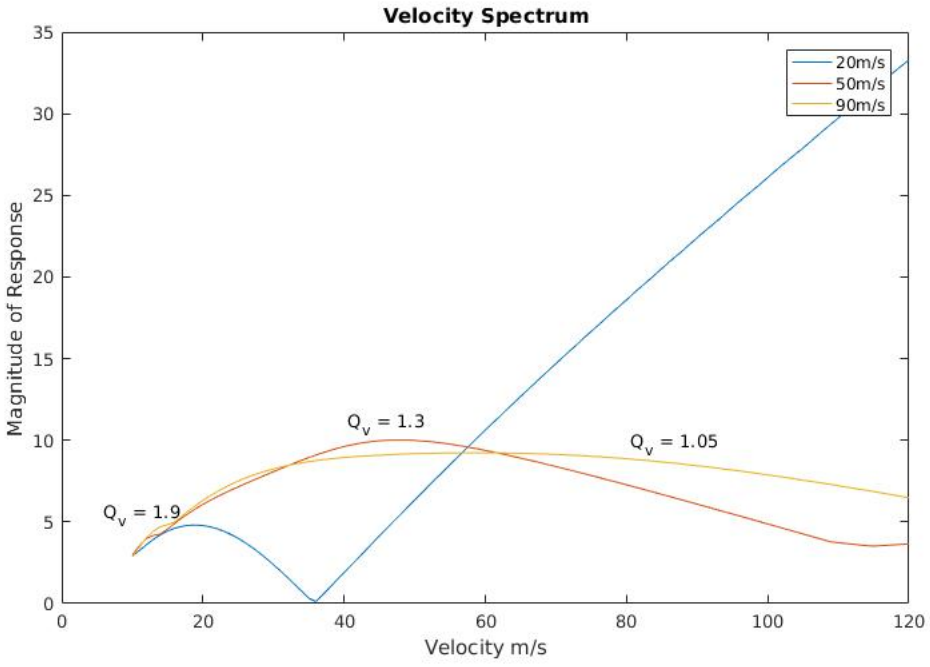


Figure 48: Tuning curves of system with centroid filters.

The Figure shows that the result is not much different than without the centroid filters. In fact, at 90 m/s performance is worsened. However, it is interesting to see that the

selectivity is slightly increased, which shows that such techniques have potential in VSR.

7 Discussion

7.1 Evaluation

This section is going to present the limitations of the methods and analyses employed, as well as, evaluate the achievements of this project.

First of all, this project clearly focused on the FIR stage of the TDNN model and did not investigate in detail the operation of the input layer bias value or output layer. These parameters are crucial for the operation of this model. The bias value in the input layer works in conjunction with the sigmoid transfer function. As mentioned in Section 5.2, it will adjust the signal in order for the sigmoid to severely suppress (almost to zero) all the non-matched APs. This achieves high Q_v values. This theory was not included in the validation section, as it was not applied to the time response of the designed FIR filters, hence it is not conclusive. The time scope of the project was such that more important and interesting aspects of the model were prioritised. Furthermore, the output layer, i.e. the neuron bias and weight values, were also not investigated in detail. Some of their functions were examined, such as the inversion of the signal when the model managed to do selectivity in the negative phase of the pulses, described in Section 5.2, but their values are still to be fully understood.

The ability of the model to perform VSR in the negative phase of the APs was mentioned. Although this phenomenon was observed and discussed, the TDNNs ability to perform this kind of signal processing is still elusive. This time response originates from the FIR filters, but currently, there is no theory that explains how the FIR filters can be optimised for this kind of operation, and what affects the optimisation algorithm to reach this solution. However, this can be theoretically explained by considering that there might not be a single solution for this problem. Conceptually, a simplified case might be that there is one solution in the positive solution space, but because of the symmetry of VSR, there will be a mirror solution in the negative space. The optimisation algorithm takes steps closer to these solution, but the step size might be such, that the model sometimes converges to the negative solution. However, this explanation needs to be tested and further investigated.

After the results of the simplified three channel model, with just the FIR stage connected to the output, were presented, it was concluded that two of the channels perform delay-and-sum for a slightly lower than the matched velocity, due to the v^2 term in Equation 16. The other channel is used to align with higher velocities to cause destructive interference, thus eliminating the magnitude of the impulse response of the model for high velocity APs. This theory was also validated by the designed FIR filters, but this project did not investigate how this theory scales to more channels. It is interesting to investigate the role that more channels play in the model's ability to increase selectivity across the velocity spectrum, which is a desired characteristic in VSR.

An important aspect of the project was the validation of the theories formed during the observation phase of time response analysis. However, the validation did not obtain the desired results. The optimised version of the model performed much better than the designed version. This can be seen by comparing the tuning curves of the two methods in Figures 29 and 45, respectively. The design proved some of the theories and success-

fully managed to select the matched velocity, which are quite important steps towards the final goal, but more investigation is required in order to achieve equal or higher performance than the optimised version. For example, the validation stage mentions the ability of the FIR filters to scale the signals in order to favour the matched velocity and the importance of the ratios of the scaling, but the actual scaling factors were not fully investigated in order to find the optimum solution, i.e. ratio.

An attempt to improve the designed filters' performance was the use of the centroid filter, which was inspired by observation of the time response of the optimised version of the model. The centroid filter was successfully implemented, but its application did not yield the expected result. Although, there were slight improvements, they were not considerable. Due to the fact the the centroid changes the timings, as well as the number of the peaks of the APs, further analysis is needed, in order to pinpoint the exact time alignment relationships of the peaks and troughs of these more complicated pulses.

One of the aims of the project was to investigate the reason the model was failing with real data. The project managed to confirm that the model failed to accurately select the matched velocities and behaved erratically. Simulations with the real data were challenging due to the high computational requirements of the recorded signals. The inter-electrode spacings were too narrow and the signals were sampled at such high velocities that the model became too large. In addition, the inter-electrode spacings were not rigid. Namely, the actual distances varied between the channels, making the timings of the pulses different than the ones predicted. This was accounted for by applying electro-shifts to simulations used for training, but this did not yield the required result. A theory that explains the failure of the model in this situation is that the sigmoid function is sensitive to DC distortions, as mentioned in Section 5.4. Finally, a limitation of the model with the application of real data is the fact the it was trained with modelled AWGN. Frequency analysis on the rat recordings revealed that the main noise component resembles Brownian noise. Hence, the simulation was not accurate enough for training the model. Also, simulations with a more computable dataset would be extremely valuable.

7.2 Future Work

Having analysed the limitations of the project it is necessary to propose ways to overcome these limitations, which did not fit in the time scope of this project. Drawing from the limitations of the project discussed in the Evaluation section, the designed FIR filters lack much of the desired characteristics the trained version has. In order to achieve the same level of performance one of the most fundamental steps is to understand the values of the weights and bias values of the trained model. A first step could be to look at the values that the system converges to and try to understand the role they have in conjunction with the sigmoid transfer function of the first layer. This has the potential to provide the key to achieving high Q_v values across the velocity spectrum.

As mentioned in the Evaluation section, the study that designed the FIR filters did not focus on optimising the scaling values. These ratios play an important role in the way the system works and it is of critical importance to find the optimal ratio. In addition, since the designed FIR filters were far from equivalent to the optimised version,

it is critical to understand what is missing from the model. Applying the centroid filter was an important step, but deep analysis on the timings of the phases of these pulses is needed in order to boost the systems performance and achieve the operation shown in Figure 37, where all the negative and positive phases of the pulses are perfectly aligned. There is clearly a lot of work to be done in designing these FIR filters more efficiently and after that is done, the design must be scaled to multiple channels, which is challenging.

An interesting technique has been discussed in [16] for AP velocity detection. This algorithm looks at each response from multiple velocity APs, $V_d[n, dt]$, and detects the matched velocity using the criterion, $V_{d-1} < V_d > V_{d+1}$. This essentially, detects the envelope of the response of the system to detect a specific velocity. This technique shows potential and is worth implementing and testing it with the ANN model or the designed FIR filters.

In this project the centroid filter was used with the mindset that it makes the alignment of pulses more robust to noise and the pulses more narrow, thus improving the selectivity potential. However, other filters can also be used in order to improve the performance of the system. A proposed idea is to use a matched filter to detect specific velocity APs in real data with noise and interference. A matched filter has an impulse response described by:

$$g(t) = c.h(T - t) \tag{19}$$

This equation maximises the output SNR at time $t = T$ [22]. Thus, it is said that the filter is matched to the signal $h(t)$. If $h(t)$ is designed to be the expected waveform from the matched velocity AP, then in theory, even in the presence of noise the system will provide a peak at the matched AP velocity. This is definitely an idea worth exploring in the future.

An additional, idea in order to help the system perform VSR with real data is to train it with real noisy data. In this way the optimisation algorithm will converge to a different solution, which may be optimum for real AP detection. In turn, the model can be investigated and its operation can be transferred to the designed FIR filters.

7.3 Conclusion

This project has taken previous MATLAB models and simulation of the TDNN, produced by other research, and has extended them in order to investigate and understand its operation. Although the TDNN is still not fully understood, its key parameters, such as the FIR filters and sigmoid transfer function, have been decoded by observing the time responses and using signal processing theory. Some hypotheses were made about the FIR filters' inner workings, which were validated by designing FIR filters that have the same function. Although the design's performance was not equivalent to the optimised model, some important steps were made in the right direction. Future work needs to be done in order to find the missing properties of the optimised model, but it is obvious that there is promising potential in optimally designing these FIR filters. Also, one of the goals of the project was to investigate the behaviour of the system with real data. Although a solution was not suggested, the reasons for the erratic behaviour of the model are explained.

All in all, the aims of the project were achieved. Although the design of the FIR filters

was not fully successful, this project's source code and report should be enough for a skilled engineer to finish the design. If this is achieved the optimisation process will be eliminated. In addition, the full understanding of the TDNN will inspire the extension of its performance beyond what is achieved today. This will have extensive impact in real time processing of ENG signals, and evolve FES to a closed loop method, which solves the current issues encountered in many patients, as mentioned in Section 1.

References

[1] J. Taylor, A. Al-Shueli, C. Clarke, N. Donaldson, "Improved Signal Processing Methods for Velocity Selective Neural Recording Using Multi-Electrode Cuffs", IEEE, Volume: 8, Issue: 3, June 2014.

[2] J.Taylor, "Neural Recording"[Lecture], University of Bath, slide 5, 4, November, 2016.

[3] https://en.wikipedia.org/wiki/Central_nervous_system, 2017 [image]

[4] OpenStax. (6 Mar 2013). Anatomy and Physiology [Online]. Available: https://opentextbc.ca/anatomyandphysiology/ [image]

[5] Steven C. Schachter, MD, Joseph I. Sirven, MD. (Aug 2013). Vagus Nerve Stimulation [Online]. Available at: http://www.epilepsy.com/learn/treating-seizures-and-epilepsy/devices/vagus-nerve-stimulation-vns

[6] A. D. Back, A. C. Tsoi, "FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modelling", Neural Computation, Volume: 3, Issue: 3, Sept. 1991.

[7] M. S. Lewicki, "A review of methods for spike sorting: the detection and classification of neural action potentials," Network: Comput. Neural Syst., vol. 9, pp. R53–R78, 1998.

[8] Jack W. Judy, Dejan Markovic, "Spike Sorting: The First Step in Decoding the Brain", IEEE Signal Processing Magazine, Jan 2012.

[9] Nicholas G. Hatsopoulos, John P. Donogue, "The Science of Neural Interface Systems", Annu Rev Neurosci., vol. 32, pp. 249-266, 2009.

[10] Taylor, J., Clarke, C., Schuettler, M. and Donaldson, N,(2012). "The theory of velocity selective neural recording: a study based on simulation". Med. & Biol. Eng. & Comput., 42 (5), pp. 309-318, 2012.

[11] J. Taylor, R. Rieger, C. T. Clark, "Signal Processing for Velocity Selective Recording Systems Using Analogue Delay Lines", IEEE, May 2012.

[12] A. Vlissidis, "Time Delay Neural Network for Velocity Selective Recording", *Literature Review*, pp. 1-5, Feb. 2017.

[13] N. Donaldson, R. Rieger, M. Schuettler, and J. Taylor, "Noise and selectivity of velocity-selective multi-electrode nerve cuffs," Med. Biol. Eng. Comput., vol. 46, pp. 1005–1018, 2008.

[14] Taylor J, Donaldson N, & Winter J, (2004). "Multiple-electrode nerve cuffs for low velocity and velocity-selective neural recording". Med. & Biol. Eng. & Comput., 42 (5), pp. 634-43.

[15] T. Lefurge, E. Goodall, K. Horch, L. Stensaas, and A. Schoenberg, "Chronically implanted intrafascicular recording electrodes," Annals of Biomedical Engineering, vol. 19, pp. 197–207, 1991.

[16] B. Metcalfe, D. J. Chew, C. T. Clarke and J. Taylor, "A New Method for Spike Extraction Using Velocity Selective Recording Demonstrated with Physiological ENG in Rat", Journal of Neuroscience Methods, May 2015.

[17] B. Metcalfe, C. T. Clark, N. Donaldson and J. Taylor, "A New Method for Neural Spike Alignment: The Centroid Filter", IEEE Transactions on Neural Systems and Rehabilitation Engineering.

[18] Abramowitz, M. and Stegun, I. A. (Eds.). "Hyperbolic Functions." §4.5 in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 9th printing. New York: Dover, pp. 83-86, 1972.

[19] D. Marquardt, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," SIAM Journal on Applied Mathematics, vol. 11, no. 2, pp. 431–441, June, 1963.

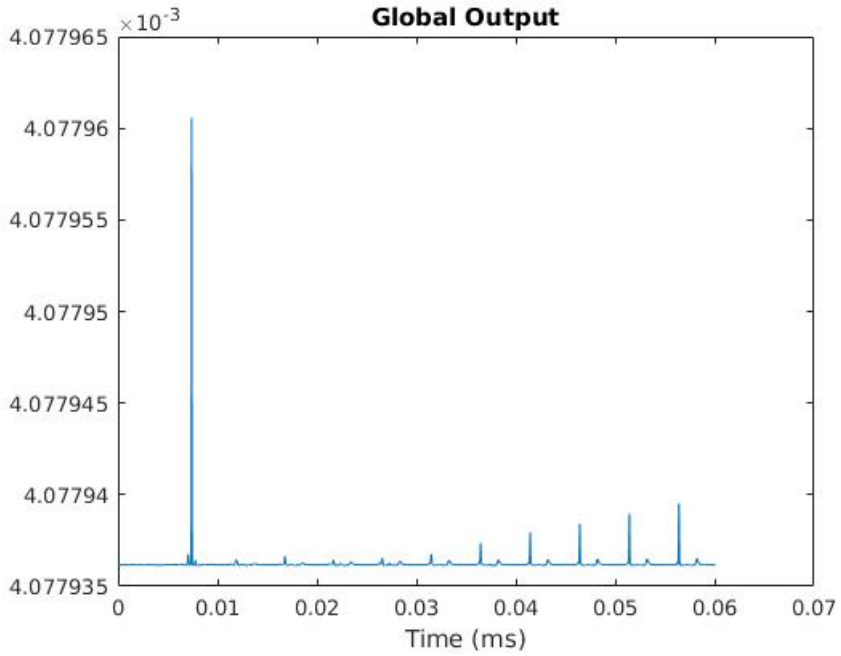
[20] M. T. Hagan and M. Menhaj, "Training feed-forward networks with the Marquardt algorithm," IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989–993, 1994.

[21] L. F. Tiong, "Neural Networks to Analyse Neural Signals", University of Bath, Final Project Report, May 6 2016.

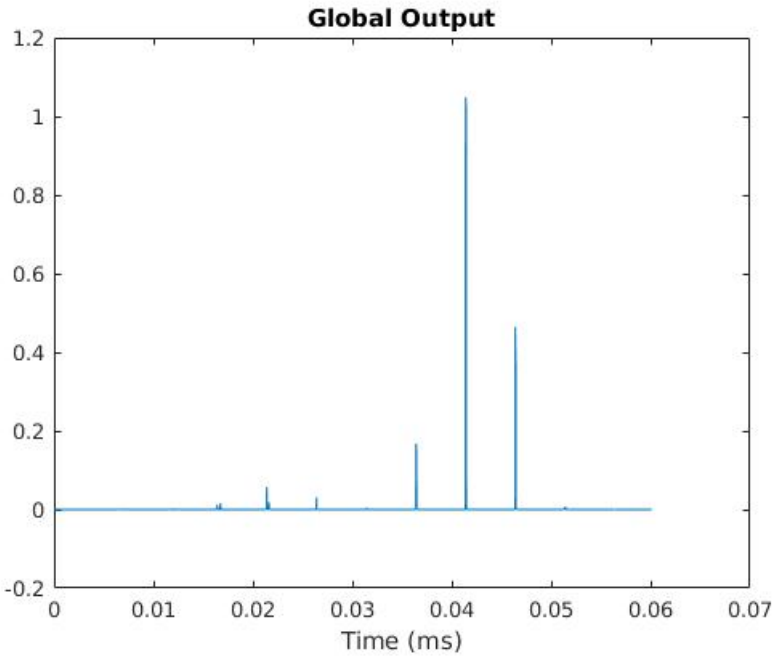
[22] A.P. Clark, "Principles of Digital Data Transmission", Pentech Press Limited, §12 Matched Filter Detection, pp. 96-108.

Appendices

COMPLETE MODEL TIME RESPONSE



(a) Complete model output for a matched velocity of 20 m/s.



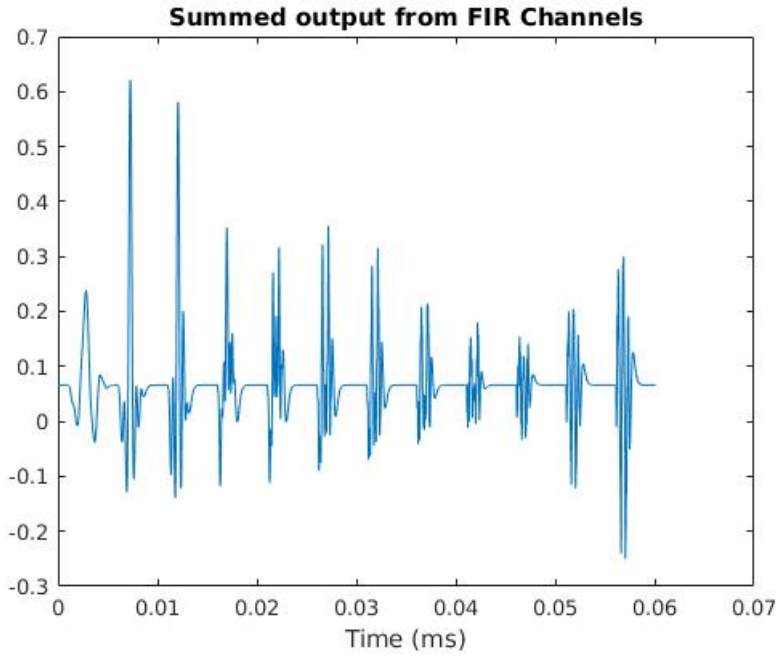
(b) Complete model output for a matched velocity of 90 m/s.

Figure 49: Model outputs at various matched velocities.

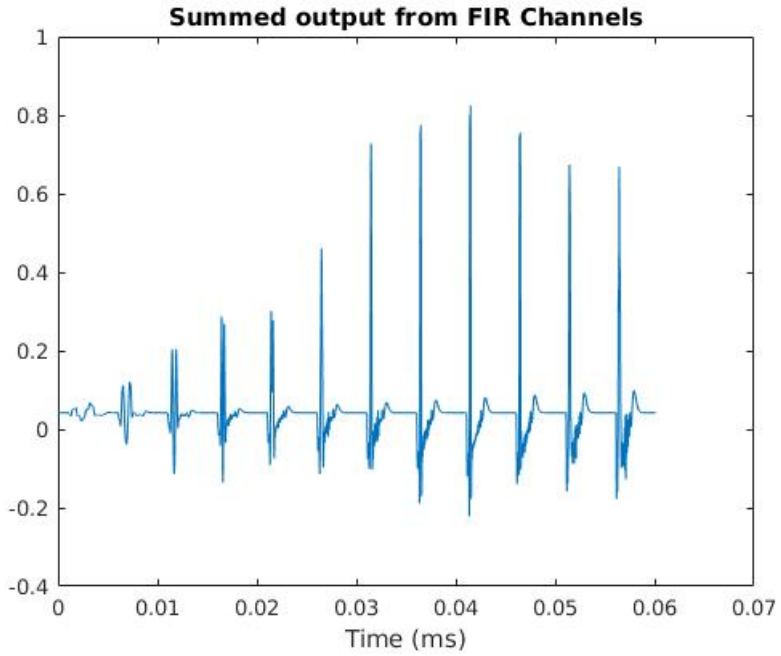
FIR STAGE TIME RESPONSE SOURCE CODE

```
%% Plot each FIR filter output individually
testnet = ANN(ANNindex).net;                    % copy neural network
testnet.outputConnect = [1 0];                  % connect output to first layer
testnet.biasConnect = [0; 0];
testnet.layers{1}.transferFcn = 'purelin';

figure;
hold on
% Pre-allocate for speed
AnnOutput = zeros(DataLines, numel(SequenceTime));
for ChannelIndex = 1:DataLines
    % Apply the AGC
    NormInput = AgcSim(InputSequence);

    % Set every other channel to zero
    NormInput(1:end ~= ChannelIndex, :) = zeros(DataLines-1, numel(
    SequenceTime));

    % Convert data format to be input to the ANN from a R-by-TS matrix
    % to a 1-by-TS cell array of R-by-1 vectors
    InputCellArray = con2seq(NormInput);

    % Run the ANN
    AnnOutputCellArray = sim(testnet, InputCellArray);

    % Convert ANN output back to a conventional array
    AnnOutput(ChannelIndex,:) = cell2mat(AnnOutputCellArray);

    % Plot channel time response
    subplot(DataLines, 1, ChannelIndex);
    plot(SequenceTime, AnnOutput(ChannelIndex, :));
    if ChannelIndex == 1
        title('Individual FIR filter outputs');
    end
end
```

FIR STAGE TIME RESPONSE



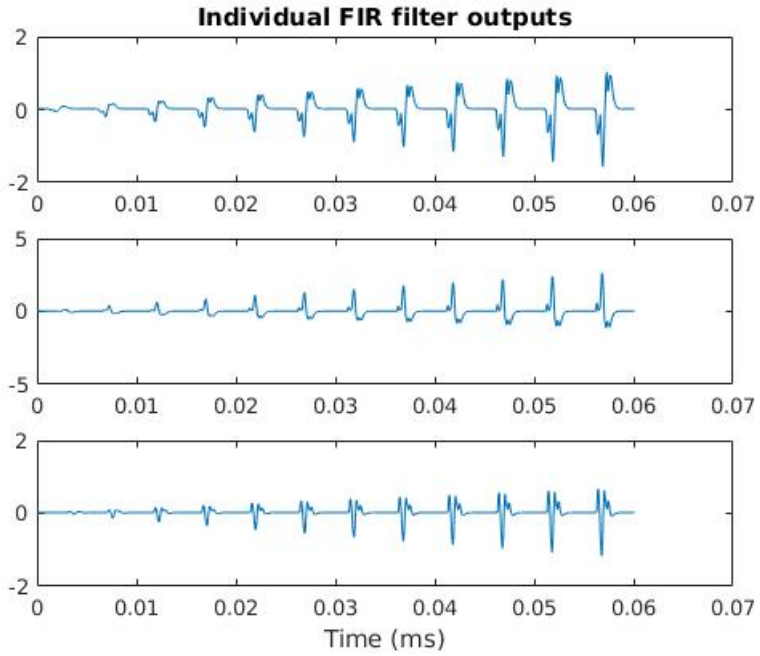
(a) FIR stage output for a matched velocity of 20 m/s.



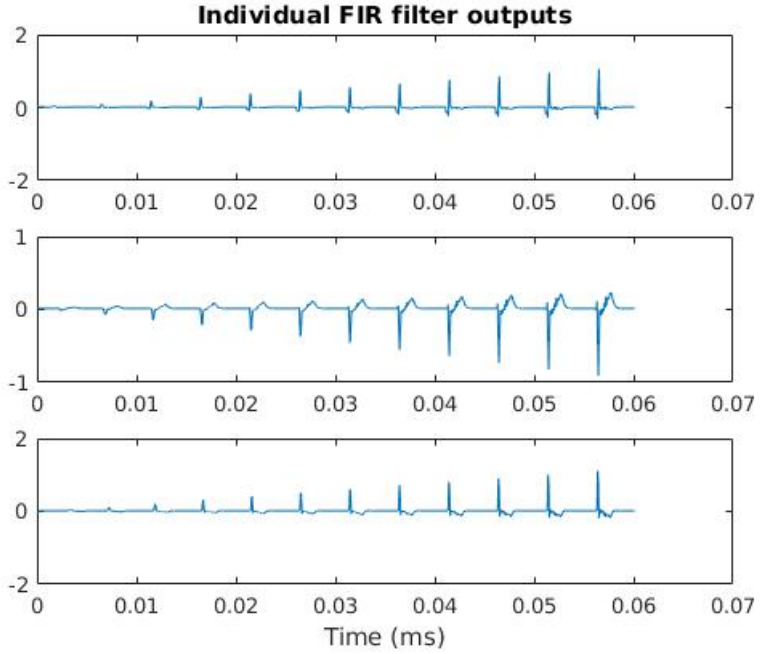
(b) FIR stage output for a matched velocity of 90 m/s.

Figure 50: Model outputs at various matched velocities.

INDIVIDUAL FIR CHANNEL TIME RESPONSE



(a) Individual FIR channel time response for a matched velocity of 20 m/s.



(b) Individual FIR channel time response for a matched velocity of 90 m/s.

Figure 51: Model outputs at various matched velocities.

FIR FILTER DESIGN SOURCE CODE

```
function design_fir(Parameters, velocity, v2)
% Design FIR filters to perform VSR
%
% A. Vlissidis

% Generate the input signal x
velocities = 10:1:120;
InitTime = 0.001;
EndTime = 0.004;
time = -InitTime:1/Parameters.SamplingFrequency:EndTime;
timelen = numel(time);
global_time = (0:(timelen * numel(velocities)) - 1) ...
                    / Parameters.SamplingFrequency;

data = zeros(numel(velocities), 3, timelen);
for i = 1:numel(velocities)
    % Get the velocity
    v = velocities(i);

    % Calculate the time range
    start = 1 + (i - 1) * timelen;
    tend = start + timelen - 1;

    % Insert the AP into the input sequence
    x(:, start:tend) = GetBiPolar(Parameters, v, time);
    data(i, :, :) = GetBiPolar(Parameters, v, time);
end
data = AgcSim(data);
y = zeros(size(x)); % Allocate the space for the output

% Initialize the coefficients for the filters
a = 1; % FIR
b = zeros(Parameters.channels, Parameters.AnnLength(1)); % Preallocate for
    speed
c = zeros(Parameters.channels, Parameters.AnnLength(1));

Plot the input
figure;
for channel = 1:Parameters.channels
    subplot(Parameters.channels, 1, channel);
    plot(global_time, x(channel, :))
    if channel == 1
       title('Input Action Potential')
    end
end
xlabel('time (s)')

% Calculate the interchannel delay in number of samples
n = Parameters.ElectrodeSpacing * 2 * Parameters.SamplingFrequency/velocity
    ;
n2 = Parameters.ElectrodeSpacing * 2 * Parameters.SamplingFrequency/v2;

figure;
hold on
for channel = 1:Parameters.channels
    % Apply centroid
    b(channel, :) = -2 * (1:Parameters.AnnLength(1))/Parameters.AnnLength
```

```
    (1) + 1;
    x(channel, :) = filter(b(channel, :), a, x(channel, :));

    if channel < Parameters.channels
        % Apply delay
        N = (Parameters.channels - channel) * round(n);
        N2 = (Parameters.channels - channel) * round(n2);
        c(channel, :) = 0;
        if mod(channel, 2) == 0
            c(channel, N2) = -1.803;
        else
            c(channel, N) = 0.3;
        end
        y(channel, :) = filter(c(channel, :), a, x(channel, :));
    else
        y(channel, :) = 1.813 * x(channel, :); % scale last channel
    end
    % Plot the result
    subplot(Parameters.channels, 1, channel);
    plot(global_time, y(channel, :))
    if channel == 1
      title('FIR Outputs')
    end
end
xlabel('time (s)')
legend('1', '2', '3')
hold off

% Sum the output of each channel
h = sum(y);
figure;
plot(global_time, h);
title('Summed Output');
xlabel('Time (s)');
ylabel('Amplitude');

peaks = zeros(1, numel(velocities));
for i = 1:numel(velocities)
    % Get the velocity
    h = sim_fir(b, c, Parameters, reshape(data(i, :, :), 3, 501));
    peaks(i) = max(abs(h));
end
% Normalise
peaks = peaks./max(peaks);

figure;
plot(velocities, peaks)
title('Velocity Spectrum')
xlabel('Velocity m/s')
ylabel('Magnitude of Response')
end

function h = sim_fir(b, c, Parameters, x)
%SIM_FIR Summary of this function goes here
%   Detailed explanation goes here

a = 1;
y = zeros(size(x)); % Allocate the space for the output

for channel = 1:Parameters.channels
```

```
    % Apply centroid
    x(channel, :) = filter(b(channel, :), a, x(channel, :));

    if channel < Parameters.channels
        % Apply delay
        y(channel, :) = filter(c(channel, :), a, x(channel, :));
    else
        y(channel, :) = 1.813 * x(channel, :); % scale last channel
    end

end

% Sum the output of each channel
h = sum(y);
end
```

DESIGN ANN SOURCE CODE

```
function ANN = DesignAnn (AnnSelections, Parameters)
% Design ANNs using the built in training systems to detect APs at
% particular velocities.
%
% C.T.Clarke based on the work of Assad al Shueli
% Edited by L F Tiong 06/05/2016
% Edited by A Vlissidis 06/05/2017
%
% Use a parameter structure like this:
%
% Parameters = struct (                 ...
%   'Electrodes',              11     , ...
%   'ElectrodeSpacing',         0.003, ...
%   'SamplingFrequency',   100000     , ...
%   'ActionPotentialType', 'long'     , ...
%   'StartTestVelocity',        1     , ...
%   'StepTestVelocity',         1     , ...
%   'EndTestVelocity',        100     , ...
%   'NoiseLevel',               0.01 );
%
% Other parameters may be included but will be ignored

% Load all ANNs which have been stored as array called
% ANN(i).net where i is the ANN index that includes all
% the values in AnnSelections below.

% Get the number of ANNs to train
NumAnns = numel(AnnSelections);
% Pre-allocate for speed
ANN(1:NumAnns) = struct('net', []);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Setup required variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Handle to set action potential type
if strcmp(Parameters.APType, 'UniPolar')
    GetData = @GetUniPolar;
    DataLines = Parameters.Electrodes;
elseif strcmp(Parameters.APType, 'TriPolar')
    GetData = @GetTriPolar;
    DataLines = Parameters.Electrodes - 2;
elseif strcmp(Parameters.APType, 'BiPolar')
    GetData = @GetBiPolar;
    DataLines = floor(Parameters.Electrodes/2);
end

% Set up the time sequence for the signals at each velocity
InitTime = 0.001;
EndTime = 0.004;
Time = -InitTime:1/Parameters.SamplingFrequency:EndTime;
TimeLength = numel(Time);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design each ANN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for AnnIndex=1:NumAnns
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create input signals for training
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Retrieve the matched velocity of the selected ANN
MatchedVelocity = AnnSelections(AnnIndex);

% Set the initial Velocity set including the repeated matched set at
% the end that match the ANN's target velocity
Velocities = cat(2,Parameters.StartTestVelocity: ...
    Parameters.StepTestVelocity:Parameters.EndTestVelocity, ...
    MatchedVelocity*ones(1,Parameters.MatchRepeats));

NumVelocities = numel(Velocities);

% Set up the array to hold the input training data
TrainingInputSequence = zeros(DataLines, ...
                        NumVelocities * TimeLength);

% Create an input sequence for the ANN
for VelocityIndex = 1:NumVelocities

    % Retrieve the selected velocity
    Velocity = Velocities(VelocityIndex);

    % Get a data set of the appropriate velocity
    Data = GetData(Parameters, Velocity, Time);

    % Insert the data into the input sequence
    InsertStart = 1 + (VelocityIndex - 1) * TimeLength;
    InsertEnd = InsertStart + TimeLength - 1;

    TrainingInputSequence (:,InsertStart:InsertEnd) = Data;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create an output target signal for training
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear the target
TrainingTargetSequence = zeros(1,NumVelocities*TimeLength);

% Add a pulse for each velocity that is the same as the matched
% velocity
VelocityIndices = 1:NumVelocities;
for VelocityIndex = VelocityIndices(Velocities == MatchedVelocity)

    % Get the insertion point for the data in the input sequence
    InsertStart = 1 + (VelocityIndex - 1) * TimeLength;
    InsertEnd = InsertStart + TimeLength - 1;

    % Get a data set of the appropriate velocity with no noise
    NoiseLessParameters = Parameters;
    NoiseLessParameters.NoiseLevel = 0;
    Data = GetData(NoiseLessParameters, MatchedVelocity, Time);

    % Get the final channel
    FinalChannel = Data(DataLines,:);
```

```
    % Normalise it
    FinalChannel = AgcSim(FinalChannel);

    switch lower(Parameters.TargetOutput)
        case 'pulse'
            % Create a pulse at the 3dB points
            TargetPulse = zeros(size(FinalChannel));
            TargetPulse(FinalChannel >= (max(FinalChannel)/sqrt(2))) =
1;

            % Apply it to the target sequence
            TrainingTargetSequence(InsertStart:InsertEnd) = TargetPulse
;
        case 'ap'
            % Target sequence is action potential with matched velocity
            TrainingTargetSequence(InsertStart:InsertEnd) =
FinalChannel;
        otherwise
            disp('Unknown output training target type.')
    end;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create input signals for validation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%2e-21;
valParameters = Parameters;
valParameters.StepTestVelocity = 5;
valParameters.NoiseLevel = 2e-22;
valParameters.MatchRepeats = 5;

% Set the validation Velocity set including the repeated matched set at
% the end that match the ANN's target velocity
Velocities = cat(2,valParameters.StartTestVelocity: ...
    valParameters.StepTestVelocity:valParameters.EndTestVelocity, ...
    MatchedVelocity*ones(1,valParameters.MatchRepeats));
NumVelocities = numel(Velocities);

% Set up the array to hold the input training data
ValidationInputSequence = zeros(DataLines, ...
                          NumVelocities * TimeLength);

% Create an input sequence for the ANN
for VelocityIndex = 1:NumVelocities

    % Retrieve the selected velocity
    Velocity = Velocities(VelocityIndex);

    % Get a data set of the appropriate velocity
    Data = GetData(valParameters, Velocity, Time);

    % Insert the data into the input sequence
    InsertStart = 1 + (VelocityIndex - 1) * TimeLength;
    InsertEnd = InsertStart + TimeLength - 1;
    ValidationInputSequence (:,InsertStart:InsertEnd) = Data;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Create an output target signal for validation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Clear the target
ValidationTargetSequence = zeros(1,NumVelocities*TimeLength);

% Add a pulse for each velocity that is the same as the matched
% velocity
VelocityIndices = 1:NumVelocities;
for VelocityIndex = VelocityIndices(Velocities == MatchedVelocity)

    % Get the insertion point for the data in the input sequence
    InsertStart = 1 + (VelocityIndex - 1) * TimeLength;
    InsertEnd = InsertStart + TimeLength - 1;

    % Get a data set of the appropriate velocity with no noise
    valParameters.NoiseLevel = 0;
    Data = GetData(valParameters, MatchedVelocity, Time);

    % Get the final channel
    FinalChannel = Data(DataLines,:);

    % Normalise it
    FinalChannel = AgcSim(FinalChannel);

    switch lower(valParameters.TargetOutput)
        case 'pulse'
            % Create a pulse at the 3dB points
            TargetPulse = zeros(size(FinalChannel));
            TargetPulse(FinalChannel >= (max(FinalChannel)/sqrt(2))) =
1;

            % Apply it to the target sequence
            ValidationTargetSequence(InsertStart:InsertEnd) =
TargetPulse;
        case 'ap'
            % Target sequence is action potential with matched velocity
            ValidationTargetSequence(InsertStart:InsertEnd) =
FinalChannel;
        otherwise
            disp('Unknown output training target type.')
    end;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the ANN training
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Normalise the data into the range 1 to -1 for each channel separately
TripolarNorm = AgcSim([TrainingInputSequence, ValidationInputSequence])
;
%TripolarNorm = AgcSim(TrainingInputSequence);

TargetSequence = [TrainingTargetSequence, ValidationTargetSequence];
%TargetSequence = TrainingTargetSequence;

% Assad's ANN training setup
p = con2seq(TripolarNorm);      % Cell array of input vectors
t = con2seq(TargetSequence);    % Cell array of target vectors
```

```
    d = cell(1, numel(Parameters.AnnLength));
    for i = 1:numel(Parameters.AnnLength)
        d{i} = 0:Parameters.AnnLength(i) - 1;     % Delay vector for ith
    layer
    end

    net = distdelaynet(d, Parameters.HiddenLayerSize, Parameters.
    TrainingMethod);
    net.layers{1:end-1}.transferFcn = Parameters.ActivationFcn1;
    net.layers{end}.transferFcn = Parameters.ActivationFcn2;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Levenberg-Marquardt backpropagation default parameters
    % net.trainParam.epochs = 1000;          % Maximum number of epochs to
    train
    % net.trainParam.goal = 0;               % Performance goal
    % net.trainParam.max_fail = 6;           % Maximum validation failures
    % net.trainParam.min_grad = 1e-7;        % Minimum performance gradient
    % net.trainParam.mu = 0.001;             % Initial mu
    net.trainParam.mu_dec = 0.1;            % mu decrease factor
    % net.trainParam.mu_inc = 10;            % mu increase factor
    % net.trainParam.mu_max = 1e10;          % Maximum mu
    % net.trainParam.show = 25;              % Epochs between displays (NaN
    for no displays)
    % net.trainParam.showCommandLine = 0;    % Generate command-line output
    % net.trainParam.showWindow = 1;         % Show training GUI
    % net.trainParam.time = Inf;             % Maximum time to train in
    seconds
    % net.trainParam.mem_reduc = 1;          % Reduce memory and speed to
    calculate the Jacobian jX
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    net.trainParam.max_fail = 5;
    net.divideFcn = 'divideind';
    net.divideParam.trainInd = 1:size(TrainingInputSequence,2);
    net.divideParam.valInd = (1:size(ValidationTargetSequence,2)) + ...
                             size(TrainingInputSequence,2);

    % These are Assad's training criteria so they have been kept
    net.trainParam.epochs = Parameters.MaxIterations;
    net.trainParam.goal = 1e-6;
    %net.divideFcn = 'dividetrain';         % Allocate all data for training
    %net.trainParam.lr = 0.00005;           % trainlm does not have a
    learning rate?

%       net.outputConnect = [1 0];
%       net.biasConnect = [0; 0];
%       net.layers{1}.transferFcn = 'purelin';

    % Do the ANN training
    net = train(net,p,t);

    % Store the ANN in an array
    ANN(AnnIndex).net = net;
end
```

MAIN SCRIPT

```
% DesignANN
%
% C. T. Clarke based on work by Assad al Shueli
% Edited by L F Tiong 06/05/2016
%
% This code configures Matlab and trains a set of ANNs to detect APs at
% particular velocities.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Parameter setup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Basic System parameters
Parameters = struct (                    ...
    'Electrodes',              10      , ...
    'ElectrodeSpacing',       0.5e-3   , ...
    'SamplingFrequency',       1e5     , ...
    'ActionPotentialType',    'tmap2'  , ...
    'StartTestVelocity',        5      , ...
    'StepTestVelocity',         1      , ...
    'EndTestVelocity',          40     , ...
    'MatchRepeats',             20     , ...
    'NoiseLevel',               0      , ...
    'AnnLength',           [120, 1]    , ...
    'HiddenLayerSize',          1      , ...
    'ActivationFcn1',        'tansig'  , ...
    'ActivationFcn2',        'purelin' , ...
    'ActivationFcn3',         'poslin' , ...
    'ActivationFcn4',        'softmax' , ...
    'TrainingMethod',        'trainlm' , ...
    'TargetOutput',           'pulse'  , ...
    'MaxIterations',          100      , ...
    'MinError',                 0      , ...
    'APType',               'BiPolar', ...
    'channels',                 5);

% ANN velocities to train for
%AnnVelocities = [80, 100, 120];
AnnVelocities = [10, 20, 30];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Design each ANN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ANN = DesignAnn(AnnVelocities, Parameters);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Analyse the resultant ANNs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%AnalyseAnn(ANN, AnnVelocities, Parameters);
```

UNIPOLAR GENERATION SOURCE CODE

```
function UniPolarSignal = GetUniPolar(Parameters, ActionPotentialVelocity,
    Time)
% Create a Unipolar action potential
%
% C T Clarke
% Edited by L F Tiong 06/05/2016
%
% Use a parameter structure like this:
% Parameters = struct (                    ...
%   'Electrodes',              11     , ...
%   'ElectrodeSpacing',         0.003, ...
%   'SamplingFrequency',    100000     , ...
%   'ActionPotentialType', 'long'     , ...
%   'StartTestVelocity',        1     , ...
%   'StepTestVelocity',         1     , ...
%   'EndTestVelocity',        100     , ...
%   'NoiseLevel',               0.01 );
% Other parameters may be included but will be ignored

% Establish implied cuff parameters

if (Parameters.Electrodes > 1)
    ElectrodeDelay = Parameters.ElectrodeSpacing ./ ActionPotentialVelocity
    ;
else
    ElectrodeDelay = 0;
end

% Get the length of sequence to produce
SequenceLength = max(size(Time));

A = [2.2e7,0.47e9,2.6e1,4.08e-3,7.44e-11];
B = [3.6e3,  1e4, 1.5e4,  1.5e4,    1e4];
n = [3,        3,     1,      1,      3];
%Use 1 for long, 2 for short, 3 for new

% Use on of the standard formulae for the action potential
switch lower(Parameters.ActionPotentialType)
    case 'long'
        TMAP = 1;
    case 'short'
        TMAP = 2;
    case 'new'
        TMAP = 3;
    case 'tmap1'
        TMAP = 4;
    case 'tmap2'
        TMAP = 5;
    otherwise
        disp('Unknown Action potential type.')
end;

% Standard Unipolar signal (ActionPotentialVelocity^2 * )
UniPolarSignal = zeros(Parameters.Electrodes,SequenceLength);
for i = 1 : Parameters.Electrodes
    UniPolarSignal(i,:) = ActionPotentialVelocity^2 * max(0,A(TMAP)*((Time
    - (i-1) * ElectrodeDelay).^n(TMAP)).*(exp((-B(TMAP))*(Time - (i-1) *
```

```
    ElectrodeDelay))));
end;

% Work out the approximate RMS value of the signal
% max/root(2) is used as this negates the effect of an overly long sequence
% with a short AP.
%SignalRMS = max(max(UniPolarSignal)) / sqrt(2);

% Constant noise level for all AP amplitudes (SignalRMS * )

% Add normally distributed noise
UniPolarSignal = UniPolarSignal + Parameters.NoiseLevel * randn(Parameters.
    Electrodes,SequenceLength);
```

BIPOLAR GENERATION SOURCE CODE

```
function BiPolarSignal = GetBiPolar(Parameters, ActionPotentialVelocity,
    Time)
% Create a BiPolar action potential
%
% A Vlissidis

BiPoles = floor(Parameters.Electrodes/2);
SequenceLength = max(size(Time));

UniPolarSignal = GetUniPolar(Parameters, ActionPotentialVelocity, Time);

idx = 1;
BiPolarSignal = zeros(BiPoles, SequenceLength);
for i = 1:2:Parameters.Electrodes - 1
    BiPolarSignal(idx, :) = UniPolarSignal(i, :) - UniPolarSignal(i+1, :);
    idx = idx + 1;
end

end
```

TIME RESPONSE ANALYSIS SOURCE CODE

```
function time_analysis (channels, ANNindex, opt, rawdata)
% Analyse the time response of the neural network
%
% A Vlissidis

    ANN = evalin('base', 'ANN');
    Parameters = evalin('base', 'Parameters');

    if strcmp(Parameters.APType, 'UniPolar')
        GetData = @GetUniPolar;
        DataLines = Parameters.Electrodes;
    elseif strcmp(Parameters.APType, 'TriPolar')
        GetData = @GetTriPolar;
        DataLines = Parameters.Electrodes - 2;
    elseif strcmp(Parameters.APType, 'BiPolar')
        GetData = @GetBiPolar;
        DataLines = floor(Parameters.Electrodes/2);
    end

    %% Create the input data
    % Set the initial velocity set
    Velocities = Parameters.StartTestVelocity:...
             Parameters.StepTestVelocity:...
             Parameters.EndTestVelocity;
    NumVelocities = numel(Velocities);

    % Set up the time sequence for the signals at each velocity
    InitTime = 0.001;
    EndTime = 0.004;
    Time = -InitTime:1/Parameters.SamplingFrequency:EndTime;
    TimeLength = numel(Time);
    SequenceTime = (0:(TimeLength * NumVelocities) - 1) ...
                   / Parameters.SamplingFrequency;

    for VelocityIndex = 1:NumVelocities

        % Retrieve the selected velocity
        Velocity = Velocities(VelocityIndex);

        % Get a data set of the approriate velocity
        Data = GetData(Parameters, Velocity, Time);

        % Insert the data into the input sequence
        InsertStart = 1 + (VelocityIndex - 1) * TimeLength;
        InsertEnd = InsertStart + TimeLength - 1;
        InputSequence (:,InsertStart:InsertEnd) = Data;
    end

    if strcmp(opt, 'real')
       InputSequence = rawdata(1:7000, 2:end)';
       SequenceTime = rawdata(1:7000, 1)';
    end

    %% Plot each FIR filter output individually
    testnet = ANN(ANNindex).net;                % copy neural network
    testnet.outputConnect = [1  0];             % connect output to first
    layer
```

```
testnet.biasConnect = [0; 0];
testnet.layers{1}.transferFcn = 'purelin';

figure;
hold on
% Pre-allocate for speed
AnnOutput = zeros(DataLines, numel(SequenceTime));
for ChannelIndex = 1:DataLines
    % Apply the AGC
    NormInput = AgcSim(InputSequence);

    % Set every other channel to zero
    NormInput(1:end ~= ChannelIndex, :) = zeros(DataLines-1, numel(
SequenceTime));

    % Convert data format to be input to the ANN from a R-by-TS matrix
    % to a 1-by-TS cell array of R-by-1 vectors
    InputCellArray = con2seq(NormInput);

    % Run the ANN
    AnnOutputCellArray = sim(testnet, InputCellArray);

    % Convert ANN output back to a conventional array
    AnnOutput(ChannelIndex,:) = cell2mat(AnnOutputCellArray);

    % Plot channel time response
    %subplot(DataLines, 1, ChannelIndex);
    plot(SequenceTime, AnnOutput(ChannelIndex, :));
    if ChannelIndex == 1
        title('Individual FIR filter outputs');
    end
end
xlabel('Time (ms)');
hold off
%legend('1', '2', '3')
%% Plot the summed output of the specified channels
figure;
% Sum the specified channels
out = sum(AnnOutput(channels, :), 1);

NormInput = AgcSim(InputSequence);
InputCellArray = con2seq(NormInput);

AnnOutputCellArray = sim(testnet, InputCellArray);
AnnOutput = cell2mat(AnnOutputCellArray);
sum(AnnOutput - out)

plot(SequenceTime, out);
xlabel('Time (ms)');
title(strcat('Summed output from FIR Channels'));

%% Plot the input
if strcmp(opt, 'real')
    figure;
    for i = 2:6
        subplot(5, 1, i-1);
        plot(rawdata(1:7000, 1),rawdata(1:7000, i));
    end
    xlabel('Time (s)');
```

```
    else
        figure;
        input = cell2mat(InputCellArray);
        plot(SequenceTime, input(DataLines, :));
        xlabel('Time (ms)');
        title('Input Sequence');
    end

    %% Plot summed output with bias
    testnet = ANN(ANNindex).net;
    testnet.outputConnect = [1 0];
    testnet.layers{1}.TransferFcn = 'purelin';

    Norm = AgcSim(InputSequence);
    InputCellArray = con2seq(Norm);
    AnnOutputCellArray = sim(testnet, InputCellArray);
    AnnOutput = cell2mat(AnnOutputCellArray);
    figure;
    plot(SequenceTime, AnnOutput);
    title('Summed output with bias');
    xlabel('Time (ms)');

    %% Plot summed output with bias and sigmoid function
    testnet = ANN(ANNindex).net;
    testnet.outputConnect = [1 0];
    testnet.layers{1}.transferFcn = 'tansig';

    AnnOutputCellArray = sim(testnet, InputCellArray);
    AnnOutput = cell2mat(AnnOutputCellArray);
    figure;
    plot(SequenceTime, AnnOutput);
    title('Summed output with bias and sigmoid');
    xlabel('Time (ms)');

    %% Revert back to normal and plot global output
    testnet = ANN(ANNindex).net;
    testnet.outputConnect = [0 1];
    testnet.layers{1}.transferFcn = 'tansig';

    AnnOutputCellArray = sim(testnet, InputCellArray);
    AnnOutput = cell2mat(AnnOutputCellArray);
    figure;
    plot(SequenceTime, AnnOutput);
    title('Global Output');
    xlabel('Time (ms)');
end
```