Fall 2011

NeuralSTA: a software tool for neural stimulation and recording applications with a laser control

Rimi Sahu
New Jersey Institute of Technology

ABSTRACT

NeuralSTA: A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING APPLICATIONS WITH A LASER CONTROL

by
Rimi Sahu

The subject of this thesis is a software tool created in Matlab ® for neural stimulation and recording of the evoked potentials. The stimulation output can be delivered in electrical units for immediate use for neural stimulation or as a power control signal to a laser. NeuralSTA is designed to generate arbitrary waveforms of stimulation and simultaneously record the neural response while implementing the spike triggered averaging (STA) method, hence the name NeuralSTA. The output (stimulation) signal can be created in monophasic, biphasic or inverted monophasic forms (for laser control applications). Controls for pulse modulation allow shaping of the pulse waveform, and the controls for the signal allow for modulation of the entire stimulation pattern. In addition to generating arbitrary waveforms, NeuralSTA can also be programmed for repetitive generation of the output stimulation pattern as well as a stepwise increase of the amplitude during each repetition.

NeuralSTA was tested in several animal experiments during a year, and continuously updated to meet the specific requirements that were noticed during each testing. As a result, many options are provided to the user in order to meet typical requirements in most common applications of neural stimulation and recording.

NeuralSTA: A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING APPLICATIONS WITH A LASER CONTROL

by
Rimi Sahu

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Biomedical Engineering

Department of Biomedical Engineering

January 2012

Blank Page

APPROVAL PAGE

NeuralSTA: A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING APPLICATIONS WITH A LASER CONTROL

Rimi Sahu

Dr. Mesut Sahin, Thesis Advisor Date
Associate Professor of Biomedical Engineering, NJIT

Dr. Sergei Adamovich, Committee Member Date
Associate Professor of Biomedical Engineering, NJIT

Dr. Joel Schesser, Committee Member Date
Senior University lecturer of Biomedical Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: **Rimi Sahu**

Degree: Master of Science

Date: January 2012

Undergraduate and Graduate Education:

- Master of Science in Biomedical Engineering,
  New Jersey Institute of Technology, Newark, N.J USA, 2012

- Bachelor of Science in Biomedical Engineering,
  New Jersey Institute of Technology, Newark, N.J USA, 2009

Major: Biomedical Engineering

Presentations and Publications:

Sahu R. "Fabrication of macroporous silicon for biomedical applications." New Jersey Institute of Technology McNair Symposium. Newark, New Jersey 2009

To my mother

ACKNOWLEDGEMENT

My utmost gratitude goes to Dr. Sahin for allowing me to work with him. I am indebted to him for his guidance and understanding and most of all his encouragement. This would not be possible without him. I would also like to acknowledge Ammar Abdo for suggesting ideas for improving the program.

My sincerest thanks and appreciation to my thesis committee members, Dr. Adamovich and Dr. Schesser for their invaluable support and guidance

I would like to thank the Mathworks community for their useful tutorials and timely response.

I would also like to thank mother for being my strength and my dad for his protection and my sister for her encouragement and smile. Last, my gratitude to the lord for his support.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

NeuralSTA allows the user to generate signals to stimulate neural tissue, and to simultaneously record and study the evoked response. The stimulation signal can be formed in biphasic, monophasic and inverted-monophasic (for laser applications) modes.

The significance of using biphasic signals in neural stimulations is tremendous and widely acknowledged in research studies and clinical applications such as deep brain stimulation [1]. One of the main advantages of using biphasic signals to collect evoked response is that this causes minimal tissue damage to the surrounding tissue and also the electrode [1], [2] and there by optimizing the effect of the stimulation and consequently the evoked potential [3]. The purpose for having a positive phase in the signal is for reversing the chemical species that are generated during faradic reaction. In applications for research and clinical applications, the biphasic signals are in a rectangular form. NeuralSTA allows the user to generate non-rectangular biphasic signals that can be used for future research and applications.

As opposed to biphasic signals, monophasic signals have been noted in several studies for causing extensive tissue damage. There are several papers that state conflicting results as to the role of monophasic signals [4]. NeuralSTA can find application in studies where artificial tissue damage needs to be induced or studied.

NeuralSTA allows the user to create an invited monophasic signal to control analog input lasers (DLS-500-830FS-100). This laser wirelessly controls a floating light-activated micro-electrical stimulator (FLAMES) for neural stimulation.

The unique aspect of this project that differentiates it from other currently available software is the ability to produce non-rectangular stimulation waveforms and to modulate the stimulus train with a second math function over the entire stimulus pattern. It is our belief that this will be a very useful feature for neural applications where there is increasing interest in investigating alternative stimulation waveforms that may require less energy from the implanted battery.

1.1. Functional requirements

In implementation of NeuralSTA, it was set out to achieve several features that would be required from a neural stimulation software of this kind. The initial requirements from NeuralSTA were as follows:

i. Option for simultaneous stimulation pattern generation and recording of evoked potentials.
ii. Option to generate signals in various stimulation modes (biphasic, monophasic and laser mode) based on user specified parameters, for e.g. amplitude, frequency, pulse width, duration, sampling rate, modulation etc.
iii. Option to use different pulse shapes, i.e. rectangular, exponential, linear, Gaussian or sinusoidal waveform. In addition to the pulse shaping, provide option to modulate the entire stimulus waveform with a rectangular sine, cosine, square or sawtooth function.
iv. Option for plotting the recorded signal after each frame while the stimulation pattern is still being executed.
v. Option to save a file with the parameters used during each stimulation trial and be able to reload the file at a later time.

vi. Option to check for any errors/inconsistencies with the specified parameters by providing pop-up menus describing the parameters and summarizing inconsistencies in the parameters entered by the user.

vii. Option to allow access to the parameters of the analog input and output of the data acquisition (DAQ) card, e.g. sampling rate, channel ID, voltage range and channel configuration.

CHAPTER 2

BACKGROUND

In Chapter 2, information regarding the versions of Matlab ® in which the program was tested and more detailed information about DAQ cards is included. In addition to this, there are also sections that explain the formulas and terms used to implement the software.

2.1. Information about Matlab ® and data acquisition card requirements

NeuralSTA uses the Matlab® data acquisition toolbox to implement the desired functions. This program has been tested in the 32-bit Matlab® 2008-2011 (a & b) versions. This program may or may not work in Matlab® versions earlier than 2008 and later than 2011.

This program is designed to support a variety of DAQ hardware including USB, PCI, PCI-Express®, PXI, and PXI-Express devices, from National Instruments [5]. However, it must also be noted that the National Instruments legacy interface [6] (PCI and PXI serial interfaces for Windows etc) cannot be used on 64–bit Matlab® [7]. For this reason a 32-bit Matlab® Version was used to design and use NeuralSTA. The 32-bit Matlab® version is compatible with a 64-bit, windows 7 operating system.

In addition to this, it should also be noted that NeuralSTA is supported by all National Instruments data acquisition hardware except the NI CompactDAQ devices [8] and Devices using the counter/timer subsystem [7].

2.2. Advanced DAQ boards and analog interfaces

Most data acquisition hardware devices contain subsystems such as Analog Input, Analog/Digital (A/D) converters, etc. to convert real world analog data into digital data that can be collected and stored for future analysis/use.

The analog input subsystem, samples (takes a "snapshot" of input signal) and quantizes (divides the input voltage/current into discrete amplitudes) the input analog signal from one or multiple channels [7].

2.3. Multiplexing de-multiplexing of analog input signals

In most digital converters, the sampling function is implemented by a sample and hold (S/H) circuit (figure 2.1). The S/H circuit contains a signal buffer which is followed by an electronic switch that is connected to a capacitor [8]. For multiple channel recordings, the channels are multiplexed to one A/D converter, which then performs its functions in the same way as it would for a single channel [8]. The following is a diagram of a data acquisition device that has one A/D converter and is multiplexed to multiple input channels:

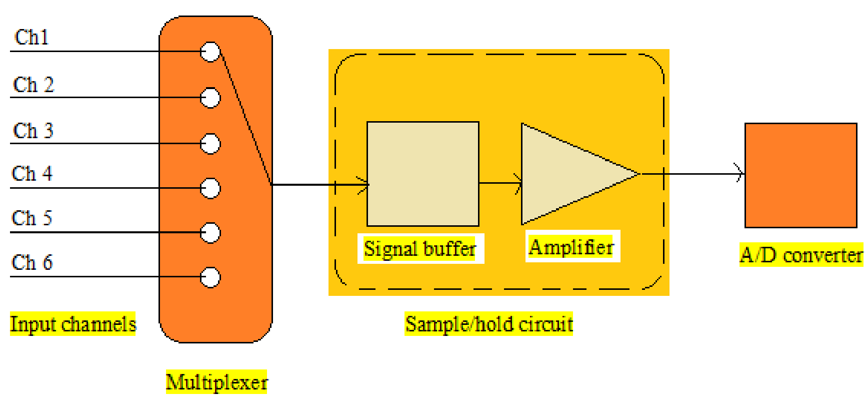


Figure 2.1 Data Acquisition card with A/D converter that multiplexes multiple channels.

Source: Adapted from the Mathworks ® website [9]

2.4. Scanning method and limitations

NeuralSTA was developed using National Instruments' Data Acquisition (DAQ) Card. The bit resolution of the DAQ device is one of the factors that can affect resolution and accuracy of the recorded evoked potential [8]. The following are some of the important factors that are pertinent to improving the data collected with NeuralSTA.

2.4.1. Channel skew

In case of a single channel recording, the multichannel DAQ card samples the channel sequentially. The following diagram shows three values of data collected by a channel. Each data value is recorded during one sampling period. As it can be noticed, each data value is recorded after a constant time interval.

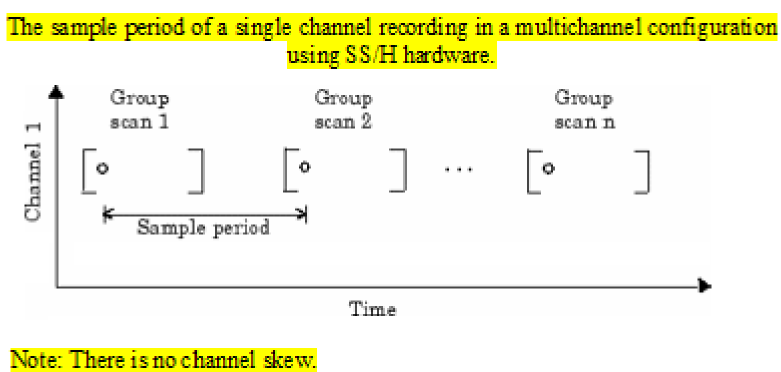


Figure 2.2 Sample period of a single channel configuration with no channel skew

Source: Adapted from the Mathworks ® website [9]

In multiple channel recordings, which is a feature used in NeuralSTA, the data in the channels are not simultaneously sampled. In the following figure, data is collected from multiple channels. During one sampling period, rather than collecting data from the channels at the same time, the data are collected at different times. This discrepancy is caused by the channel skew, which is the time gap between consecutive sampled channels in one sample period [9].

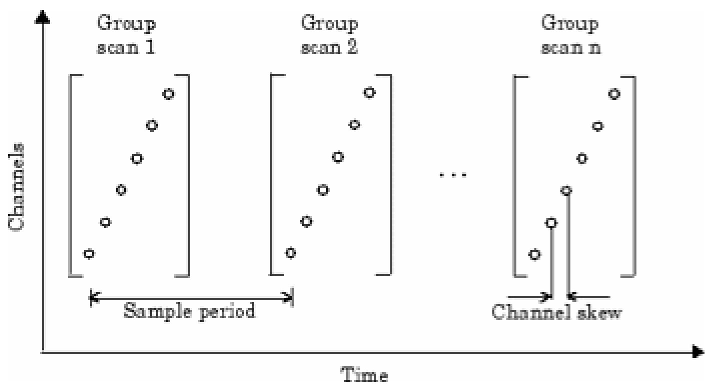


Figure 2.3 Sample period of multi-channel configuration with channel skew

Source: Retrieved from the Mathworks ® website [9]

This channel skew can be resolved by using a simultaneous sample and hold (SS/H) hardware [9]. The following is an example of sample period for a multichannel configuration using SS/H hardware. In this case the data from all the channels are sampled at the same time.

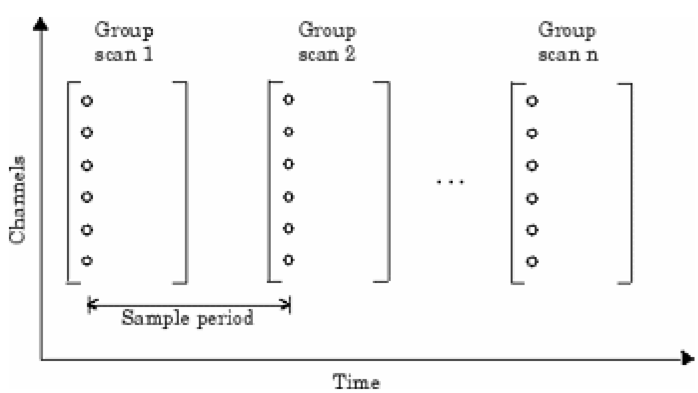


Figure 2.4 Sample period of a multichannel configuration (SS/H hardware) without channel skew

Source: Retrieved from the Mathworks ® website [9]

In case the user does not have access to hardware with such a setting (simultaneous sample and hold hardware), the NeuralSTA minimizes the channel skew by setting the analog input channel skew mode to “equisample” by giving the Matlab® command: “ ai.ChannelskewMode = 'Equisample'; ”

This sets the channel skew, i.e. the time interval for sampling one channel to a known value according to the following formula:

$$Channel\ Skew = \frac{1}{Sampling\ rate * Number\ of\ channels} \qquad (2.1)$$

NeuralSTA assumes that the user does not have SS/H hardware and has the Channel skew mode set in the ‘equisample’ mode.

2.4.2. Sampling rate

In order to collect data/response, the DAQ card takes a snapshot of the input signal (ie, evoked potential) at discrete times. This time value is the sampling rate which usually has the unit of samples per second. This value must be greater than twice the maximum frequency of the input signal [7].

For multiple channel recording, there is a hardware limitation for the sampling rate that can be used. The user can only use the maximum sampling rate (specified in the DAQ card) for single channel recordings. But this maximum value reduces as the number of channels used for recording increases. The following formula gives the maximum sampling rate that can be used during a multiple channel recording:

$$Maximum\ sampling\ rate\ per\ channel\ = \frac{Maximum\ board\ rate}{Number\ of\ channels} \tag{2.2}$$

2.4.3. Quantization

As the data is being sampled, the DAQ card converts the voltage value of the signal to a binary number that can be stored by the computer [10]. This conversion includes quantization. Some of the factors that determine the resolution of the recorded signal are the DAQ card's input voltage range and the number of bits used for conversion (word length). The least significant bit (lsb) is represented by the following formula:

$$\mathbf{lsb\ value} = \frac{\mathbf{full\ voltage\ range}}{\mathbf{2^{word\ length}}} \tag{2.3}$$

During the testing phase of NeuralSTA, the collected data were accurate according to the calculated lsb value.

2.4.4. Input polarity

During the programming phase, it was noted that the DAQ card's default settings were from the 0 to the maximum positive voltage value. It is essential that during stimulations (especially biphasic) the voltage range (polarity) of the expected input signal is specified. The following is an example of unipolar and bipolar voltage ranges:

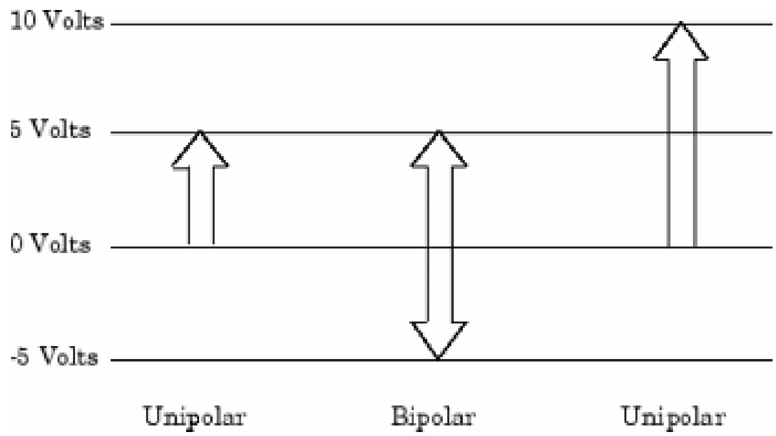


Figure 2.5 Input range polarities

Source: Retrieved from Mathworks website [10]

2.5. Methods of data transfer to computer memory

The acquired data is transferred from the hardware to the memory through the first-in-first-out buffer interrupts.

2.5.1. The hardware FIFO (first-in first-out) buffer stores the acquired data

The FIFO buffer is used to temporarily store the acquired data until it can be transferred to the system memory.

2.5.2. Using interrupts, the data is transferred to the DMA (direct memory access)

The slowest but most common method to move acquired data to system memory is for the board to generate an interrupt request (IRQ) signal. This signal can be generated whether one sample or multiple samples are acquired [11].

The actual data relocation is fairly quick, but there is a large overhead time spent saving, setting up, and restoring the register information. Therefore, depending on the specific system, transferring data by interrupts may not be a good choice when the sampling rate is greater than around 5 kHz [11].

Direct memory access (DMA) is a system whereby samples are automatically stored in the system memory while the central processor is occupied with other tasks [11].

CHAPTER 3

STIMULATION SIGNAL PATTERN & RECORDING

Chapter 3 contains detailed descriptions the basic methodology of signal creation (in three levels of modulation) and recording of the evoked potentials. It also contains descriptions of the process for setting the parameters and achieving signals with various modulations.

3.1. Software design method

NeuralSTA is designed to allow the user to create a stimulation signal and output it through a DAQ card which is connected to an appropriate transducer that is close to the neural tisse. In response to the signal received by the tissue, and evoked response is generated which is then collected by another transducer connected to the analog input channels of the DAQ card. This data from the analog input channels is received and saved on the computer by NeuralSTA. Figure 3.1 is an overview of the process of stimulation and recording.

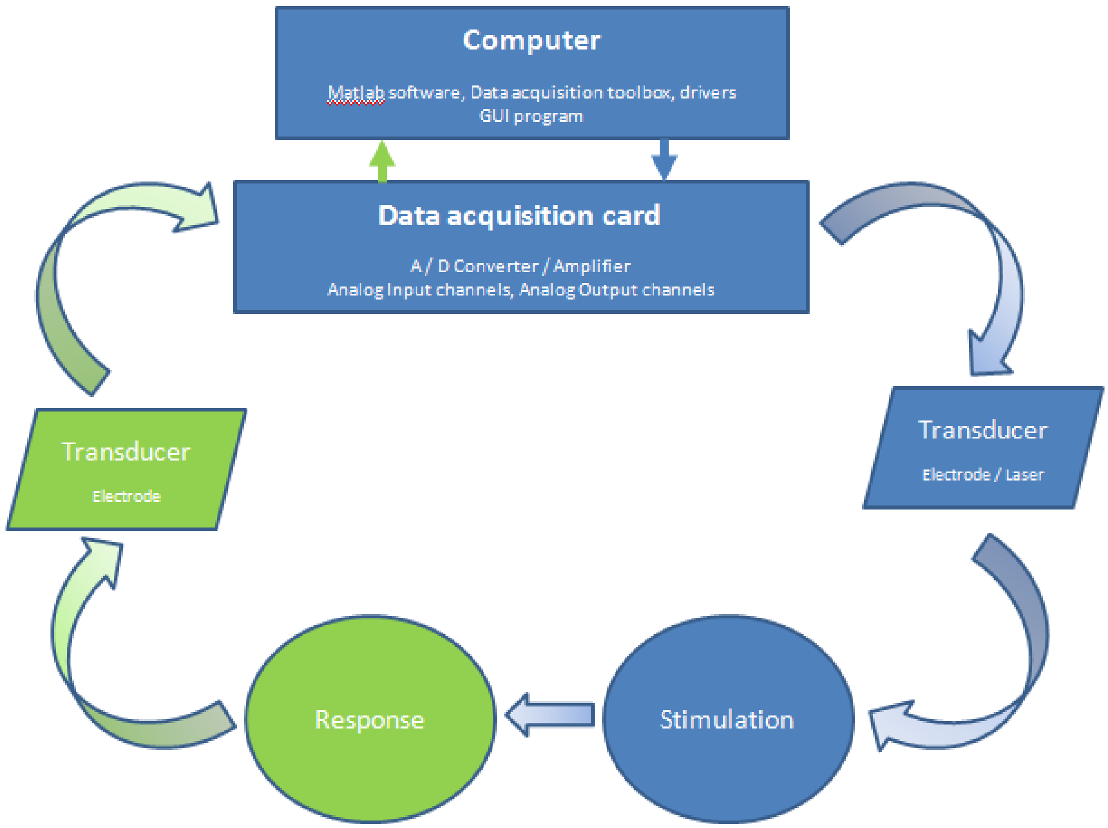


Figure 3.1 Flowchart of stimulation and recording with NeuralSTA

For creating a stimulation signal, one of the three modes can be chosen. The first mode is biphasic with a positive and a negative phase, the second is monophasic with a positive phase and third is for controlling a laser (DLS-500-830FS-100). The signal can be further designed by specifying parameters such as amplitude, pulse width, frequency, duration, sampling rate, etc.

After creating a stimulation signal the analog output and input channels of the DAQ card can be set to stimulate and record (respectively) the appropriate signals. At the end of stimulation and recording frame, recorded data can be displayed individually or as an average of all collected frames. Following completion of the stimulation and recording, two ".mat" files containing the collected data and parameters used during stimulation are automatically saved for future use/analysis. NeuralSTA is equipped with 'help' push-buttons next to the parameter boxes. These push-buttons give a short description of the parameters. An "Error Check" push-button gives a summary of conflicting parameters before stimulation allowing the user to make the appropriate adjustments. Last, the "Plot Initial Signal" button allows the user to check the initial signal before it is used for stimulation.

The source code is also made available as an appendix (appendix E) to this thesis. Most lines in the code have comments explaining their function, thus making the modifications easy for the user.

3.2. Stimulation / output signal organization

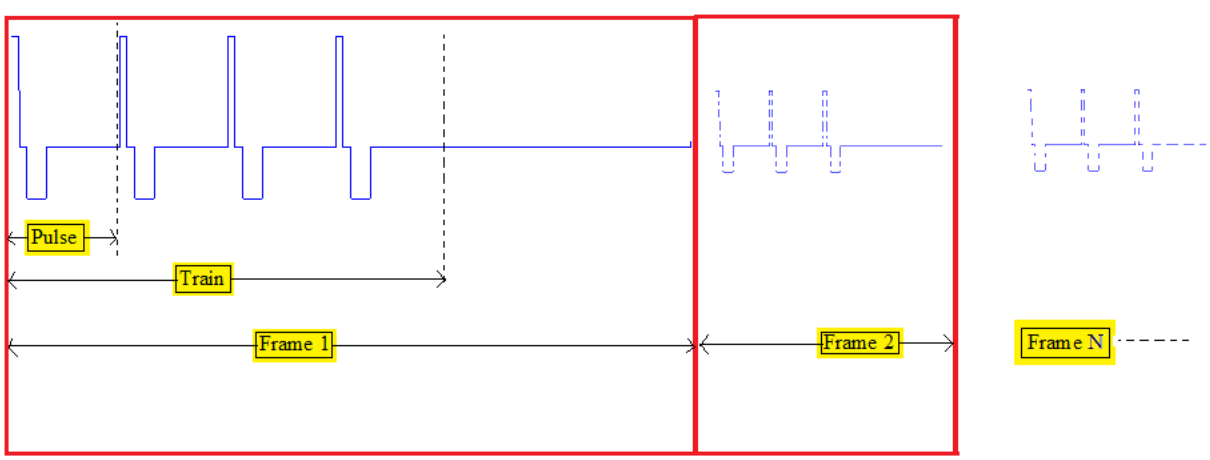


Figure 3.2 Breakdown view of frame, train and pulse signals

The Figure 3.2 shows the three distinct parts of the stimulation signal. The first part is the pulse waveform of the signal. The figure displays a biphasic pulse, which can also be monophasic or inverted monophasic for laser control. The next part is the "train."

A train is a collection of pulses, and its length is determined by the time duration specified by the user. The next part is the “frame.” The length of this part is also determined by the user. When the frame length exceeds the train length, the difference between the frame time and the train time constitutes the no stimulation or zero volt phase of the frame. After building the frame, the user has the option to repetitively generate the frame multiple times during one stimulation period.

In addition to creating rectangular pulses and trains, the user also has the option to modulate the pulse into non rectangular waveforms such as exponential, linear, Gaussian or sinusoidal, and to modulate the train into rectangular sine, cosine, square or sawtooth functions. There is also the option to modulate the train and the frame at the same time.

3.3. Pulse signal modes

The following are basic parts of a rectangular pulse in the biphasic, monophasic and laser modes:

<u>Biphasic Mode:</u>

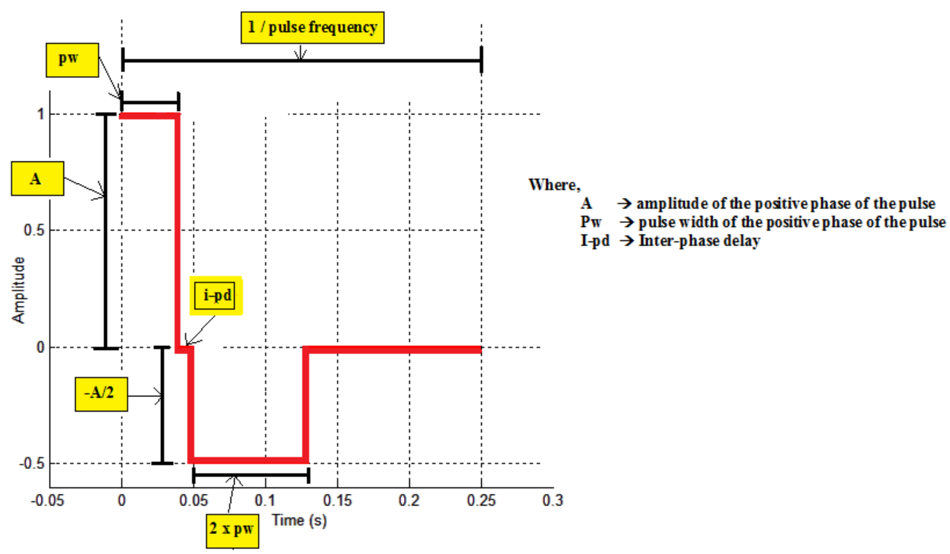


Figure 3.3 Detailed view of a biphasic pulse

In Figure 3.3, the biphasic pulse is displayed in red. The first part of the pulse is the positive phase. The duration (time) of this part is referred to as the pulse-width (pw) in NeuralSTA. The amplitude (A) and duration (pw) of this part is specified by the user in addition to the pulse frequency. The next phase is the inter-phase delay between the positive and the negative phases. The user has the option to determine the duration of this part and can also remove this delay by specifying zero milliseconds for its duration.

The next part is the negative phase. The duration of this section is automatically calculated from the amplitude and duration values entered by the user for the postitive phase. The following is the calculation that is used to define the pulse:

- Positive amplitude (in volt): A
- Positive phase pulse width (in milliseconds): pw
- Inter-phase delay (in milliseconds): i-pd
- Negative amplitude (in volt): $\frac{-A}{2}$
- Negative pulse width (in milliseconds): 2 x pw

The duration of the last zero volt segment seen in the Figure 3.3 (between 0.13s to 0.25s) is calculated by subtracting from the sum of the first three sections of the pulse (positive pulse width, inter-phase width and negative pulse width).

Monophasic Mode:

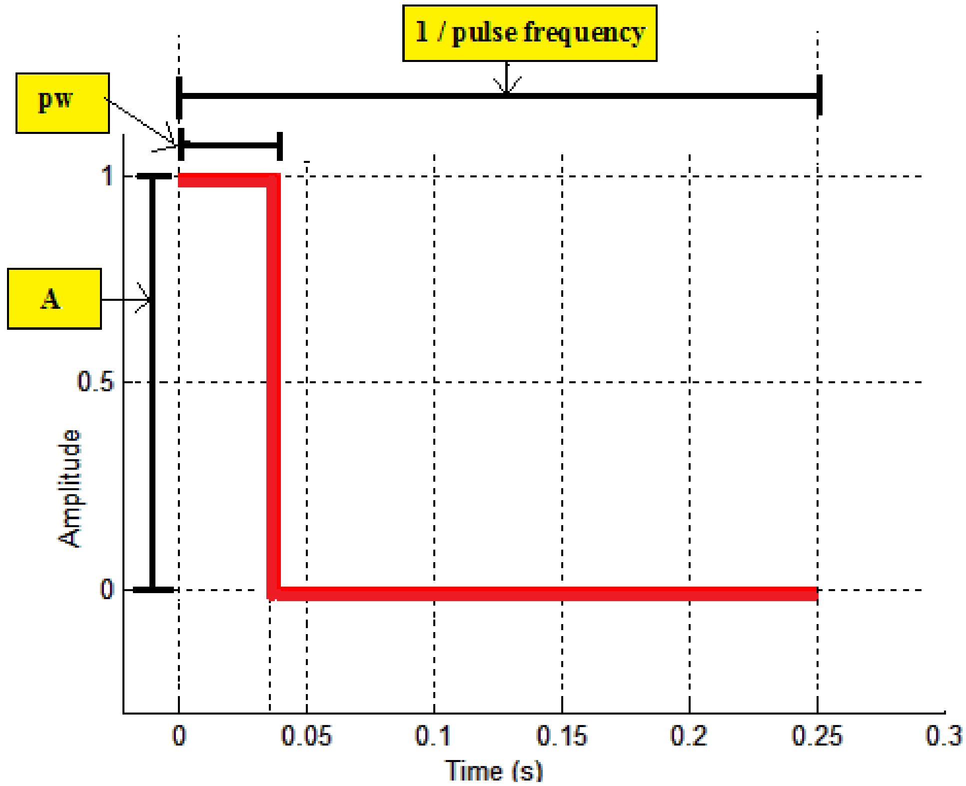


Figure 3.4 Detailed view of a monophasic pulse

The monophasic pulse allows the user to create positive pulses by specifying the amplitude, duration and frequency of the pulse. The first segment is the positive phase. The duration (time) of this interval is referred to as the pulse width (pw) in NeuralSTA. The amplitude (A) and pulse-width (pw) along with the pulse frequency is specified by the user.

The next interval is a silence for the rest of the monophasic signal (0V). The duration of this section is automatically calculated from the difference of the pulse period (1/pulse frequency) and the pulse width.

Laser mode:

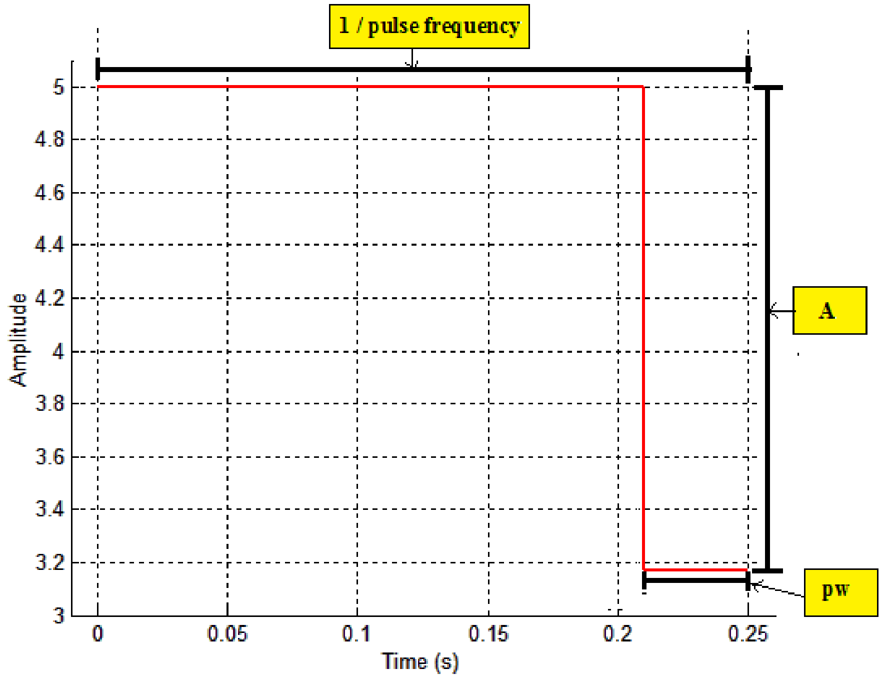


Figure 3.5 Detailed view of a laser pulse

The laser pulse is specifically designed for a laser with an analog input control (DLS-500-830FS-100). The input voltage is inversely proportional with the laser power. This pulse is created by the user by specifying the amplitude and pulse width and the frequency. Like the biphasic and monophasic pulses, this pulse has two distinctive parts. The amplitude of the last part of the pulse is specified by the user as well as its duration. The amplitude is given in terms of percent power. This value is then converted to an equivalent voltage value using the formula:

$$b = (113.4 - a) / 25.39 \tag{3.1}$$

Where 'a' is the percent power given by the user and 'b' is the control voltage. This equation was decided based upon experimental testing of a specific laser unit.

3.4. Train signal

For creating a "train", the user first gives parameters for pulse and then sets the parameters for the train. A train consists of multiple pulses that were described in section 3.2. The train depends on the pulse frequency and the train duration selected by the user. The following are the plots of trains in various modes:

<u>Biphasic Mode:</u>

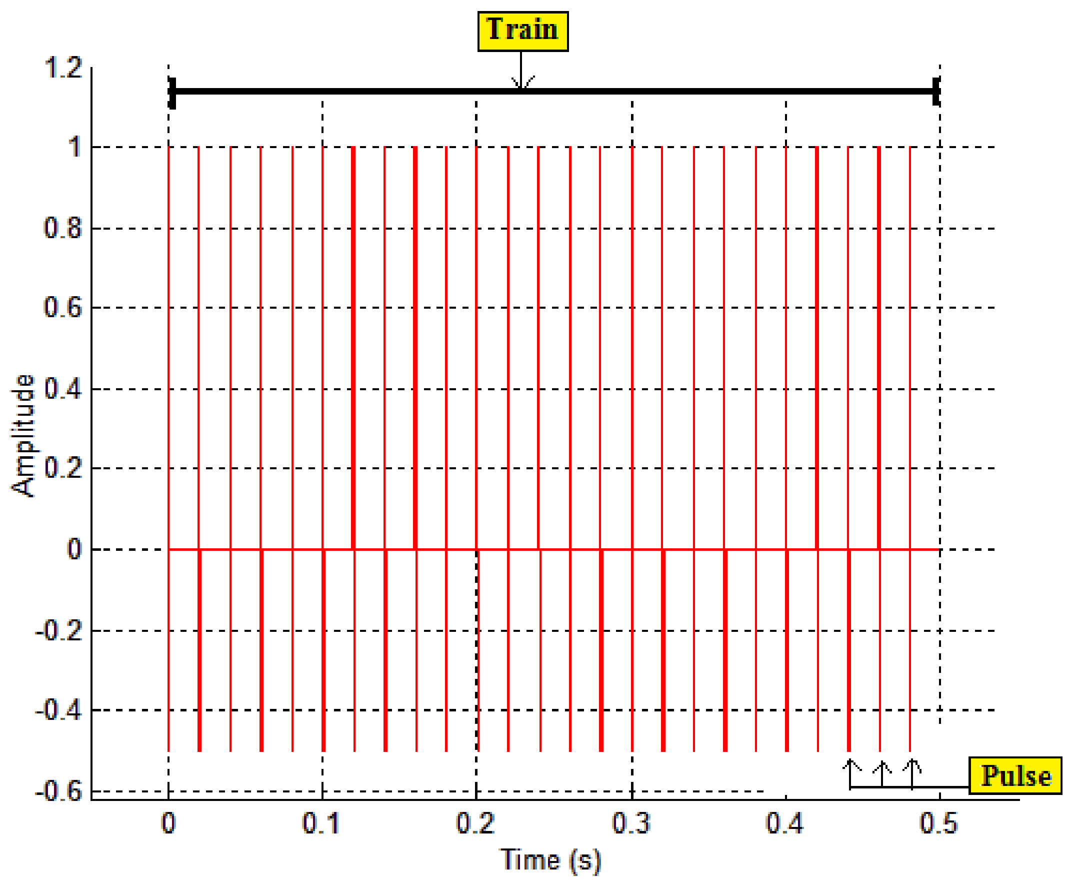


Figure 3.6 Detailed view of a biphasic signal train

Monophasic Mode:

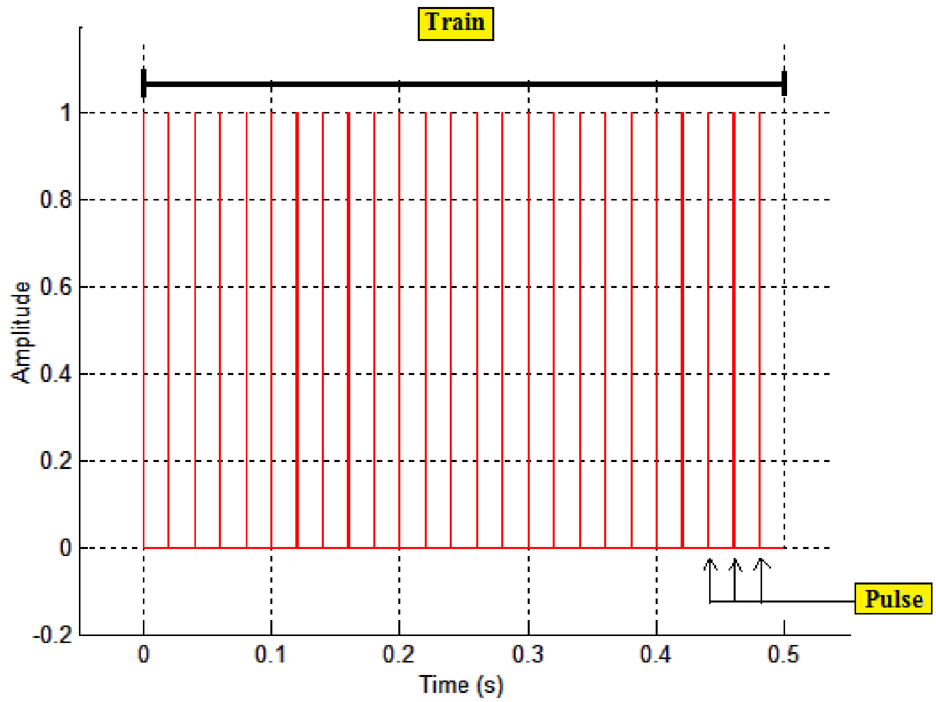


Figure 3.7 Detailed view of a monophasic signal train

Laser Mode:

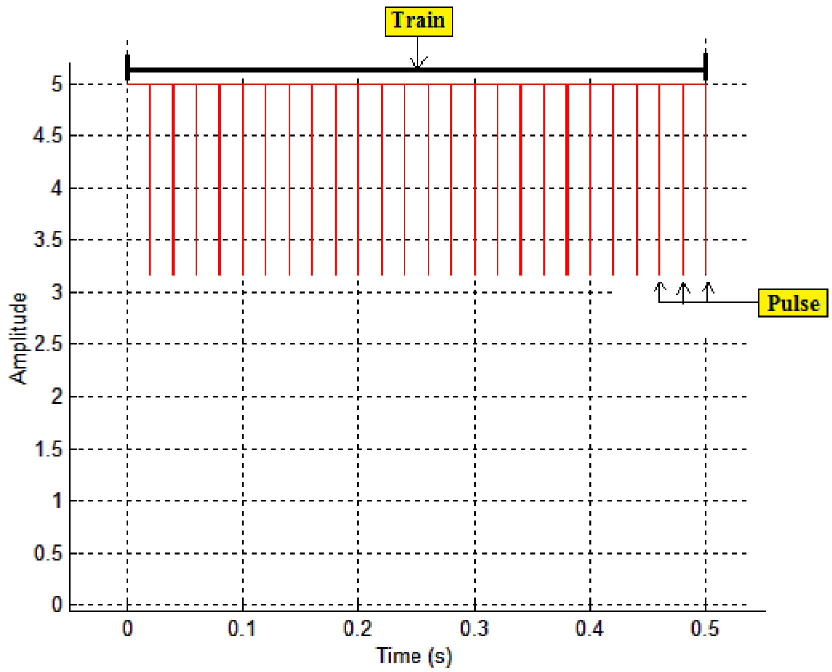


Figure 3.8 Detailed view of a laser mode signal train

3.5. Frame signal

A “frame” is created when there is a need to have a stimulus free (0V) interval after the duration of the ‘train.’ The duration of the stimulus free region is determined by the difference between the ‘frame’ duration (larger value) and the ‘train’ duration. If this 0V region is not required, then the value of the frame duration must be set to the same value as the train duration. The following is an example of the stimulus free period in a frame.

<u>Biphasic Mode:</u>

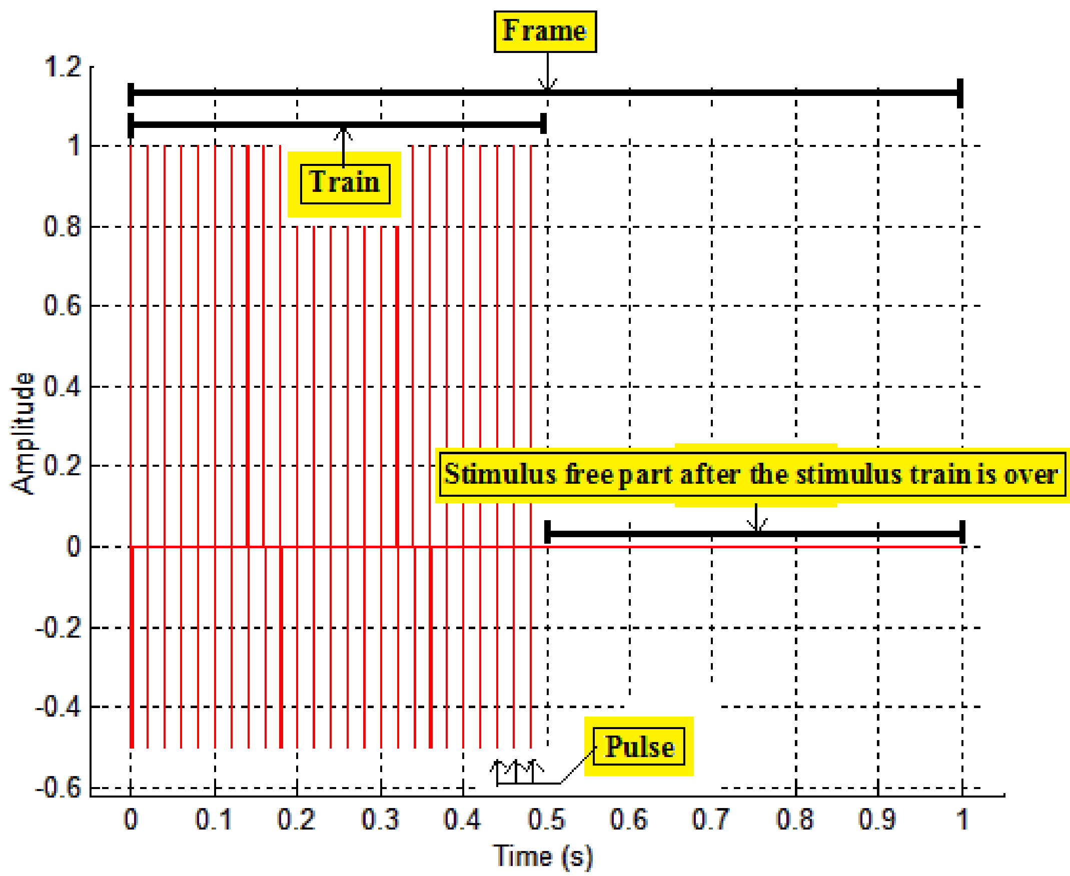
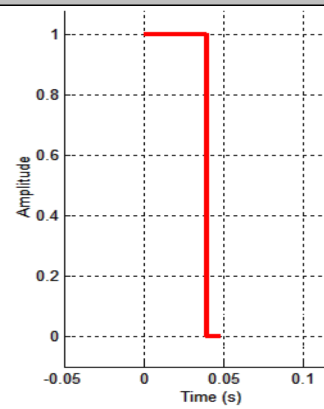
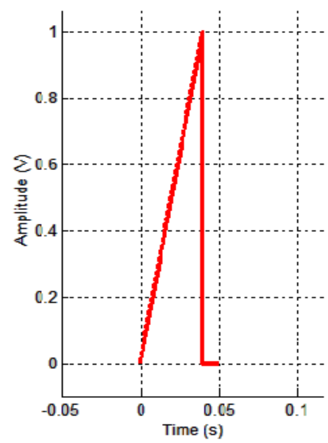


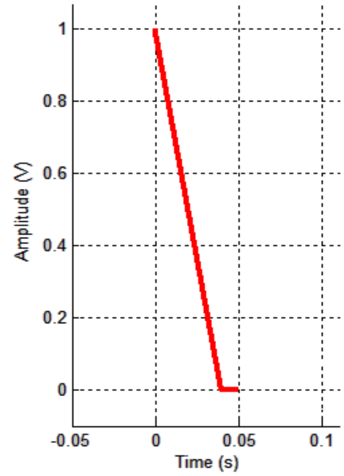
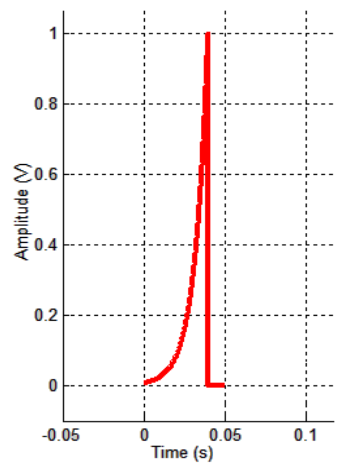
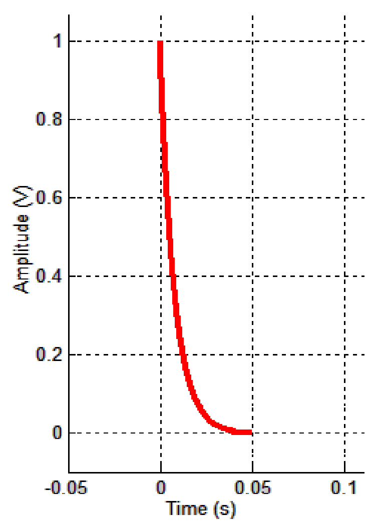
Figure 3.9 Detailed view of a biphasic signal frame

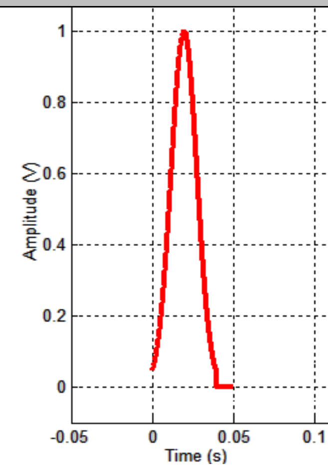
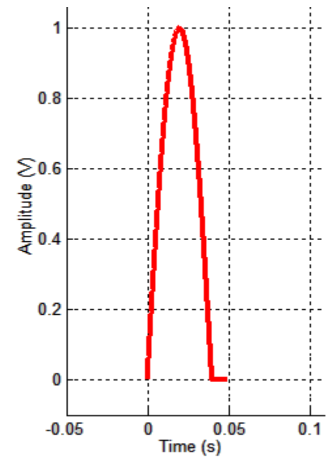
3.6. Pulse modulation

As an alternative to normal rectangular waveform, NeuralSTA provides the user with the following optional pulse waveforms presented in Table 3.1. All examples included in the table are in the monophasic mode.

Table 3.1 Waveforms for pulse modulation

	Waveform function	Formula[12]	Example graph*
1	**Normal (rectangular)**	$K\,[\,u\,(t) - u\,(t-\tau)\,]$	
2	**Linear Increase**	Kt	

	Waveform function	Formula[12]	Example graph*
3	**Linear Decrease**	$K(\tau - t)$	
4	**Exponential Increase**	$K\ (e^{-\frac{5\ (\tau - t)}{\tau}})$	
5	**Exponential Decrease**	$K(e^{-\frac{5\ t}{\tau}})$	

	Waveform function	Formula[12]	Example graph*
6	**Gaussian**	$\frac{K}{(\tau/5)\sqrt{2\pi}}\, e^{-\left(\frac{t-\frac{\tau}{2}}{\sqrt{2}\frac{\tau}{5}}\right)}$	
7	**Sinusoidal**	$K\ \sin(\frac{\pi t}{\tau})$	

Source: [12]

Where,

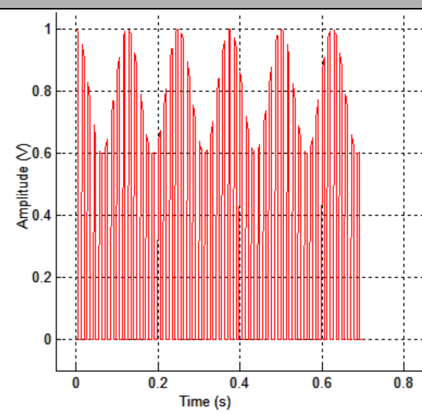
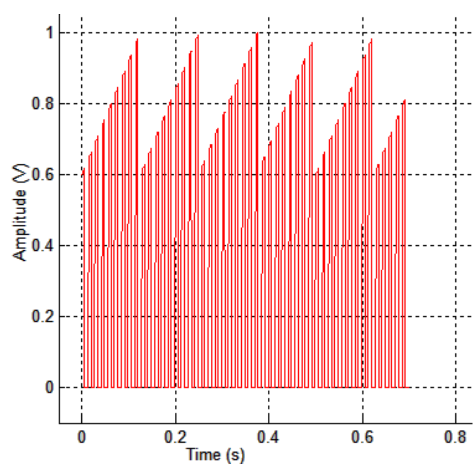
- 0 < t < τ
- τ (pulse - width) is specified by the user
- K is the stimulus strength factor. (NeuralSTA uses the product of the pulse amplitude and K-value for the total stimulus strength. A value of one is given for the "K" value for disabling it)
- u(t) is the unit step function

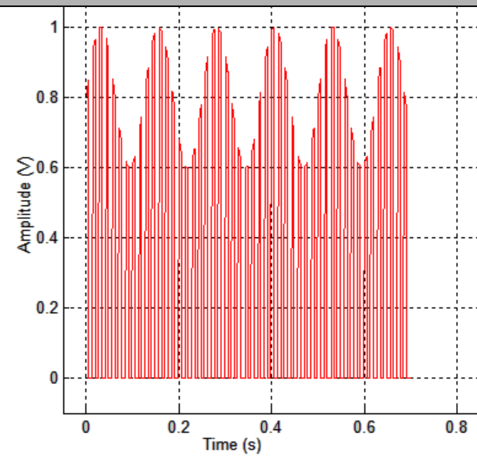
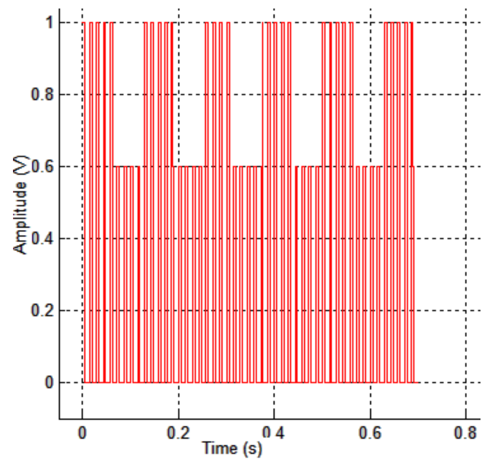
* Time (seconds) vs. Amplitude (volt)

3.7. Signal modulation

NeuralSTA allows modulation of the train signal by four different waveforms as shown in table 3.2. The user selectable parameters are the waveform frequency, amplitude (normal depth) and offset. Additionally, table 3.2 also lists examples of trains in the biphasic, monophasic and laser mode with different signal/train modulations.

Table 3.2 Waveforms for signal modulation

	Waveform function	Formula	Example plot*
1	Cosine	$A.\cos(2\pi ft) + B$	
2	Sawtooth / ramp	$\begin{cases} t \; for \; t \geq 0 \\ 0 \; for \; t < 0 \end{cases}$	

	Waveform function	Formula	Example plot*
3	**Sine**	$A.\sin(2\pi f t) + B$	
4	**Rectangular**		

Source: [13], [14]

Where the specific values are:

- A: amplitude (1V)
- B: vertical offset (0.8V)
- f: frequency (4Hz)
- t: train duration (.7s)

* Time (seconds) vs. Amplitude (volt)

3.8. Frame by frame modulation

The frame by frame modulation allows the user to vary the amplitude of the stimulation pattern in consecutive frames during a trial. To achieve this, the user specifies the minimum amplitude (which is the pulse amplitude x K-value), the maximum amplitude, and the number of steps (n) in which the amplitude would change from the minimum to maximum. Using this value the program generates ‘n’ values of amplitude starting from the minimum amplitude and ending at the maximum amplitude.

In the example below, the following parameters were given to achieve frame by frame amplitude modulation:

- Minimum amplitude (pulse amplitude x K-value): 1V
- Maximum amplitude: 5V
- Step size: 5
- # of frames: 5

With these parameters, the following frame amplitudes can be achieved during a single stimulation of five consecutive frames:

- Frame 1: 1V
- Frame 2: 2V
- Frame 3: 3V
- Frame 4: 4V
- Frame 5: 5V

Examples with different parameters for the minimum and maximum amplitudes are presented in Appendix A. Variations in the step size and # of frames can also give a unique combination of frame amplitudes. Examples of this are presented in Appendix B.

3.9. Analog input (AI) – analog output (AO) delay

NeuralSTA allows for a delay between recording the evoked response (AI) and sending the stimulation signal (AO). The purpose for the AI – AO delay is to have a baseline recording before the stimulation pattern starts. The data recorded before and during the stimulation can be used to evaluate the effect of the stimulations by comparing the evoked neural response with that of the baseline level. The delay time is given in terms of seconds.

CHAPTER 4

SOFTWARE DEVELOPMENT

In this Chapter, a detailed guide to accessing all the features of NeuralSTA is presented. First, individual (modulation, save/load, recording parameters, etc.) sections in NeuralSTA are described along with tips on how to get access to the help feature of the program. Next, a detailed description of how to load the parameters from a default file and check the parameters entered for errors is given. Next, additional features of NeuralSTA during and after stimulation are summarized.

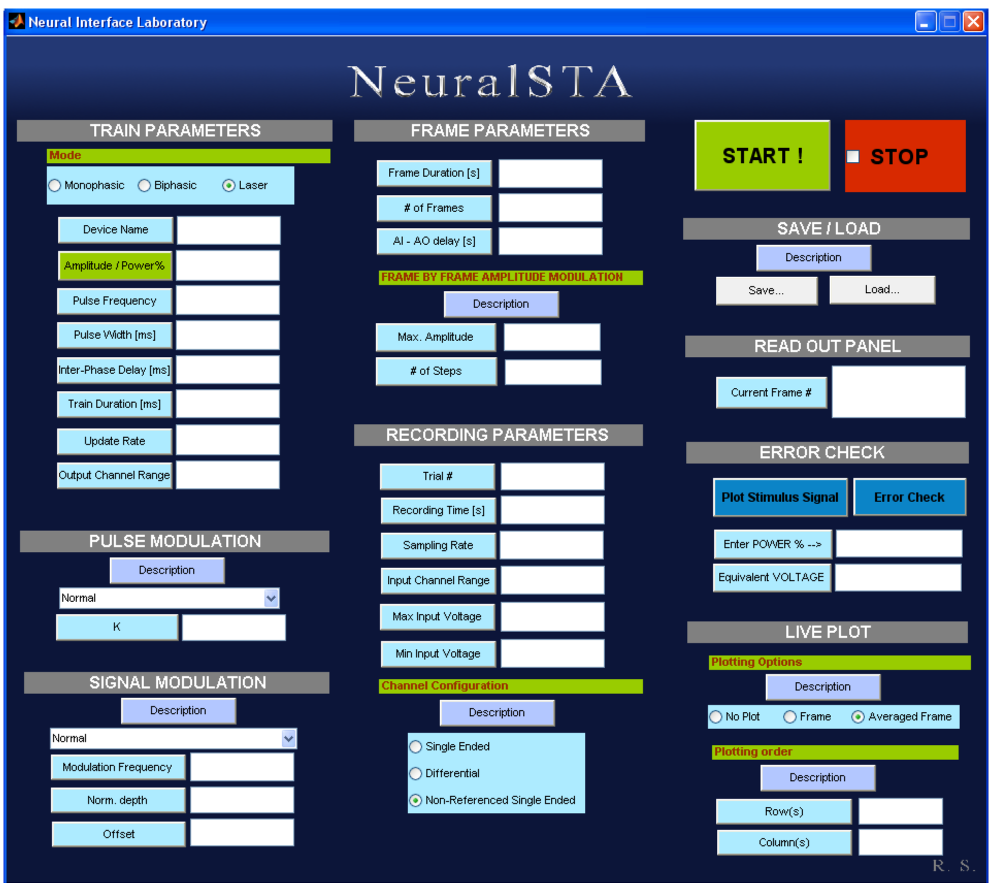


Figure 4.1 NeuralSTA front panel

A guide with a detailed description of how to use different button groups in NeuralSTA (Figure 4.1) is presented in Appendix C.

A guide for setting the parameters (Figure 4.1) for stimulation and recording along with live plot and starting and stopping the stimulation are presented in Appendix D.

4.1 Access to additional features of the program

Before initiating the trial, the user has the option to access various ‘help’ features to ensure that the signal that is generated free of errors. The following sections describe various types of help features that the user can access:

4.1.1. Description button

The description buttons give a detailed description of the panel that it is placed in. It can be accessed by left clicking on the push-button in the same way as the blue description buttons. Figure 4.2 is a picture of a description button.

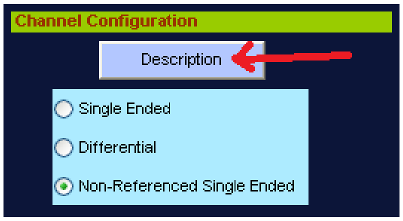


Figure 4.2 Description button for the panel

To get a description of the parameter, left click on the push button next to the parameter edit-box. Then a help window appears with a description of the parameter**.**

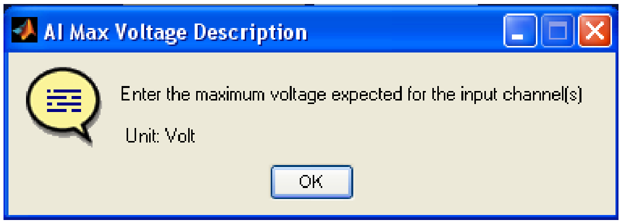


Figure 4.3 Pictorial guides to accessing the help feature for parameters

4.1.2. Read out panel

Figure 4.4 is a snap-shot of an example of read out panel:

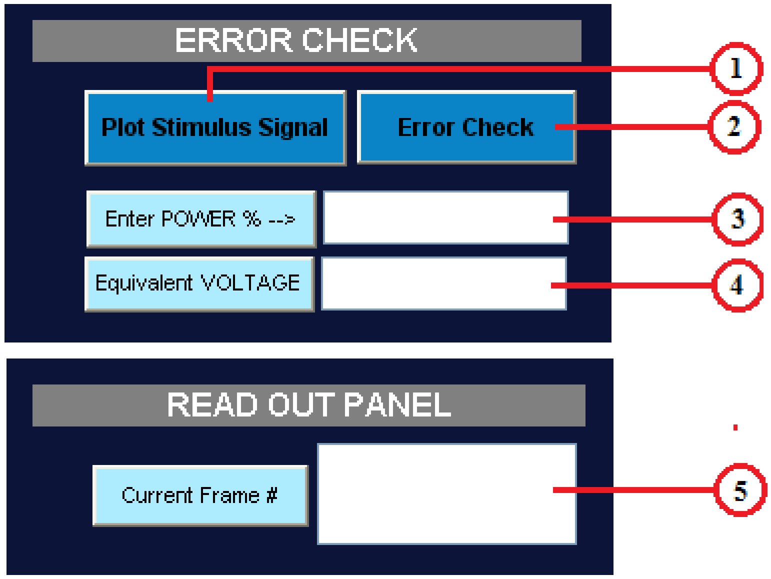


Figure 4.4 Labeled view of the read out panel

The following is a detailed explanation of each of the buttons and parameters:

1. Plot Stimulus Signal: After entering values for the pulse, train and frame parameters, the user has the option to observe the frame signal before executing the experiment. In order to do this the user can click the 'Plot Stimulus Signal' push-button. Once this button clicked, a new window with a plot of the frame signal appears. An example of the plot is shown below:

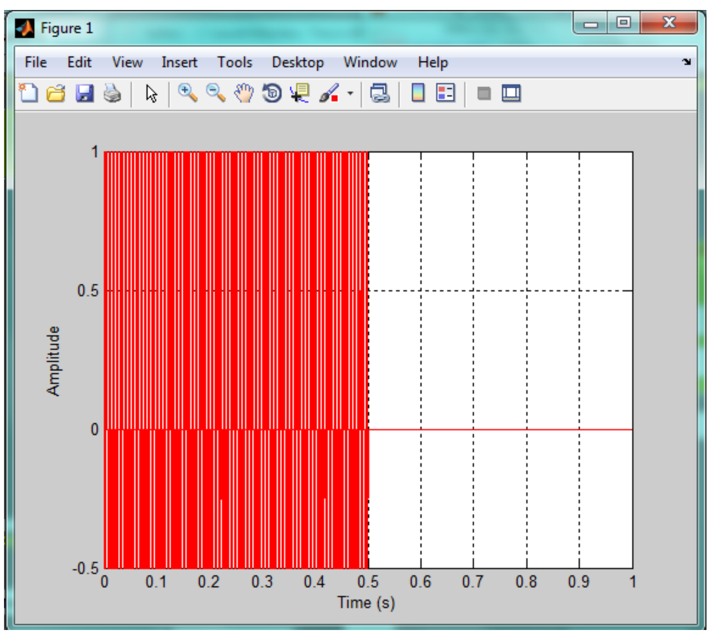


Figure 4.5 Example of plot generated by clicking the 'Plot Stimulus Signal' push button

2. Error Check: This button can be used to verify if all the parameters are cohesive. If there are any conflicting values in the parameters entered, the error check button generates a new window that displays a list of the errors in the parameters. The list also contains the location of the parameter causing the error, followed by a description of the error and for some parameters suggestions to eliminate the error. An example of the error check window is displayed in the figure below:

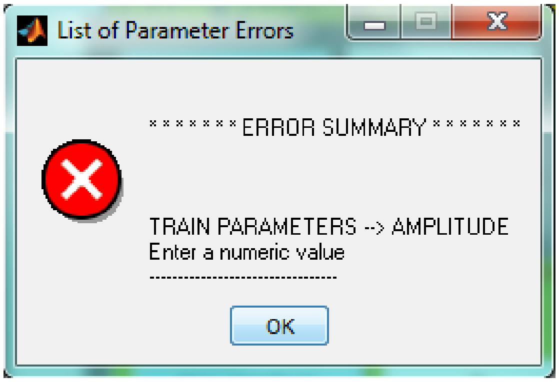


Figure 4.6 Example of 'List of parameter errors' window listing the error.

If there are no errors, then it generates a window (Figure 4.7 indicating that there are no errors:

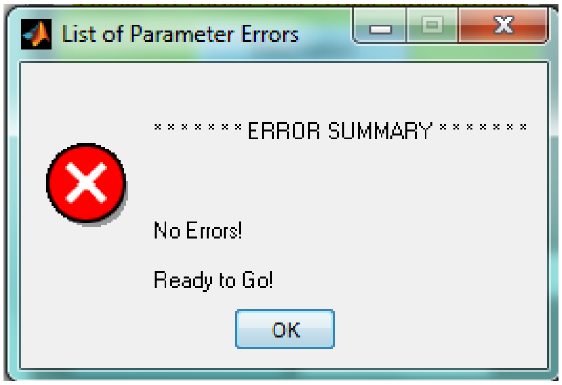


Figure 4.7 Example of 'List of parameter errors' window when there is no error in the parameter inputs.

3. Enter power %: When the laser mode option is chosen, the amplitude is entered in terms of percentage of the full laser power. In order to verify the equivalent voltage value, the user clicks on the push button and the value appears in the edit-box below it. The specific laser module (DLS-500-830FS-100) is available from various companies including Newport Corp., CA.
4. EQUIVALENT Voltage: when the laser mode is chosen. This section displays the equivalent voltage value of the power percent value entered in the edit-box above it. Alternatively, this can be used to enter the voltage value and

calculate the equivalent power percentage that would appear on the edit box above it.

5. Current Frame #: As the stimulation is being executed, this section actively displays the current number of the frame that is being outputted through the Analog Output channels.

4.1.3. Save / load

Figure 4.8 is a snap shot of the save / load panel:

Figure 4.8 Labeled view of the Save / load panel.

The following is a detailed explanation of each of the buttons:

1. Save: This push button is used to save a list of parameters in a '.mat' format in a user specified location. This saved file can be loaded for future experiments. An example of the window that appears after clicking the 'save' push-button is shown in Figure 4.9:

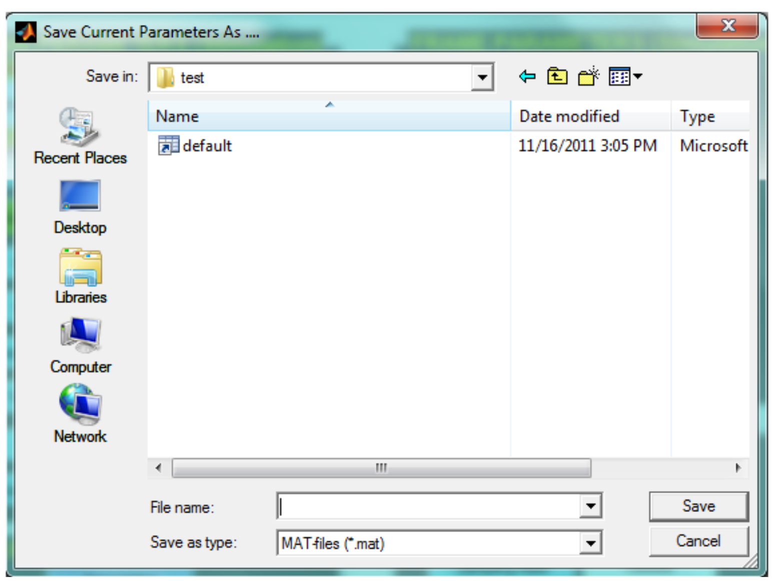


Figure 4.9 Example of the window that appears after clicking the 'save' button.

2. Load: In order to load a previously saved file, the user has the option to click on the load button to get access to the desired parameter file. An example of

the window that appears after clicking the ‘load’ push-button is shown in Figure 4.10:

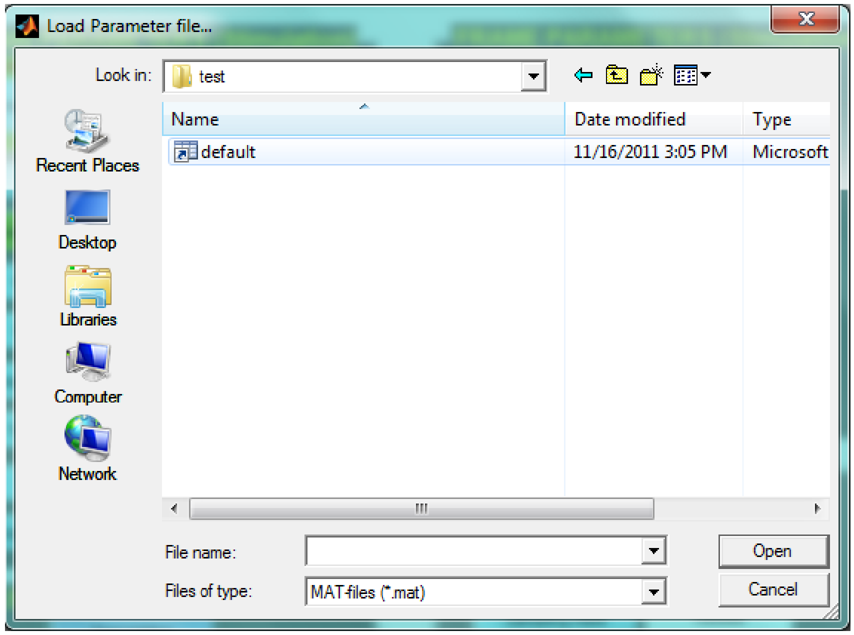


Figure 4.10 Example of the window that appears clicking the ‘load’ button.

CHAPTER 5

RESULTS

A series of tests were conducted in order to verify that NeuralSTA is outputting the appropriate stimulation signal and recording simultaneously. One analog output channel was connected to one analog input channel to create a feedback loop. After setting the channels, various stimulation signals were created to verify that the correct waveform was being outputted by the analog output channel and was being recorded by the analog input channel.

Various parameter settings were used to create multiple stimulation waveforms. For testing different stimulation waveforms, the recording parameters in Table 5.1 were kept the same:

Table 5.1 Recording parameters for testing

Recording parameters	Value
Trial #	1
Recording time [s]	1
Sampling Rate	100000
Input Channel Range	0
Input Channel Range	0
Max Input Voltage	10
Min Input Voltage	-10
Channel Configuration	Single Ended

The following sections list a set of parameter settings along with a plot of the data recorded and their explanation.

5.1. Stimulation signal in biphasic, monophasic and laser modes

For signals in all the three modes, the parameter values in Table 5.2 were used. Note that the signals were recorded through the analog input through the feedback connection after being outputted by the analog output.

Table 5.2 Parameters for stimulation waveform in biphasic, monophasic and laser modes

Parameter list	Value
Train parameters	
Pulse Frequency	300
Pulse Width [ms]	0.2
Inter-phase delay [ms]	0.2
Train Duration [ms]	500
Update Rate	100000
Output channel Range	0
Pulse Modulation Section	
Function	Normal
K	1
Signal Modulation Section	
Funciton	Normal
Modulation Frequency	
Normal Depth	
Offset	

Parameter list	Value
Frame parameters Section	
Frame Duration [s]	1
# of Frames	5
AI - AO delay [s]	0
Frame by Frame Amplitude Modulation Section	
Max. Amplitude	
# of steps	

Biphasic signal:

Figure 5.1 is a plot of the data collected with the parameters specified in Table 5.1 and Table 5.2 with a pulse amplitude of one with five frames during one stimulation period.

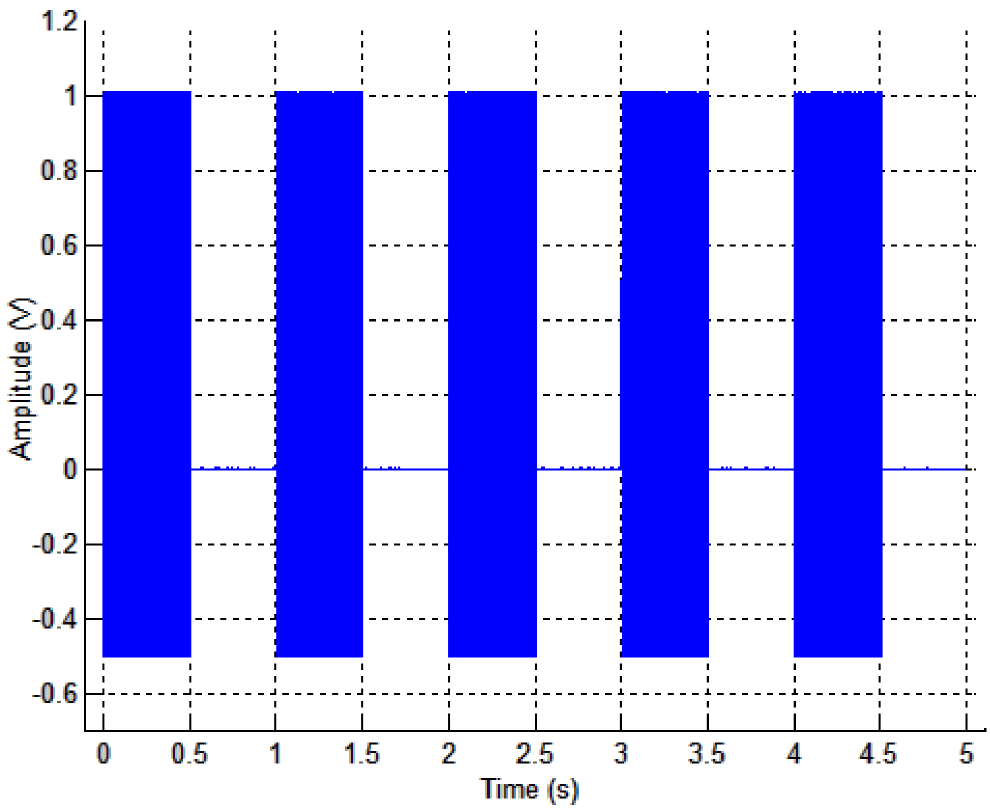


Figure 5.1 Biphasic stimulation signal with five frames

Figure 5.2 shows the expanded view of the first frame of the recorded signal.

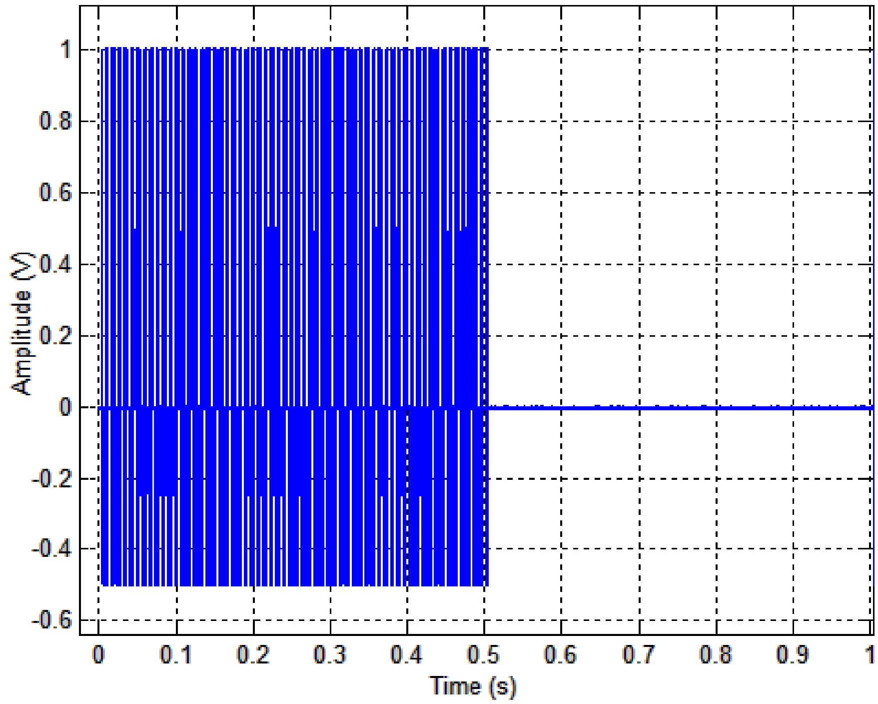


Figure 5.2 Biphasic stimulation frame

Figure 5.3 displays the individual pulses in the frame. The pulses are biphasic and have the correct amplitude as specified.

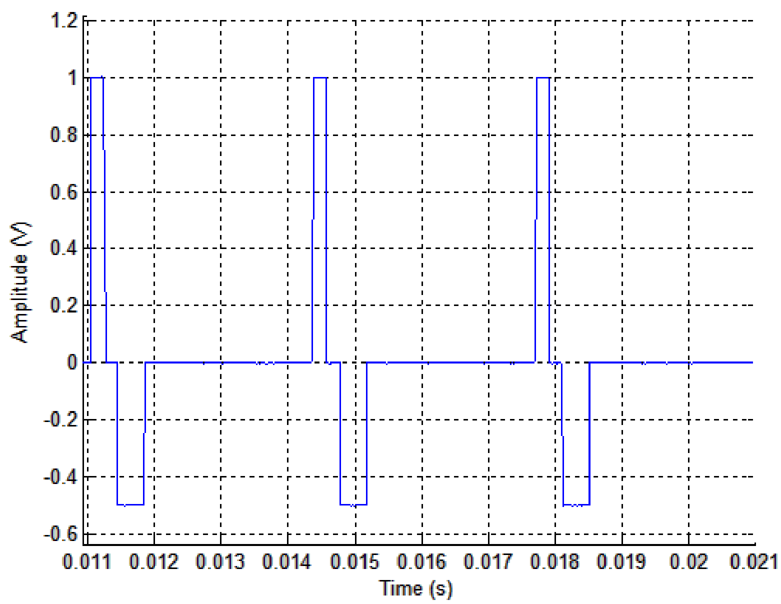


Figure 5.3 Biphasic simulation pulses

Monophasic signal:

Figure 5.4 is a plot of the data collected with the parameters specified in Table 5.1 and Table 5.2 with pulse amplitude of one, and five frames during one stimulation period.

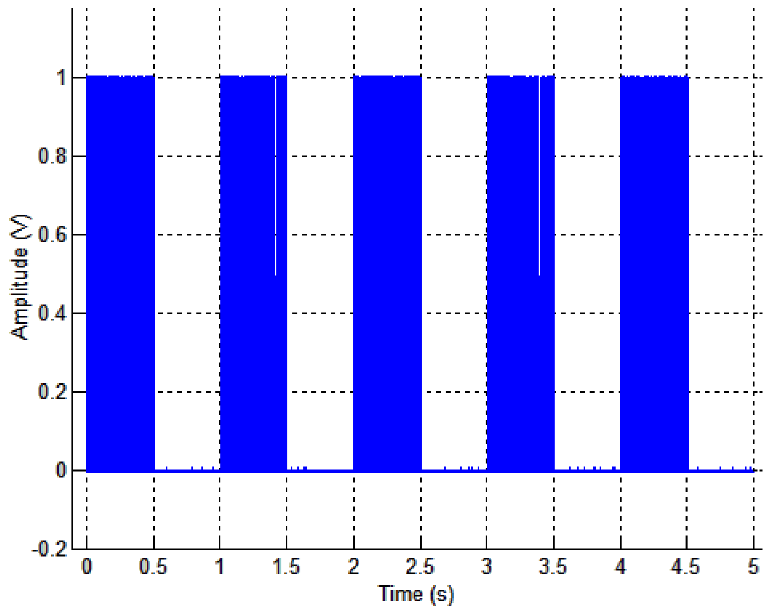


Figure 5.4 Monophasic stimulation signal with five frames

Figure 5.5 represents the first frame of the recorded signal.

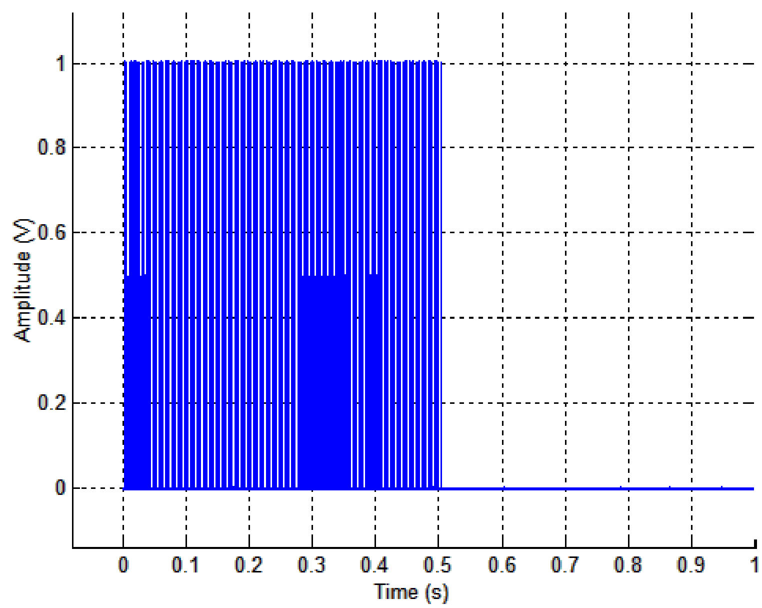


Figure 5.5 Monophasic stimulation frame

Figure 5.6 displays the individual pulses in the frame. As it can be seen, the pulses only have a positive phase and the correct amplitude as specified by the user.

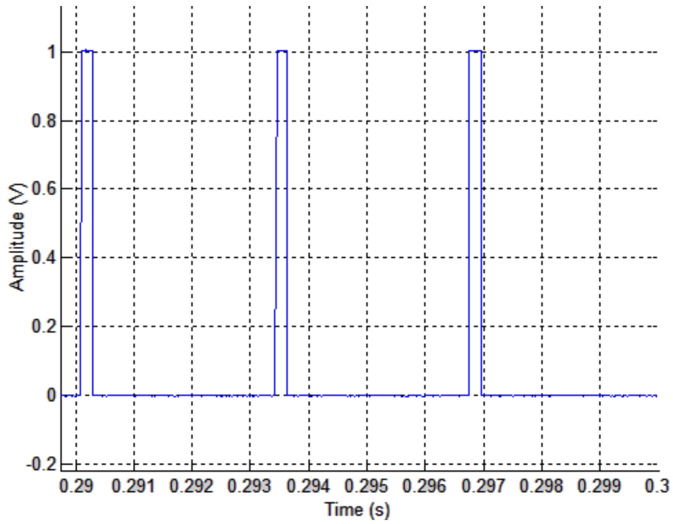


Figure 5.6 Monophasic stimulation pulses

<u>Laser signal:</u>

Figure 5.7 is a plot of the data collected with the parameters specified above with 88% laser power (equivalent voltage = 1V) for the pulse amplitude with five frames during one stimulation period.

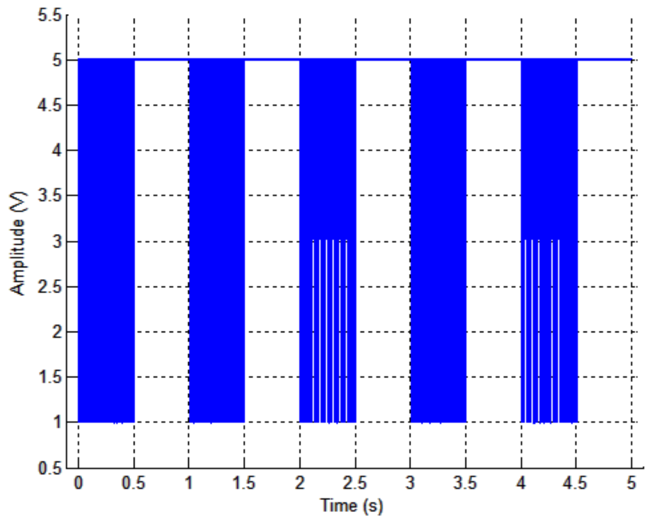


Figure 5.7 Laser stimulation signal with five frames

Figure 5.8 represents the first frame of the recorded signal.

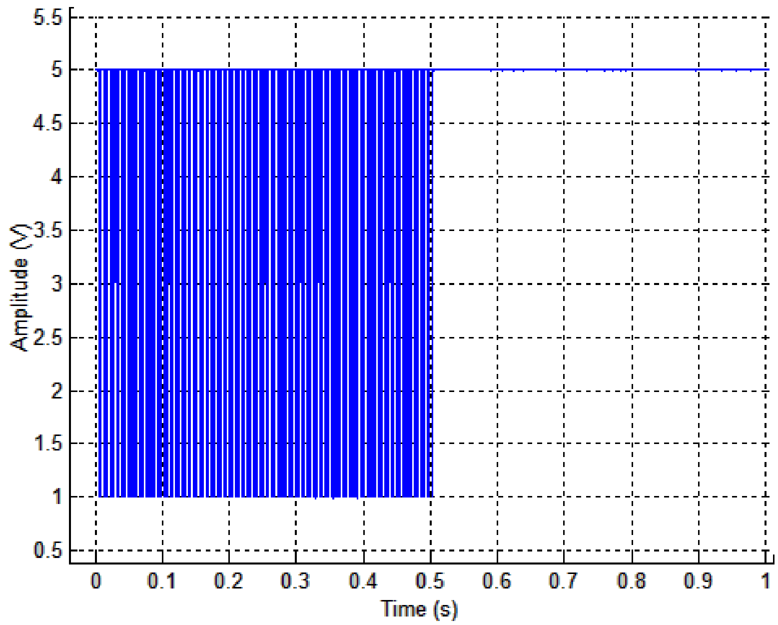


Figure 5.8 Laser stimulation frame

Figure 5.9 displays the individual pulses in the frame. As it can be seen the pulses first have a 5V phase and then a negative excursion, the amplitude of which is dependent on the % power given by the user.

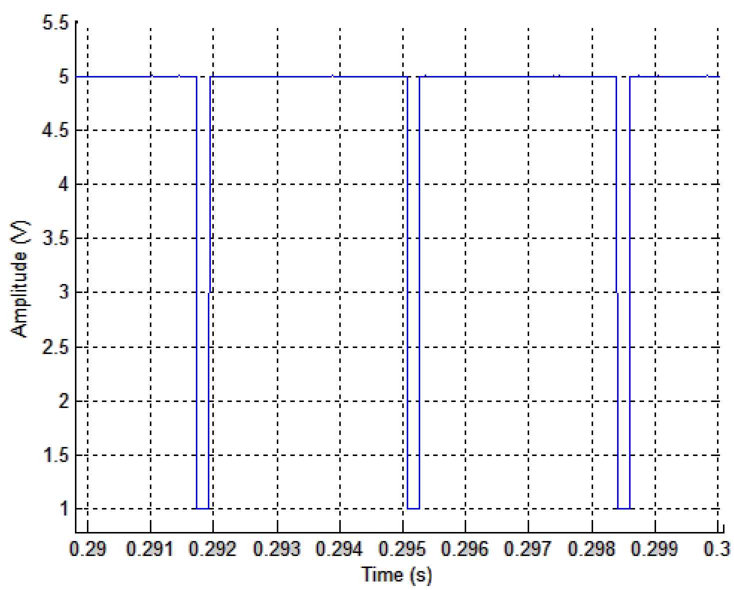


Figure 5.9 Laser stimulation pulses at 88% power level

5.2. Stimulation waveform with pulse modulation

For testing the pulse modulation function, the stimulation parameters in table 5.3 were kept constant for the biphasic, monophasic and laser modes.

Table 5.3 Stimulation parameters for generating simulation signal with pulse modulation

Parameter list	Value
Train parameters	
Pulse Frequency	300
Pulse Width [ms]	0.2
Inter-phase width [ms]	0.2
Train Duration [ms]	500
Update Rate	100000
Output channel Range	0
Pulse Modulation Section	
K	1
Signal Modulation Section	
Function	Normal
Modulation Frequency	
Normal Depth	
Offset	
Frame parameters Section	
Frame Duration [s]	1
# of Frames	5
AI - AO delay [s]	0
Frame by Frame Amplitude Modulation Section	
Max. Amplitude	
# of steps	

<u>Biphasic signal:</u>

Figure 5.10 is a plot of the data collected with the parameters specified in Table 5.1 and Table 5.3 along with pulse amplitude of one volt with an <u>exponential increase</u> waveform for pulse modulation. The stimulation signal has five frames as shown in Figure 5.10.

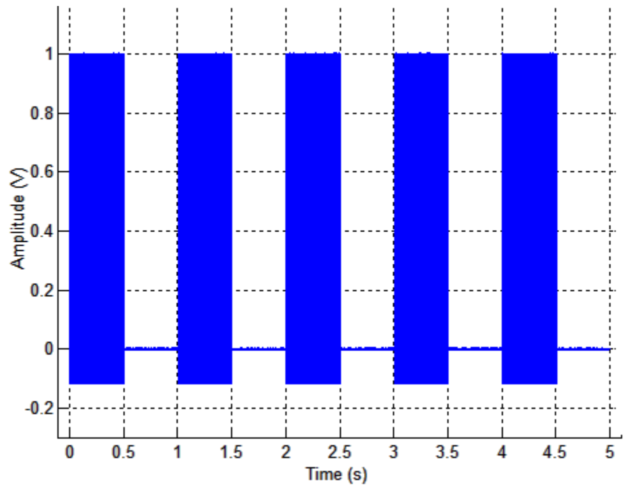


Figure 5.10 Biphasic stimulation signal with pulse modulation

Last, Figure 5.11 details the pulse modulation (exponential increase waveform) that was applied to the individual pulses.

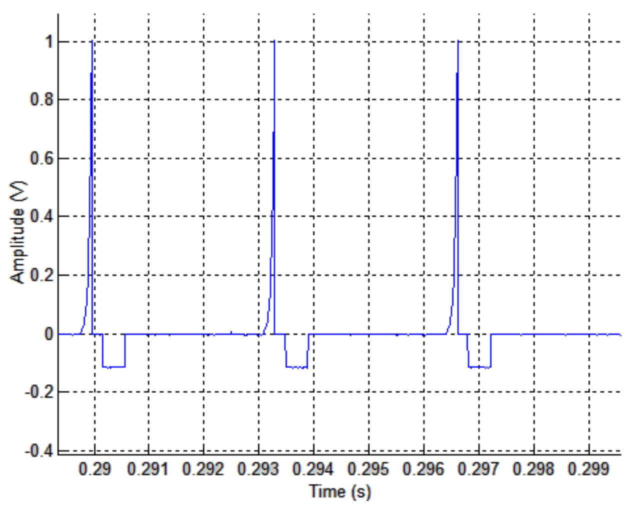


Figure 5.11 Biphasic stimulation pulses with pulse modulation

Monophasic signal:

Figure 5.12 is a plot of the data collected with the parameters specified in Table 5.1 and Table 5.3 along with pulse amplitude of one volt with an exponential decrease waveform for pulse modulation. The stimulation signal has five frames.

Figure 5.12 details the pulse modulation (exponential decrease waveform) that was applied to the pulses.

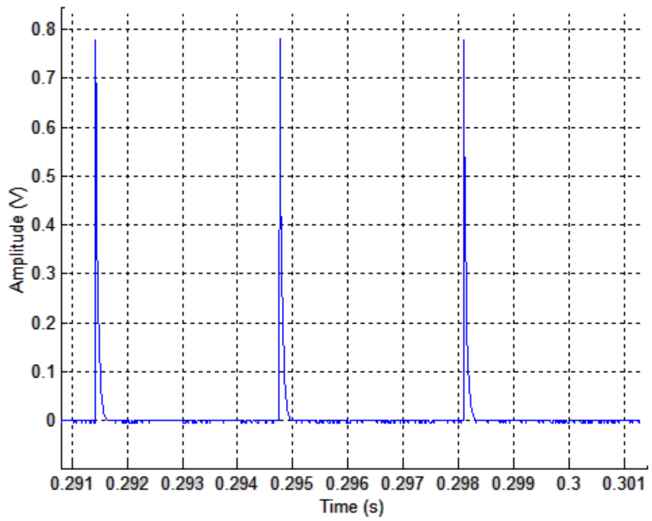


Figure 5.12 Monophasic stimulation pulses with pulse modulation (exponential decrease)

Laser signal:

Figure 5.13 is a plot of the data collected with the parameters specified in Table 5.1 and Table 5.3 along with pulse amplitude of 22% power (equivalent voltage = 3.6 V)with a Gaussian waveform for pulse modulation. The stimulation signal has five frames lasting for 5seconds.

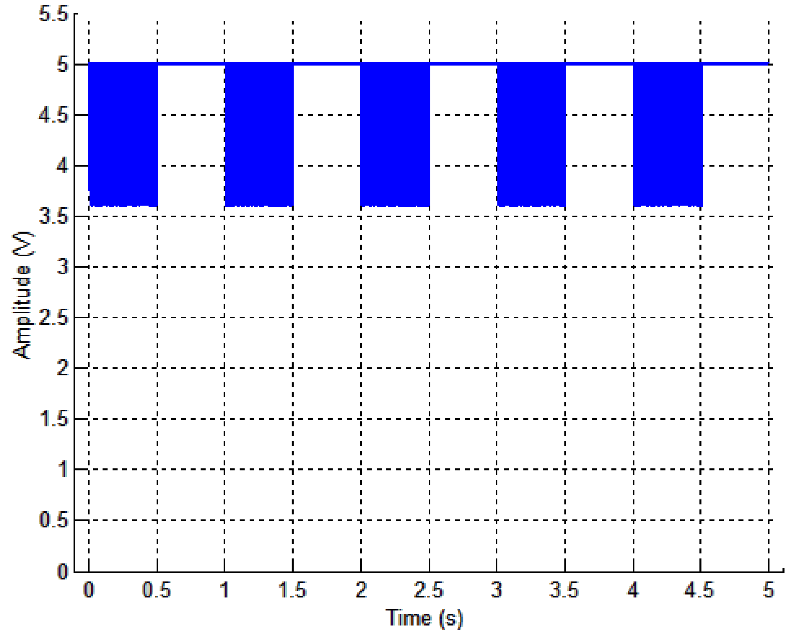


Figure 5.13 Laser signal with pulse modulation

Last, Figure 5.14 details the pulse modulation (Gaussian waveform) that was applied to the pulses.

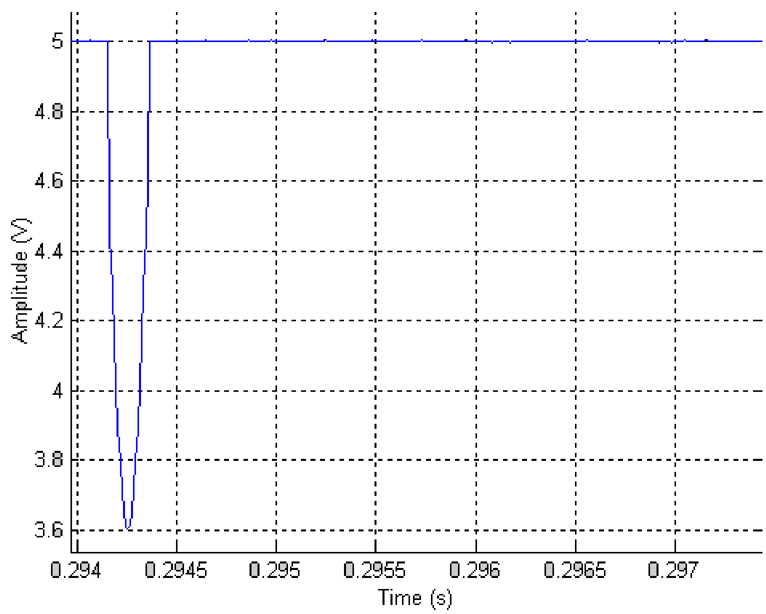


Figure 5.14 Laser signal pulse with pulse modulation

5.3. Stimulation waveform with signal modulation

For signal modulation of the frame, the minimum value of the frame takes precedence. In a case where the sum of the vertical offset and the normal depth of the signal modulation waveform exceed the minimum amplitude of the pulse, the later takes precedence and the parts of the signal that exceed the minimum amplitude are cutoff (as shown in Figure 5.15). In a case where the sum of the vertical offset and the normal depth is less than the minimum amplitude of the pulse, the sum of the vertical offset and normal depth takes precedence (as shown in Figure 5.16). In Figure 5.17 a case where the first two cases do not occur is displayed.

For testing the signal modulation function the stimulation parameters listed in Table 5.1 and Table 5.4 were kept the same for the biphasic, monophasic and laser modes. The Table 5.4 displays the parameter list.

Table 5.4 Stimulation parameters for generating simulation signal with signal modulation

Parameter list	Value
Train parameters	
Amplitude	3
Pulse Frequency	300
Pulse Width [ms]	0.2
Inter-phase delay [ms]	0.2
Train Duration [ms]	500
Update Rate	100000
Output channel Range	0

Parameter list	Value
Pulse Modulation Section	
Function	Normal
K	1
Signal Modulation Section	
Function	Sine
Modulation Frequency	4
Frame parameters Section	
Frame Duration [s]	1
# of Frames	5
AI - AO delay [s]	0
Frame by Frame Amplitude Modulation Section	
Max. Amplitude	0
# of steps	0

Biphasic signal:

Figure 5.15 is a display of a biphasic signal with a sinusoid signal modulation. The vertical offset value of the signal is kept at 2.5V and the normal depth is kept at 0.7V. With the offset and the normal depth, the maximum amplitude is 3.2V, however since the minimum amplitude is specified to be 3V, any signal above 3V will be cutoff.

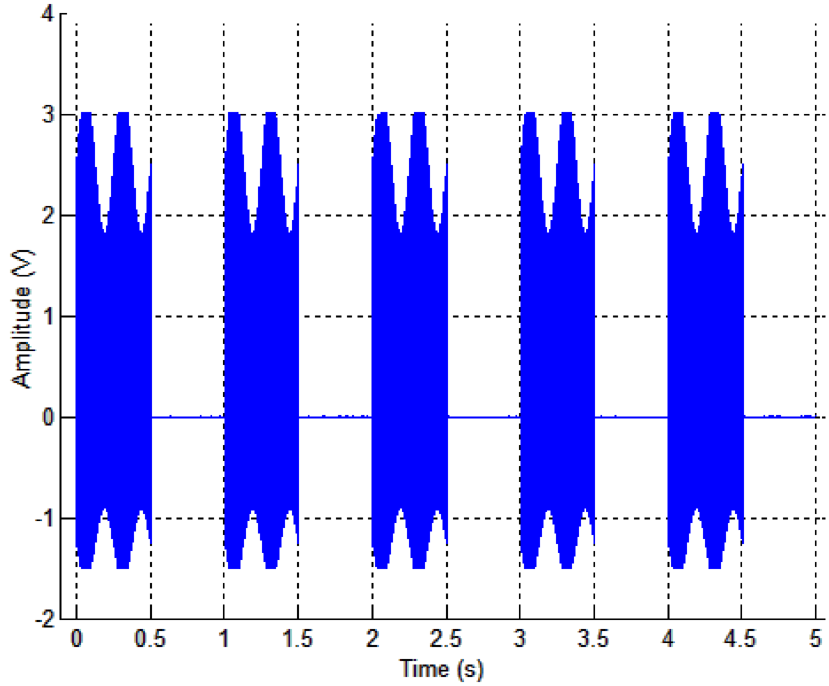


Figure 5.15 Biphasic stimulation signal with signal modulation and the final amplitude exceeding the minimum voltage

Figure 5.16 is a display of a biphasic signal with a sinusoid signal modulation. The vertical offset value of the signal is kept at 2.5V and the normal depth is kept at 0.3V. With the offset and the normal depth, the maximum amplitude is 2.8V. Since the minimum amplitude (3V) is greater than the sum, the sum will take precedence. As shown in the figure, the amplitude of the resultant is lower than the minimum amplitude specified by the user.

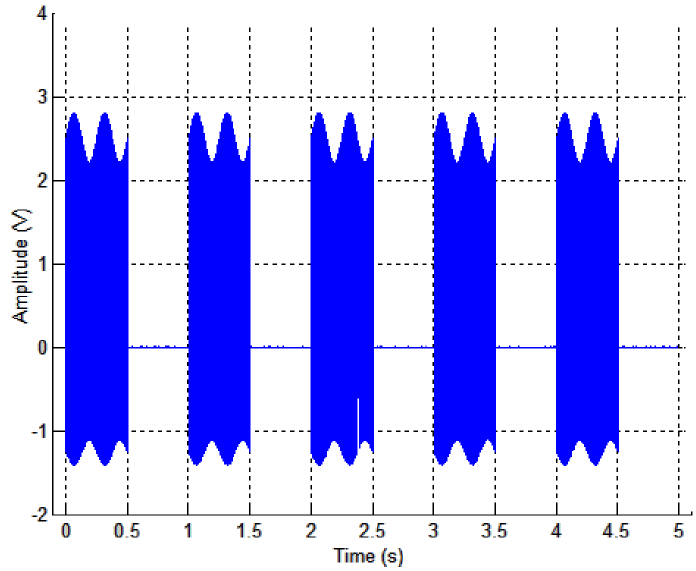


Figure 5.16 Biphasic stimulation signal with signal modulation with the final amplitude less than the minimum voltage value

The figure 5.17 is a display of a biphasic signal with a sinusoid signal modulation. The vertical offset value of the signal is set to 2.5V and the normal depth is set to 0.5V. The sum of the two parameters is 3V and the minimum amplitude is also 3V. There is no cutoff.

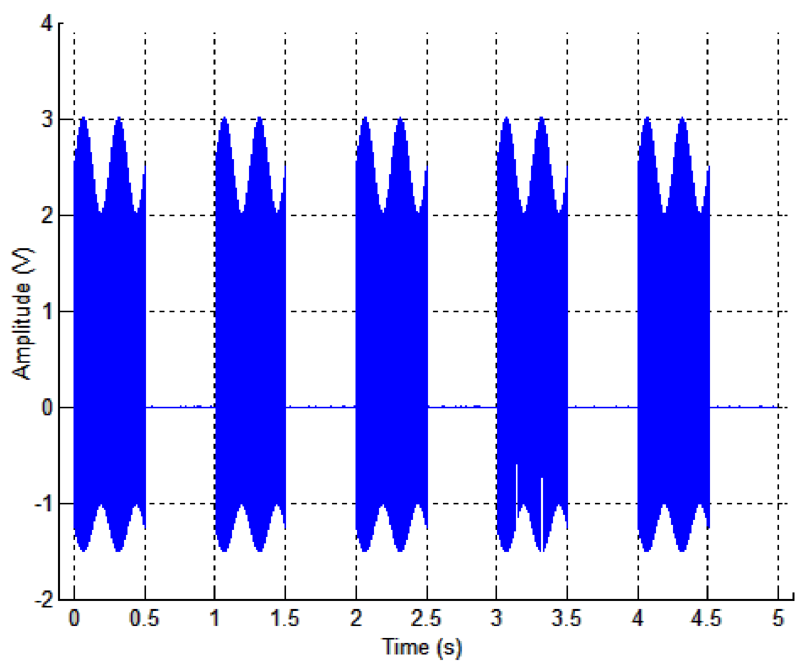


Figure 5.17 Biphasic stimulation signal with signal modulation with values equal to the minimum voltage value

5.4. Stimulation waveform with both pulse modulation and signal modulation

For creating the stimulation signal that has been modulated at the pulse level and the signal level, the parameters in Table 5.5 were chosen.

Table 5.5 Stimulation parameters for generating simulation signal with pulse modulation and signal modulation

Parameter list	Value
Train parameters	
Amplitude	3
Pulse Frequency	300
Pulse Width [ms]	0.2
Inter-phase delay [ms]	0.2
Train Duration [ms]	500
Update Rate	100000
Output channel Range	0
Pulse Modulation Section	
Function	Exponential increase
K	1
Signal Modulation Section	
Funciton	Sine
Modulation Frequency	4
Normal Depth	2.5
Offset	0.5
Frame parameters Section	
Frame Duration [s]	1
# of Frames	5

Parameter list	Value
AI - AO delay [s]	0
Frame by Frame Amplitude Modulation Section	
Max. Amplitude	0
# of steps	0

Biphasic signal:

Figure 5.18 presents a frame view of the stimulation signal collected in the feedback with both pulse modulation and signal modulation. The minimum amplitude was set to 3V, the signal modulation offset to 2.5V, and normal depth to 0.5V.

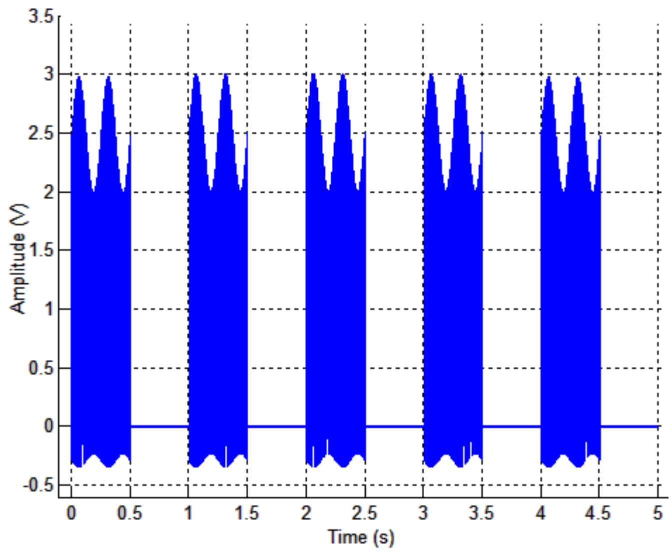


Figure 5.18 Biphasic signal with both pulse modulation and signal modulation

Using the same data as in Figure 5.18, Figure 5.19 details the signal from the first frame for duration of one second.

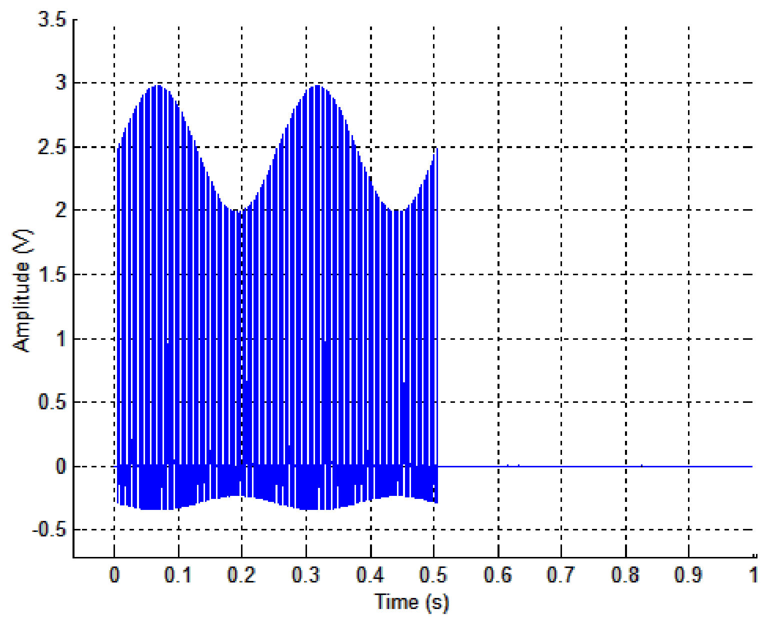


Figure 5.19 Biphasic frame with both pulse modulation and signal modulation

Last, Figure 5.20 details the pulse modulation (exponential increase waveform) that was applied to the pulses.

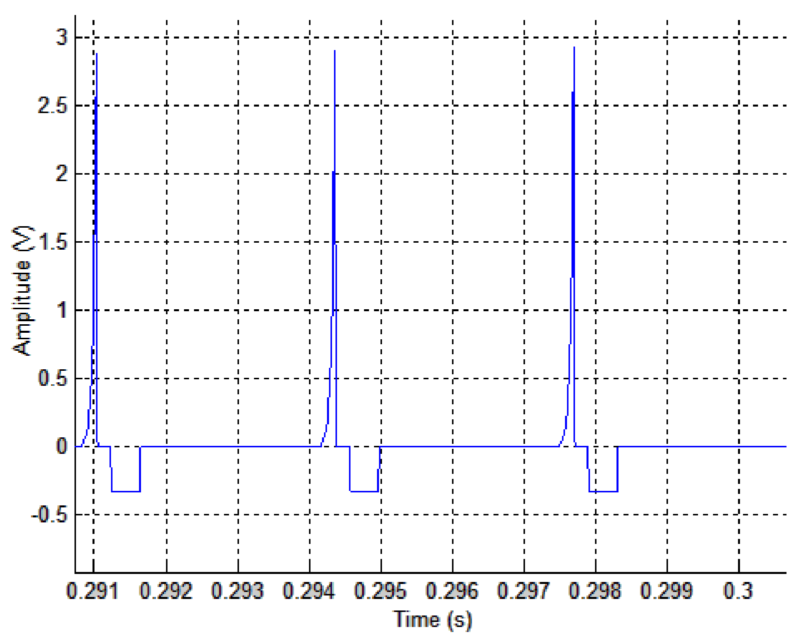


Figure 5.20 Biphasic pulse with both pulse modulation and signal modulation

<u>Monophasic signal:</u>

For the monophasic signal, the pulse modulation and signal modulation parameters were the same and the recoded signal was similar to the biphasic signal

<u>Laser signal:</u>

Figure 5.21 presents a frame view of the stimulation signal with both pulse modulation and signal modulation. The minimum amplitude was set to 88% power (equivalent amplitude = 1V) and the signal modulation offset to 22% power (equivalent amplitude = 3.6V) and normal depth to 88% power.

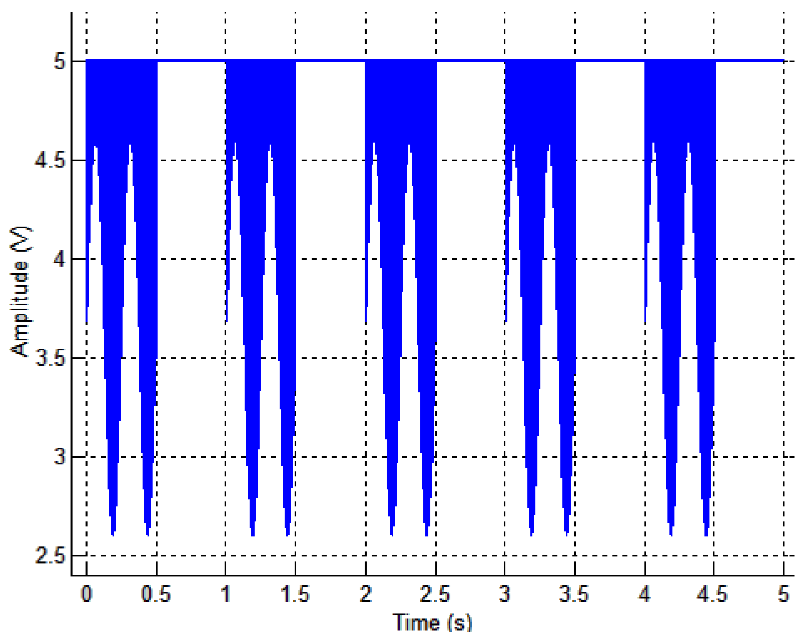


Figure 5.21 Laser signal with both pulse modulation and signal modulation

Using the same data as in Figure 5.21, Figure 5.22 details the signal from the first frame for duration of one second.

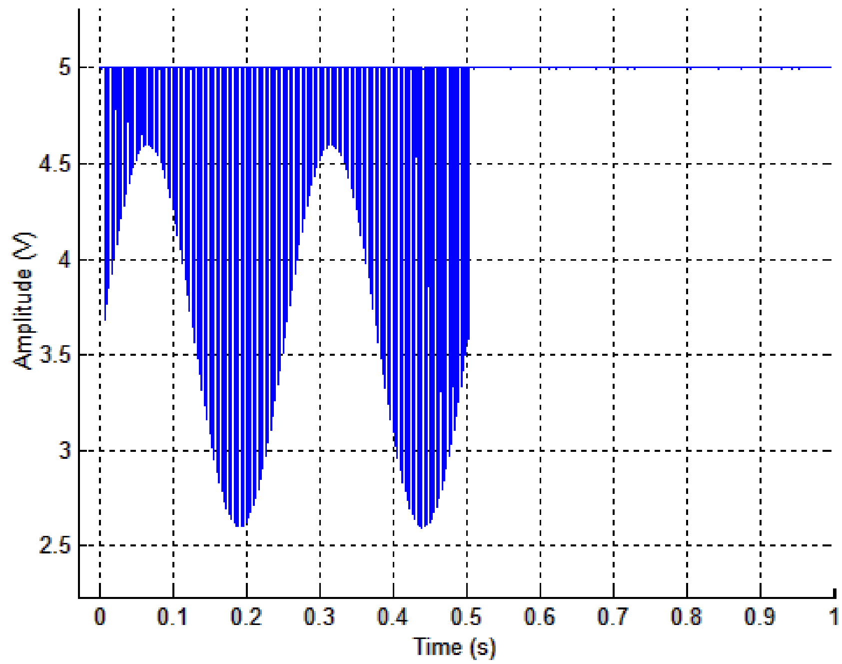


Figure 5.22 Laser signal frame with both pulse modulation and signal modulation

Last, Figure 5.23 details the pulse modulation (gaussian waveform) that was applied to the pulses.

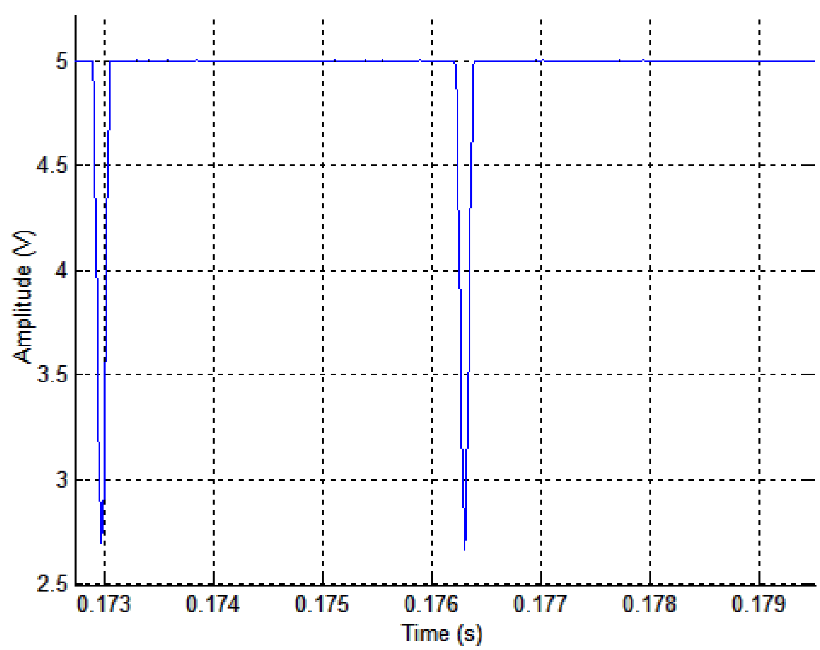


Figure 5.23 Laser signal with both pulse modulation and signal modulation

5.5. Stimulation waveform with frame by frame modulation

For testing the frame by frame modulation function the parameters in Table 5.6 were kept constant for the biphasic, monophasic and laser modes:

Table 5.6 Stimulation parameters for generating simulation signal with frame by frame modulation

Parameter list	Value
Train parameters	
Amplitude	1
Pulse Frequency	300
Pulse Width [ms]	0.2
Inter-phase delay [ms]	0.2
Train Duration [ms]	500
Update Rate	100000
Output channel Range	0
Pulse Modulation Section	
Function	Normal
K	1
Signal Modulation Section	
Function	Normal
Modulation Frequency	
Normal Depth	
Offset	
Frame parameters Section	
Frame Duration [s]	1
# of Frames	5

Parameter list	Value
AI - AO delay [s]	0
Frame by Frame Amplitude Modulation Section	
Max. Amplitude	5
# of steps	5

Biphasic signal:

Figure 5.24 is a plot of the data collected from five frames over a period of five minutes of stimulation (five frames with one minute per frame) and simultaneous recording. Here the minimum amplitude was 1V and the maximum amplitude was 5V with a step size of five. With these parameters the calculated (expected) amplitudes of the frames were:

- Frame → 1V
- Frame → 2V
- Frame → 3V
- Frame → 4V
- Frame →5V

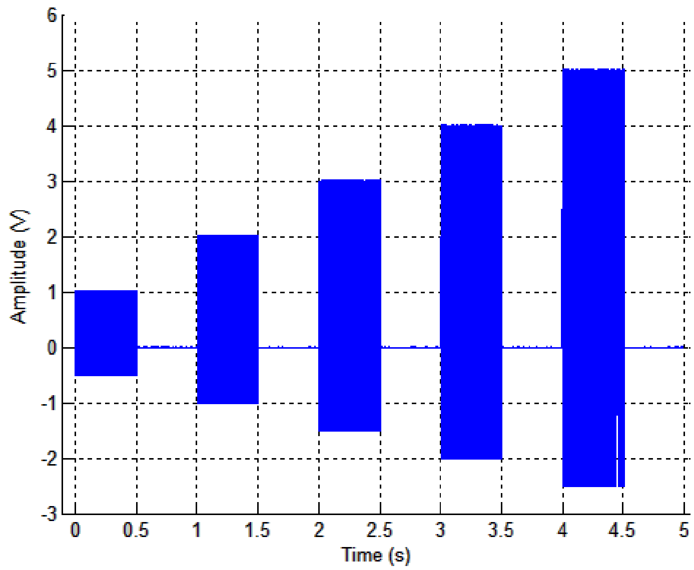


Figure 5.24 Biphasic stimulation signal with frame by frame amplitude modulation

Monophasic signal:

Figure 5.25 is a plot of the data collected from five frames over a period of five minutes of stimulation (five frames with one minute per frame) and simultaneous recording. Here the minimum amplitude was 1V and the maximum amplitude was 5V with a step size of five. With these parameters the calculated (expected) amplitudes of the frames were:

- Frame 1 → 1V
- Frame 2 → 2V
- Frame 3 → 3V
- Frame 4 → 4V
- Frame 5 →5V

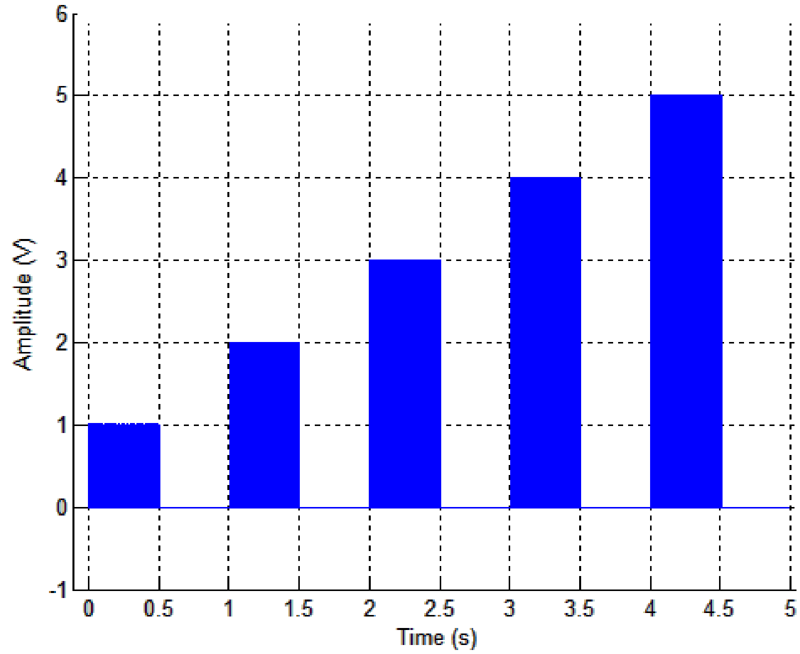


Figure 5.25 Monophasic stimulation signal with frame by frame amplitude modulation

<u>Laser signal:</u>

Figure 5.26 is a plot of the data collected from five frames over a period of five minutes of stimulation (five frames with one minute per frame) and simultaneous recording. Here the minimum amplitude was 88% power (equivalent voltage = 1V) and the maximum amplitude was 22% power (equivalent voltage = 3.6V) with a step size of five. With these parameters the calculated (expected) amplitudes of the frames were:

- Frame 1 → 1V
- Frame 2 → 1.65V
- Frame 3 → 2.3V
- Frame 4 → 2.95V
- Frame 5 →3.6V

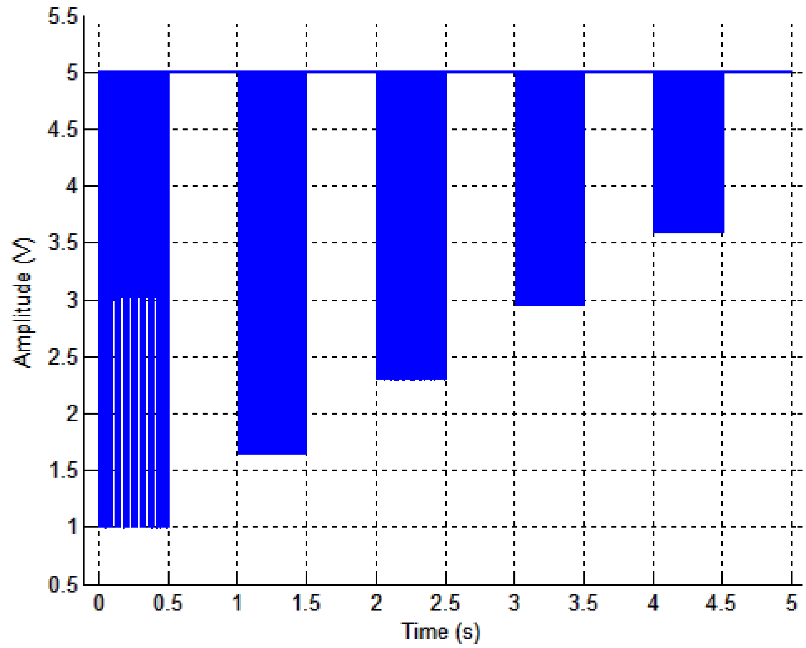


Figure 5.26 Laser signal with frame by frame amplitude modulation

5.6. Testing in animal experiments

NeuralSTA has been tested in animal experiments to stimulate and record data. In the study "Floating light-activated microelectrical stimulators testsed in the rat spinal cord," signals generated through NeuralSTA in the laser mode were used to control a laser (DLS-500-830FS-100). The laser beams (emitted from the laser) at near infrared (NIR) wavelengths were used as a means of energy transfer to control a floating light-activated microelectrical stimulator (FLAMES) for wireless intraspinal stimulation in rats. The forces elicited from these stimulations were measured with a force transducer from the ipsilateral forelimb of the rat.

Figure 5.27 is a plot of the stimulus signal that was generated in NeuralSTA. This signal has train duration of 0.5ms and frame duration of 2ms with a frequency of 50 Hz.

Figure 2.28 is a plot of the data that was collected in response to the stimulation from the laser and consequently FLAMES.

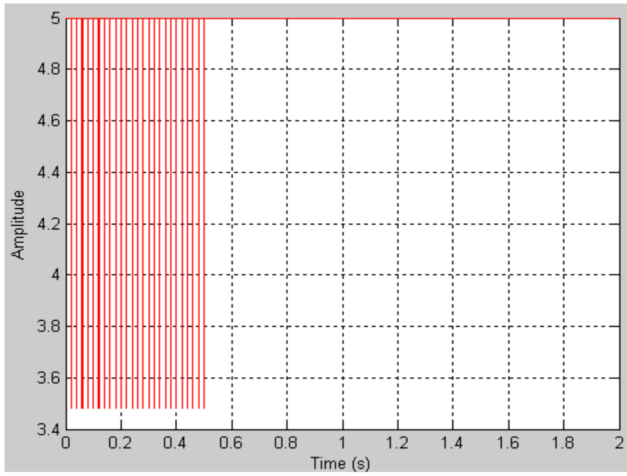
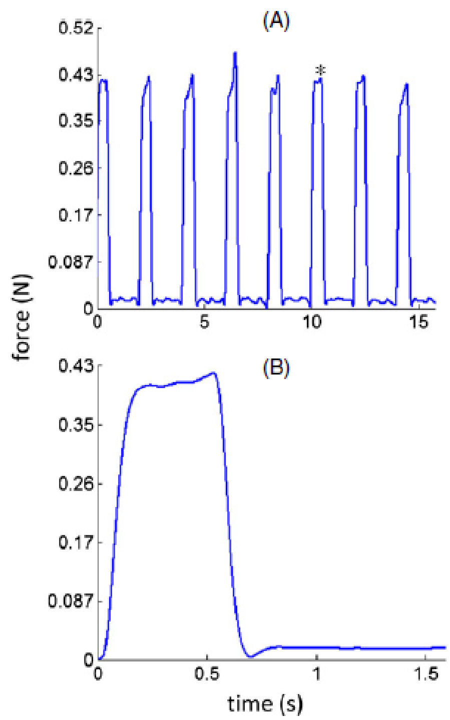


Figure 5.27 Frame signal in laser mode.



Vertical component of the elbow extension force recorded with a transducer attached to the ipsilateral hand in rat 3: (A) for a series of 0.5 s ON- 1.5 s OFF cycles. Each tetanic force is generated by a train of 50 Hz, 0.2 ms pulses. (B) Sample cycle expanded from the plot in A (marked with *) to demonstrate tetanus.

Figure 2.28 Collected data after stimulation

Source: [15]

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1. Conclusions

This report presented the development of NeuralSTA, a tool for spike triggered averaging with a direct electrical output or a laser control. A unique feature of NeuralSTA is that it can be used to the study neural response with non-rectangular stimulus waveforms.

NeuralSTA tools has been tested for the desired stimulation and recording features outlined in the Introduction and it has been verified that NeuralSTA meets the requirements at the pulse, train, and frame levels. It also includes frame by frame modulation. In addition, NeuralSTA allows accessing the DAQ card parameters. After entering all the parametes, the additional features of the program such as description buttons, error check button etc allow for verifying the parameters entered. Finally, NeuralSTA plots the averaged data after each frame and saves and loads a list of parameters.

6.2. Future work

NeuralSTA will be used to study the effect of non-rectangular waveforms as comporaed to the traditional rectangular waveform, thereby finding out optimal waveform that can activate neural tissue with minimal charge. NeuralSTA will also be used to study the strength-duration curve of neural stimulation with these non-rectangular pulse waveforms.

Additionally, NeuralSTA can be used to test if more natural forces and limb movement patterns can be generated by modulation of the stimulation pattern as a function of time, e.g. sinusoidal or triangular. This can be tested in the peripheral nerves or by intraspinal microstimulation experiments. Using NeuralSTA, one can also automatically search for the acrivation threshold in neural tissue by increasing the stimulus train amplitude in steps in each frame. This feature of the software can be improved by analyzing the recorded evoked potentials after each frame and automatically stopping the stimulations once the threshold is exceeded.

APPENDIX A

GUIDE TO USING AMPLITUDE MODULATION BY VARYING AMPLITUDES

Table A.1 Amplitude values for seven frames with varying minimum and maximum amplitudes and constant number of steps and frames

Amplitude (volt)							
	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5	Frame 6	Frame 7
Min amplitude = 1 Max amplitude = 7 *	1	2	3	4	5	6	7
Min amplitude = 2 Max amplitude = 7 *	2	3.5	5	6.5	8	9.5	11
Min amplitude = 2 Max amplitude = 15 *	2	4.167	6.333	8.5	10.667	12.833	15
Min amplitude = 14 Max amplitude = 15 *	14	14.167	14.333	14.5	14.667	14.833	15
Min amplitude = 15 Max amplitude = 15 *	15	15	15	15	15	15	15

*Number of steps = Number of frames = 7

In Table A.1, the minimum and maximum amplitudes were varied (keeping the number of steps and frames constant) to show the change in amplitudes in each consecutive frame. Note that the first frame's amplitude is the minimum amplitude (pulse amplitude x K-value) and the last frame's amplitude is the maximum amplitude specified by the user.

APPENDIX B

GUIDE TO USING AMPLITUDE MODULATION BY VARYING FRAMES AND STEPS

Table B.1 Amplitude values for seven frames with varying number of steps and frames and constant minimum and maximum amplitudes

Amplitude (volt)							
	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5	Frame 6	Frame 7
# of steps = 7 # of frames = 7 *	2	2.833	3.667	4.5	5.333	6.167	7
# of steps = 3 # of frames = 7 *	2	4.5	7	7	7	7	7
# of steps = 7 # of frames = 3 *	2	2.833	3.667	No frame	No frame	No frame	No frame

***** Minimum amplitude = 2; Maximum amplitude = 7

In Table B.1, the number of steps and frames were varied (keeping the minimum and maximum amplitudes constant) to show the how the variation in the number of steps and frames affect the amplitude of the consecutive frames. When the number of the frames exceeds the number of steps, the frames uses the maximum amplitude that was specified by the user. Additionally, if the number of frames (say n) is less than the number of steps (say m), the frame only uses the first n values for stimulation.

APPENDIX C

GUIDE TO USING BUTTON GROUPS

In NeuralSTA, there are several push buttons that are used for various functions. These push buttons can simply be accessed by a single left click on the push button (the yellow star symbolizes a left click).

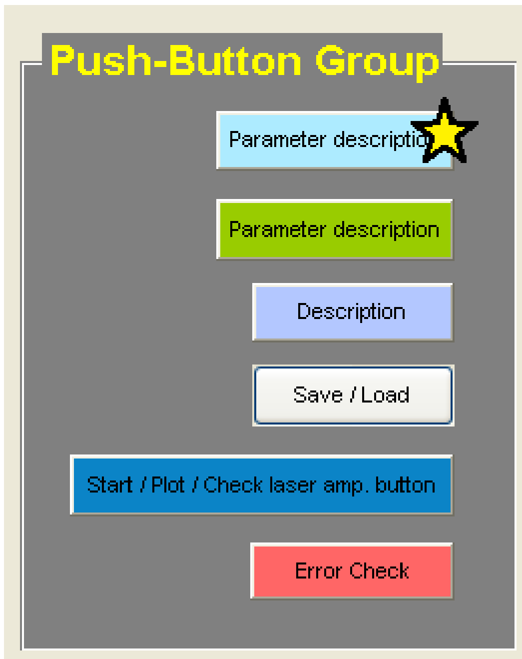


Figure C.1 Description of the usage of a push-button

Each of the sections is organized under a brief heading (in white in the program) that gives an indication of the parameters that are pertinent to that heading. In the picture below, the panel is labeled as 1.

There are several push buttons that are color coded to implement different actions. Figure C.2 shows different colors of push buttons that are used in the program.

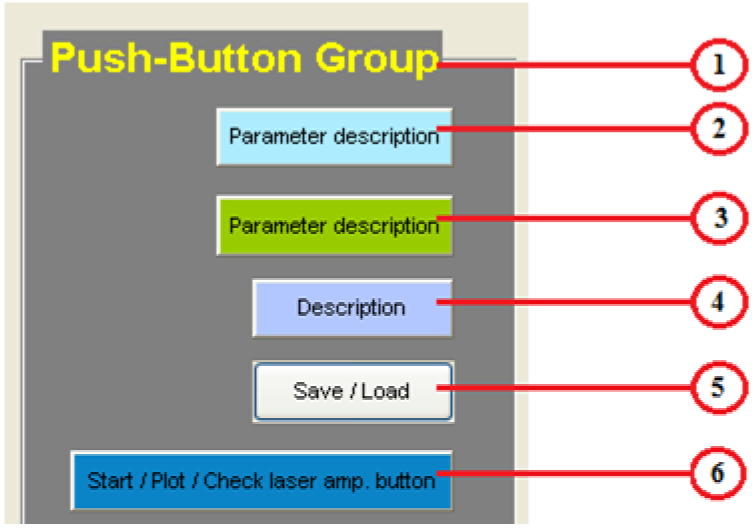


Figure C.2 Different types of push buttons

1. Section panel

2. The light blue buttons are present on the left side of every edit-box. Upon clicking it a help window appears. This window gives a description of the parameter along with the unit of the parameter. Some help windows also give some helpful suggestions to the user.

3. A green push button in the program indicates to the user that he/she has to enter an amplitude value in volts for monophasic and biphasic mode or enter a power % for the laser mode.

4. The lavender push buttons generate a help window that gives description of the entire panel.

5. The grey push buttons are specifically for loading and saving the data into and from the current interface.

6. The blue push buttons do several functions as indicated by their title. One button is used to start the stimulation, the other one is to generate a plot of a frame before stimulation, and the next one is to check the equivalent voltage value of the amplitude power % for the laser.

Additionally, the "error check" button generates a window that checks if all the parameters in the program are cohesive. If not then it lists the errors in the values entered by the user. More detailed information about this button is given in the section below.

The sections where the user enters the value of various parameters are called 'edit-boxes.' The edit-boxes are always displayed in white color. In order to enter a parameter value, the user simply left clicks on the edit-box and enters the appropriate value. Figure C.3 is a simple pictorial guide for using an edit-box:

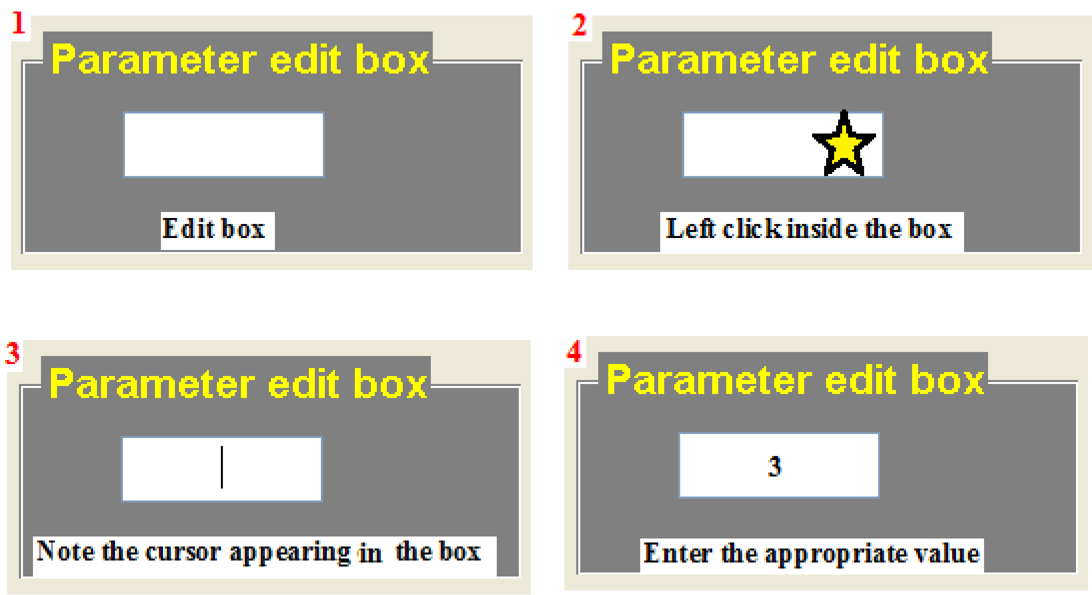


Figure C.3 Pictorial guides for using an edit-box

Next, there are two drop down menus in NeuralSTA under the Pulse Modulation panel and the Signal Modulation panel. This menu can be accessed by left clicking on the right end of the menu and again left clicking on the desired parameter. Figure C4 is a simple pictorial guide for using a drop down menu.

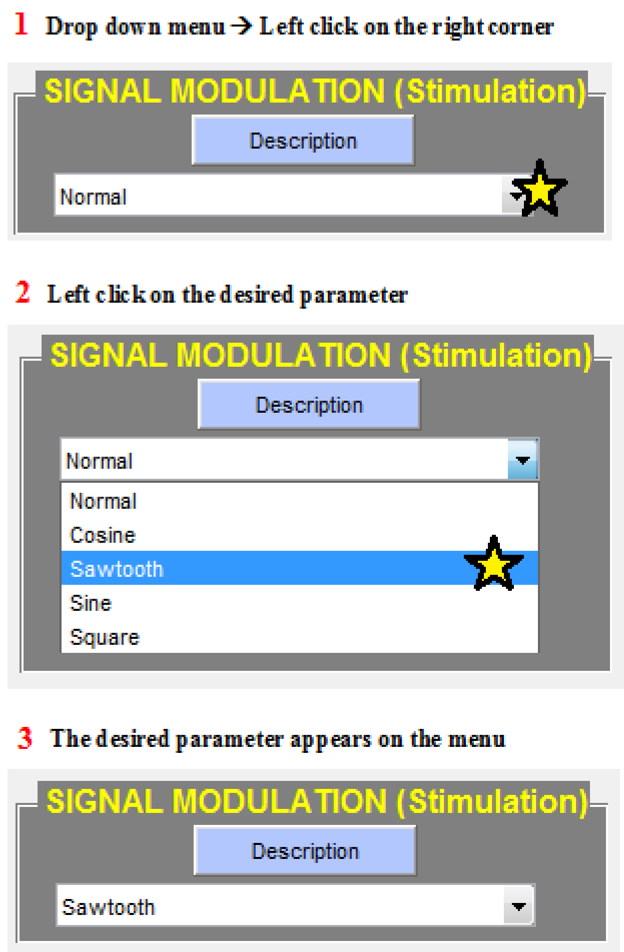


Figure C.4 Pictorial guides for using a drop-down menu

In addition to the drop-down menus, there are two different types of radio button groups. One type is the grouped button. In this type the user has the choice to use any one parameter from the list. In this type, at least one of the options is active (indicated by the green dot in the white circle). In order to access this option, the user simply left clicks on the desired option and the option becomes active (indicated by the green dot). Figure C.5 is a simple pictorial guide for using grouped radio buttons:

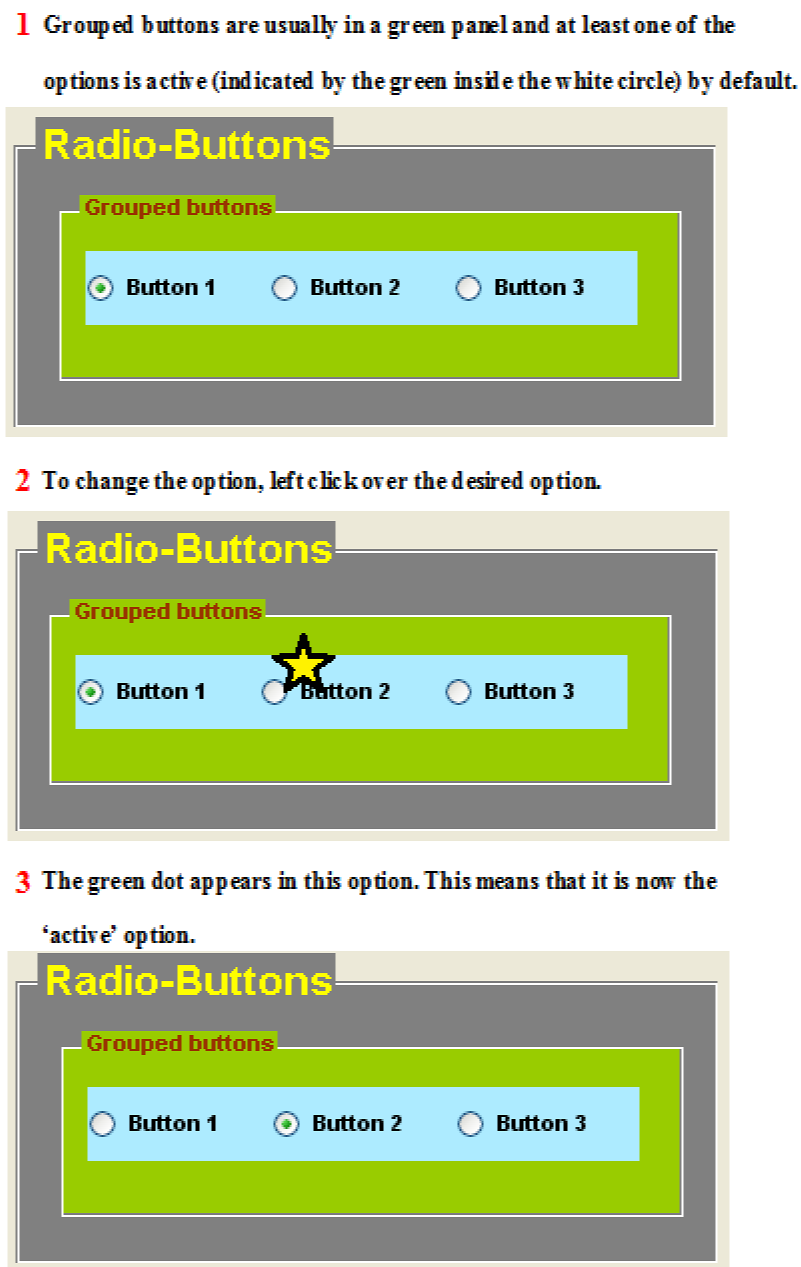


Figure C.5: Pictorial guides for using grouped radio-buttons

In the ungrouped radio button, simply left click on the button to activate it. And to deactivate it again left click on the button. Figure C.6 is a simple pictorial guide for using an un-grouped radio button:

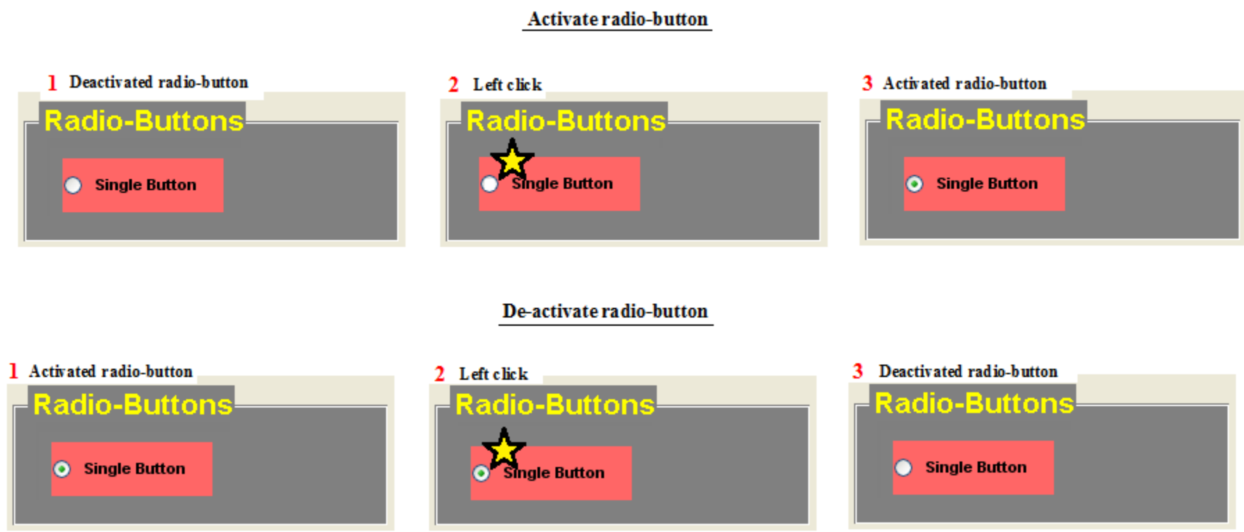


Figure C.6 Pictorial guides for using an un-grouped radio-button

APPENDIX D

GUIDE TO SETTING STIMULATION AND RECORDING PARAMETERS

Detailed guide to using 'pulse' modulation parameters

Figure D.1 is a snap shot of the Train parameter panel of NeuralSTA:

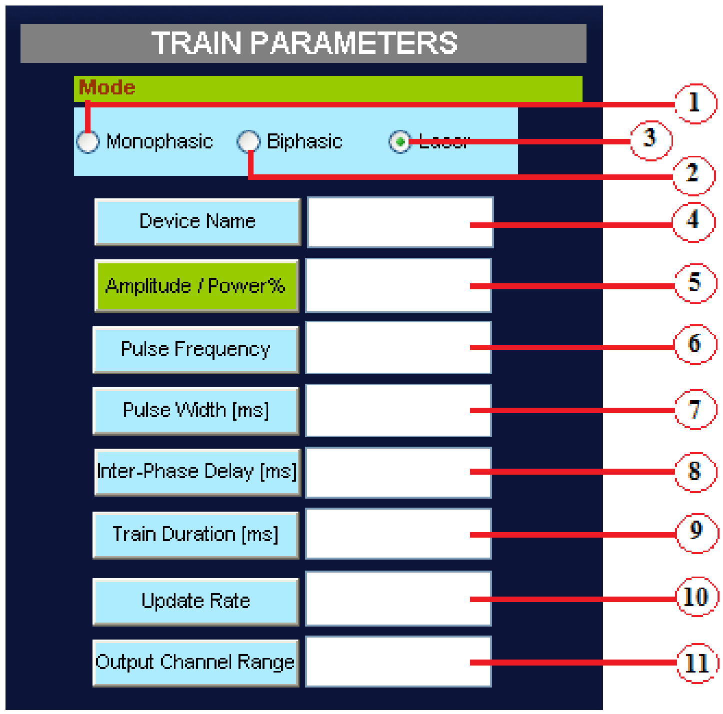


Figure D.1 Labeled view of the Train Parameter panel

The following is a detailed explanation of each of the parameters:

1. Monophasic: this option allows the user to create a monophasic signal (with a positive phase only). This value is dependent on a positive amplitude value given by the user.

2. Biphasic: this option allows the user to create a biphasic signal (with positive and negative phases). Both phase amplitudes are dependent on the positive amplitude value given by the user.

3. Laser: this option allows the user to create a signal for controlling the laser. This value is dependent on a positive amplitude value given by the user.

4. Device Name: This is the name of the Data Acquisition Card connected to the computer. The default name of the hardware usually starts with "Dev"

followed by a number. This value can be accessed by referring to either the National Instruments "Measurement and Automation Explorer" or by using the following Matlab® code to access the DAQ card name:

```
>>hw = daqhwinfo("nidaq");

>>hw.InstalledBoardIds

>>hw.BoardNames')
```

5. Amplitude/Power%: For Monophasic & Biphasic modes, this value is the amplitude (in volt) of the positive phase of the signal. For the Laser mode, this value is the power of the laser being used in percentages (%).

6. Pulse Frequency: This value is the frequency of the pulses in the train. Unit: Hertz.

7. Pulse Width: This value is the time for the first (positive) part of the pulse for the monophasic and biphasic modes, and is the last part of the pulse for the laser mode. Unit: milliseconds.

8. Inter-phase delay: For the biphasic mode only: the value is the delay between the positive and the negative phases of the pulse waveform.

9. Train Duration: Total time for the duration of the stimulus train. Unit: milliseconds

10. Update Rate: Sampling rate or update rate for the Analog Output channel(s). Unit: samples/second

11. Output Channel Range: Analog Output channel number(s). No units.

Detailed guide to using 'pulse' modulation parameters

Figure D.2 is a snap shot of the Pulse modulation panel of NeuralSTA.

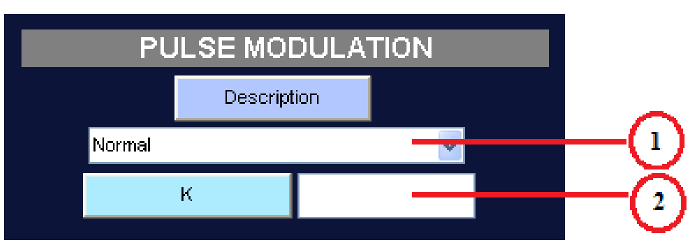


Figure D.2 Labeled view of the Pulse Modulation panel

The following is a detailed explanation of each of the parameters:

1. Pulse modulation function: This section allows the user to create non-rectangular pulses such as: Linear increase, Linear decrease, Exponential increase, Exponential decrease, Gaussian and Sinusoidal functions.

2. K – Value: When a non-rectangular function is chosen this value is multiplied with the pulse amplitude and the product becomes the final amplitude of the pulse signal. When the 'normal' option is chosen from the drop down menu, this option is automatically deactivated.

Detailed guide to using signal modulation parameters

Figure D.3 is a snap shot of the Signal modulation panel of NeuralSTA.

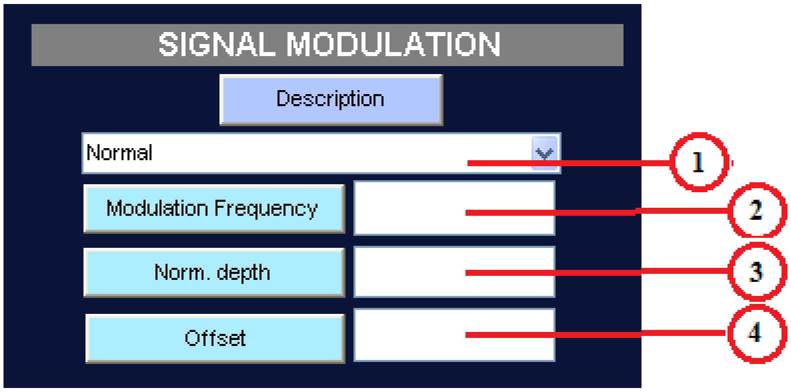


Figure D.3 Labeled view of the Signal Modulation panel

The following is a detailed explanation of each of the parameters:

1. Signal modulation function: This section allows the user to create a non-rectangular train using functions such as: Cosine, Sawtooth, Sine and Square

2. Modulation frequency: This option determines the frequency of the function chosen. If the "normal" option is chosen then this frequency is deactivated. Unit: Hertz.

3. Norm. (normal) Depth: This value determines amplitude of highest (anodic) region of the function (peak of the wave). If the "normal" option is chosen from the drop-down menu, then this value is deactivated. Unit: Volt.

4. Offset: This option determines offset of the function. This value has to be positive. If the "normal" option is chosen then this value is deactivated. Unit: Volt

<u>Detailed guide to using Frame parameters</u>

Figure D.4 is a snap shot of the Frame parameter panel of NeuralSTA.

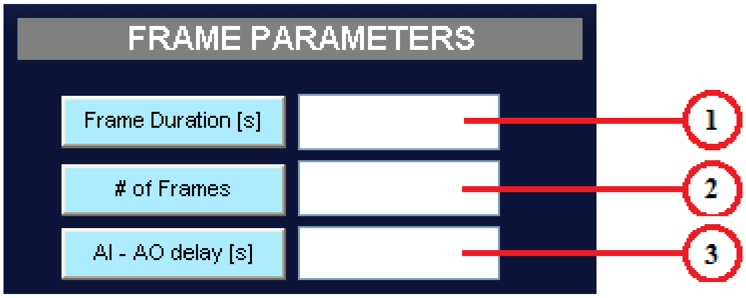


Figure D.4 Labeled view of the Frame parameter panel

The following is a detailed explanation of each of the parameters:

1. Frame Duration: Total time for the duration of the frame. This value entails time for a continuous stimulation. Unit: Seconds

2. # Of Frames: Number of times for consecutive and continuous stimulation of one frame. The value entered must be a positive whole number.

3. AI-AO delay: Delay between the start of the Analog Input channels (recording) and the start of the Analog Output channels (stimulation). Enter "0" for simultaneous recording and stimulation. Unit: Seconds

<u>Detailed guide to using frame by frame modulation</u>

Figure D.5 is a snap shot of the frame by frame modulation panel under Frame parameter panel of NeuralSTA.

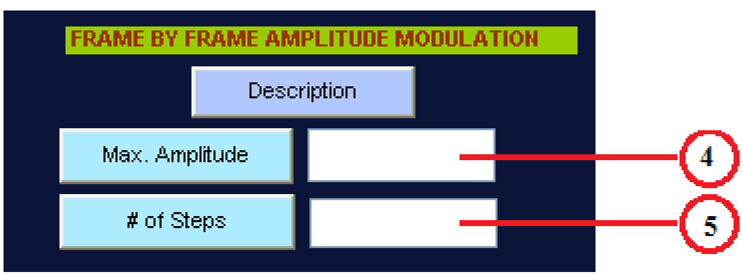


Figure D.5 Labeled view of the Frame by frame amplitude modulation panel

The following is a detailed explanation of each of the parameters:

4. Max. Amplitude: Amplitude higher than the minimum amplitude. This section is deactivated when either 0 or no value is entered. Unit: Volt.

5. # Of steps: This determines the increment of the amplitude from the specified minimum amplitude and the maximum amplitude.

Note: Please refer Appendices A and B for a more detailed description of this section.

Detailed guide to using recording parameters

Figure D.6 is a snap shot of the recording parameters panel of the of NeuralSTA.

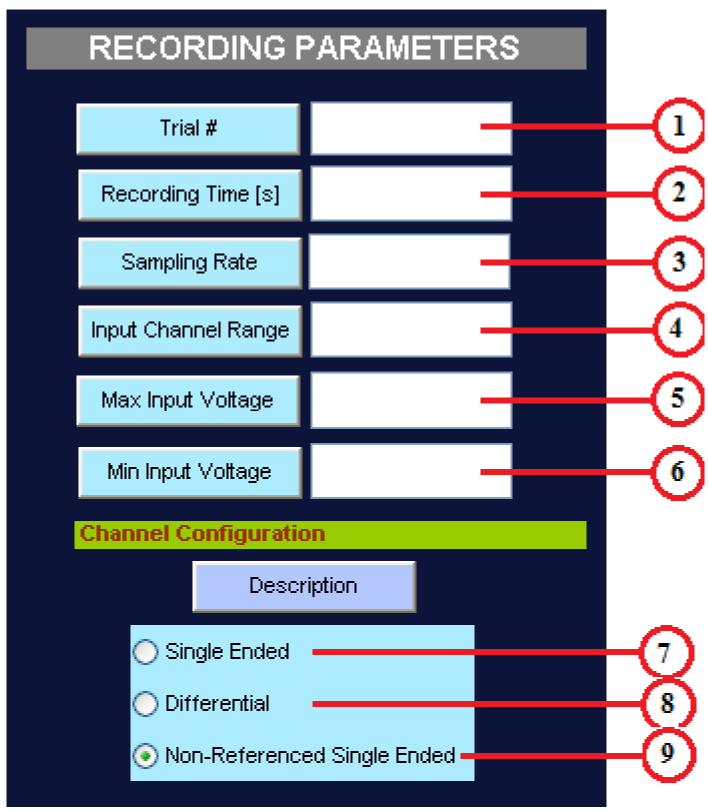


Figure D.6 Labeled view of the Recording parameter panel

The following is a detailed explanation of each of the parameters:

1. Trial #: Trial number is an identifier that can either contain numbers or alphabets or a combination of the two. At the end of the stimulation period,

the data recorded will be saved as: "trial(specified trial #).mat" arbitrarily in the current folder.

2. Recording Time: Total recording time for the Analog Input channel(s). Unit: Seconds.

3. Sampling Rate: Sampling rate for the Analog Input channel(s) Unit: samples/second

4. Input Channel Range: Analog Input channel number(s)

5. Max. Input Voltage: Maximum voltage expected for the Analog Input channel(s). Unit: Volt.

6. Min. Input Voltage: Minimum voltage expected for the Analog Input channel(s). Unit: Volt.
7. Single ended: Analog input channel configuration. Channels are configured for single-ended input.

8. Differential: Analog input channel configuration. Channels are configured for differential input.

9. Non-referenced single ended: Analog input channel configuration. This channel configuration is used when the input signal has its own ground reference, which is tied to the common negative inputs of the instrumentation amplifiers for all channels.

Detailed guide to using Plotting options from the live plot section

Figure D.7 is a snap shot of the plotting options panel under the live plot panel of NeuralSTA.

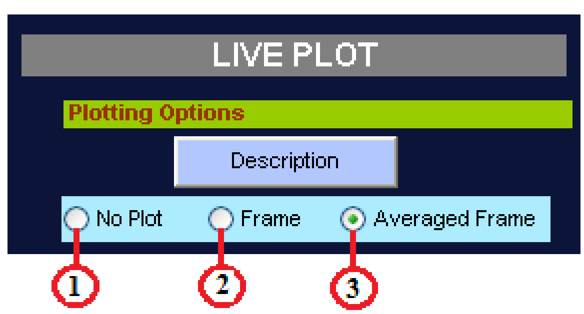


Figure D.7 Labeled view of the “Live plot” panel

These plotting options are active during the time of acquisition. The following is a detailed explanation of each of the parameters:

1. No Plot: This option does not create a plot of the data acquired after each frame. If this option is chosen, the "Plotting order" panel is disabled.

2. Frame: This option plots the data acquired from each of the input channels after each frame. The data displayed in the plot changes after each frame.

3. Averaged Frame: In this option, at the end of every frame, the data collected is averaged with the data collected from the previous frames. The data displayed in the plot changes after each frame.

Detailed guide to using Plotting order from the live plot section

Figure D.8 is a snap shot of the plotting order panel under the live plot panel of NeuralSTA:

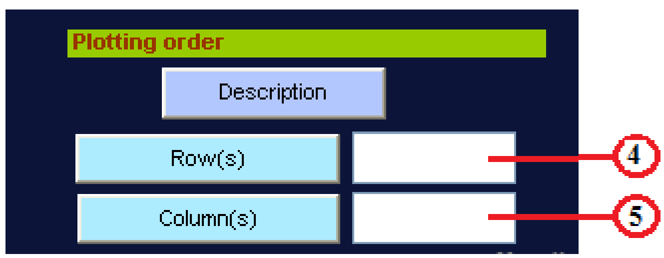


Figure D.8 Labeled view of the plotting order panel

This panel is active when the ‘frame’ or the ‘averaged frame’ option is active. It allows the user to define how the data from each of the input channels (after a frame) will be displayed on a new window. The data from each channel is displayed in a separate graph. The graph for each of the channels can be displayed in a (m x n) format where ‘m’ is the number of rows and ‘n’ is the number of columns. The following is a detailed explanation for each parameter:

4. Row(s): number of channels to be displayed in a row

5. Columns(s): number of channels to be displayed in a column

Detailed guide to using Start and stop buttons

After creating the stimulation signal, the user can trigger the stimulation by clicking on the start button.

Figure D.9 Start button

In case the user needs to stop the stimulation as it is progressing, the 'stop' check - box can be used to arbitrarily stop the stimulation. The stimulation and recordings stop at the frame that the check box is checked. After stopping the stimulation the check box clears.

Figure D.10 Stop button

APPENDIX E

MATLAB SOURCE CODES FOR NeuralSTA

NeuralSTA has been created using Matlab GUIDE. The buttons and the panels in the interface have been derived from the inbuilt functions provided by Mathworks. In addition to the two main files for the interface, the code also includes three additional functions and an image file that are required for the proper functioning of NeuralSTA.

NeuralSTA program includes the following files:

1. NeuralSTA.fig
2. NeuralSTA.m
3. NSTA_modulation_func.m
4. NSTA_error_check_func.m
5. NSTA_live_plot_func.m
6. NSTA.png
7. NSTA_default.mat

The first file is an image file that is required by the software program to load properly. The following are Matlab source codes for the rest of the files.

<u>Front panel of NeuralSTA.fig:</u>

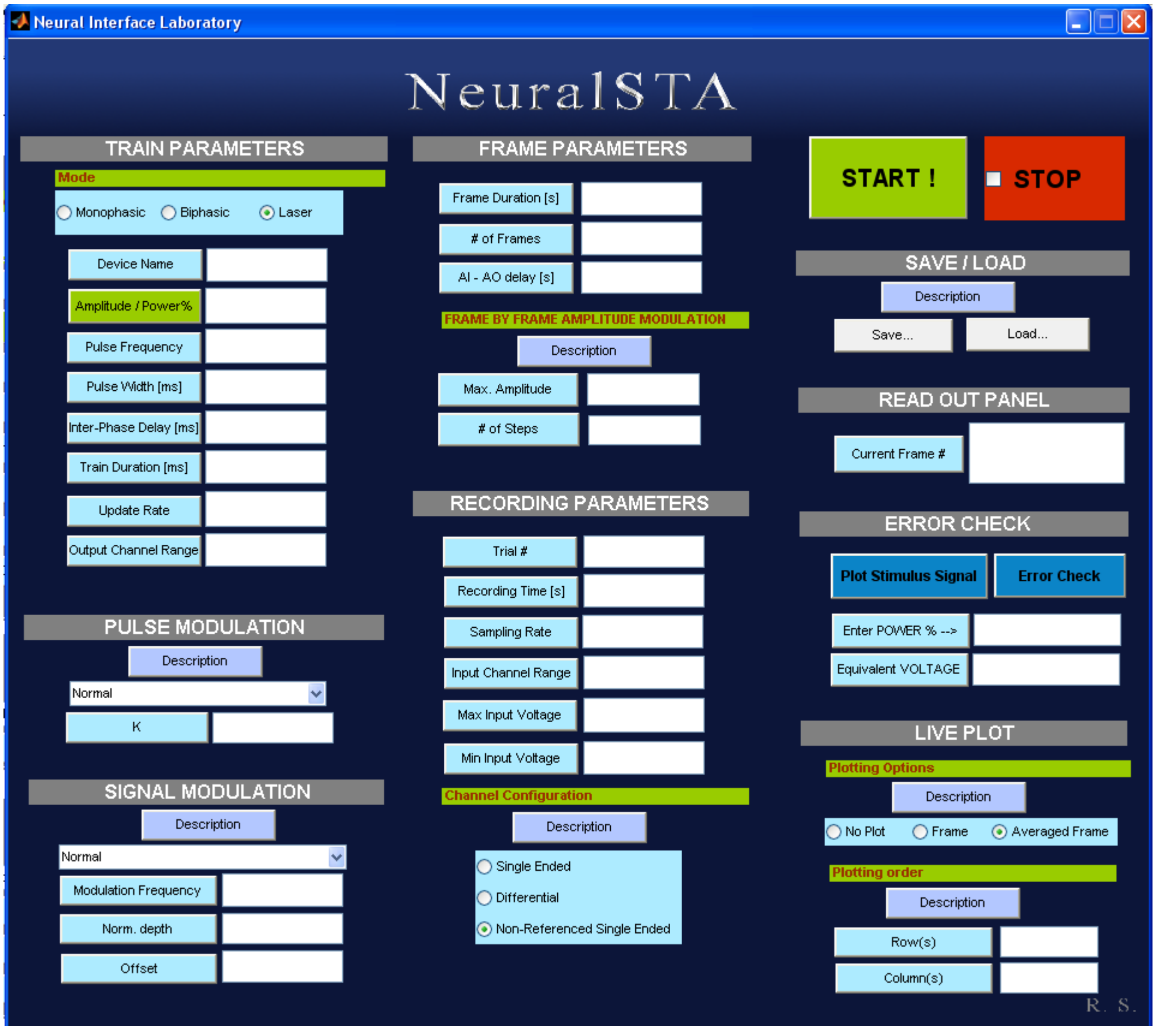


Figure E.1 NeuralSTA front panel

Figure E.1 displays the front panel that was built using the GUIDE feature of Matlab. The tag properties of each of the edit boxes were changed to the following:

1. Train parameters
    1.1. Monophasic → phasic_mono
    1.2. Biphasic → phasic_bi
    1.3. Laser →phasic_laser
    1.4. Device name → devnam
    1.5. Amplitude / power% → amplitude
    1.6. Pulse frequency →frequency
    1.7. Pulse width → pulse_width
    1.8. Inter-phase delay → interpulse
    1.9. Train duration → time_train
    1.10. Update rate → update_rate

1.11. Output channel range → channel_range

2. Pulse modulation
    2.1. Description → pushbutton36
    2.2. Modulation function → pulse_mod
    2.3. K → K_yo
3. Signal mosulation
    3.1. Description → pushbutton32
    3.2. Modulation function → sig_funct
    3.3. Modulation frequency → freq_wav
    3.4. Norm. depth → amplitude_function
    3.5. Offset → amplitude_modulation
4. Frame modulation
    4.1. Frame duration → frame_time
    4.2. # of frames → no_frames
    4.3. AI-AO delay → aIoDelay
    4.4. Description → pushbutton40
    4.5. Max. amplitude → max_amp_mod
    4.6. # of steps → step_size
5. Recording parameters
    5.1. Trial # → trial_no
    5.2. Recording time →time_recording
    5.3. Sampling rate →samplingRate_recording
    5.4. Input channel range →channelRange_recording
    5.5. Max input voltage →max_voltage
    5.6. Min input voltage →min_voltage
    5.7. Description → pushbutton28
    5.8. Single ended → config_single
    5.9. Differential →config_diff
    5.10. Non-referenced single ended → config_nonref
6. Save / Load
    6.1. Save… → pushbutton47
    6.2. Load… →pushbutton48
7. Read out panel
    7.1. Current frame # → curry_frame
8. Error check
    8.1. Plot stimulus signal → pushbutton31
    8.2. Error check → pushbutton46
    8.3. Enter POWER % → anyPower
    8.4. Equivalent VOLTAGE → Laser_volt
9. Live plot
    9.1. No plot → none_plot
    9.2. Frame → norm_plot
    9.3. Averaged frame → avgg_plot
    9.4. Row(s) → row_plott
    9.5. Column(s) → column_plott

Additional coding under the push buttons next to the parameter edit-boxes can be accessed by typing ‘guide’ on the command window and then opening the .fig file. The code to each push button can be accessed by right clicking and then clicking on the callback option.

<u>Source codes for NeuralSTA.m:</u>

```
function varargout = NeuralSTA(varargin)
% NEURALSTA M-file for NeuralSTA.fig

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~
% ---------------------------------------------------------------------
---------
% NeuralSTA - A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING
APPLICATIONS
%                              WITH LASER CONTROL.
% ---------------------------------------------------------------------
---------
% % Supporting files:
%                          NeuralSTA.fig
%                          NSTA_modulation_func.m
%                          NSTA_error_check_func.m
%                          NSTA_live_plot_func.m
%                          NSTA.png
%                          NSTA_default.mat
%
% % Note: ~ Please have all supporting files at the current directory
%        ~ NSTA_default.mat is optional

% Rimi Sahu - rs334@njit.edu
% Dr. Mesut Sahin - mesut.sahin@njit.edu
% NEURAL INTERFACE LABORATORY
% New Jersey Institute of Technology, Newark, NJ - 07102
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~


gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @NeuralSTA_OpeningFcn, ...
                   'gui_OutputFcn',  @NeuralSTA_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

```
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before NeuralSTA is made visible.
function NeuralSTA_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~
%load the background image into Matlab
%if image is not in the same directory as the GUI files, you must use
the
%full path name of the iamge file
LabLogo = importdata('NSTA.png');
%select the axes
axes(handles.axes5);
%place image onto the axes
image(LabLogo);
%remove the axis tick marks
axis off


% --- Outputs from this function are returned to the command line.
function varargout = NeuralSTA_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~
~~%
% PULSE/TRAIN PARAMETERS

meena_on=(get(handles.devnam,'string'));       %DAQ name
fr=str2num(get(handles.frequency,'string'));          %frequency
pw=str2num(get(handles.pulse_width,'string'));        %time of the
first part of the Pulse (ms)
t1=str2num(get(handles.time_train,'string'));         %time of the
Train (ms)
```

```
oww=str2num(get(handles.interpulse,'string'));         %time of the
interpulse delay (ms)
sr1=str2num(get(handles.update_rate,'string'));        %sampling rate
for the Pulse, Train, Frame
am1=str2num(get(handles.amplitude,'string'));          %amplitude of the
pulse
cr1=str2num(get(handles.channel_range,'string'));      %Channel Range of
the analog output
phasicm=get(handles.phasic_mono,'value');              %choose
monophasic
phasicb=get(handles.phasic_bi,'value');                %choose biphasic
phasicl=get(handles.phasic_laser,'value');             %choose laser
tnumber=t1/1000;
pul_val=get(handles.pulse_mod,'Value');

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~
~~%
% FRAME PARAMETERS
t2=str2num(get(handles.frame_time,'string'));           %time of the
Frame (s)
r=str2num(get(handles.no_frames,'string'));             %# of frames
delayAIao=str2num(get(handles.aIoDelay,'string'));      %start ai first
then ao. this give the delay time in [ms]

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~
~~%
% LIVE PLOTTING PARAMETERS
normal_option=get(handles.norm_plot,'Value');
avg_option=get(handles.avgg_plot,'Value');
none_option=get(handles.none_plot,'Value');

row_numb=str2num(get(handles.row_plott,'string'));
column_numb=str2num(get(handles.column_plott,'string'));

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~
~~%
% SIGNAL MODULATION PARAMETERS
sig_val=get(handles.sig_funct,'Value');
amf=str2num(get(handles.amplitude_function,'string'));
%function amplitude
amm=str2num(get(handles.amplitude_modulation,'string'));
%modulation amplitude
ff=str2num(get(handles.freq_wav,'string'));
%modulation frequency

switch get(handles.sig_funct,'Value')
    case 1                  % ---> normal
        cchoice=1;
    case 2                  % ---> cosine
        cchoice=2;
    case 3                  % ---> sawtooth
        cchoice=3;
    case 4                  % ---> sine
        cchoice=4;
    case 5                  % ---> square
        cchoice=5;
```

```
end
%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% PULSE MODULATION
KK_yo=str2num(get(handles.K_yo,'string'));

switch get(handles.pulse_mod,'Value')
    case 1
        funt_choice=1;
    case 2
        funt_choice=2;
    case 3
        funt_choice=3;
    case 4
        funt_choice=4;
    case 5
        funt_choice=5;
    case 6
        funt_choice=6;
    case 7
        funt_choice=7;
    case 8
        funt_choice=8;
end

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% RECORDING PARAMETERS
tn=(get(handles.trial_no,'string'));            %trial #
t_RP=str2num(get(handles.time_recording,'string'));    %recording time
sr2=str2num(get(handles.samplingRate_recording,'string'));
%sampling rate
MaV=str2num(get(handles.max_voltage,'string')) ;       %Maximum Voltage
Range
MiV=str2num(get(handles.min_voltage,'string'));        %Minimum Voltage
Range
cr2=str2num(get(handles.channelRange_recording,'string'));     %Channel
Range of the analog input

configS=get(handles.config_single,'value');
configD=get(handles.config_diff,'value');
configN=get(handles.config_nonref,'value');

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% AMPLITUDE MODULATION PARAMETERS

stepp=str2num(get(handles.step_size,'string'));          %step size
max_amp=str2num(get(handles.max_amp_mod,'string'));      % max level of
amplitude

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% ERROR CHECK
```

```
[main_msg tot]=NSTA_error_check_func(am1, KK_yo, oww, pw, t1,t2, MaV,
MiV, tn, t_RP, fr, amf, amm, ff, normal_option, avg_option, row_numb,
column_numb,r, meena_on, cchoice, phasicb);

if isempty(tot)
    tot=sprintf('No Errors!\n\nReady to Go!');
else
    final_msg=[main_msg tot];
    uiwait(msgbox(final_msg,'List of Parameter Errors', 'error'));
    return;
end

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%

if configS==1
    CC=['SingleEnded'];
else if configD==1
        CC=['Differential'];
    else if configN==1
            CC=['NonReferencedSingleEnded'];
        end
    end
end

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% WAVE CREATION

[w x train_mod sig_mod func_mod,max_amp]=
NSTA_modulation_func(fr,pw,t1,sr1,am1,phasicm,phasicb,phasicl,oww,t2,r,
amf,amm,ff,cchoice,funt_choice,KK_yo,max_amp,stepp);

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% ANALOG INPUT

tic
ai = analoginput('nidaq',meena_on);
ai.ClockSource = 'Internal';
ai.BufferingMode = 'Auto';
ai.InputType = CC;
addchannel(ai,cr2);
ActualRange = setverify(ai.Channel,'InputRange',[MiV MaV]);
duration = t_RP;
ActualRate = setverify(ai,'SampleRate',sr2);
set(ai,'SamplesPerTrigger',duration*ActualRate);
ai.ChannelskewMode = 'Equisample';


%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% ANALOG OUTPUT
ao = analogoutput('nidaq',meena_on);
addchannel(ao,cr1);
```

```
set(ao, 'SampleRate', sr1);

if phasicb==1
    ActualRange = setverify(ao.Channel,'OutputRange',[-10 10]);
    ActualRange = setverify(ao.Channel,'UnitsRange',[-10 10]);
end

%new addition 04.07.11
set([ai ao],'TriggerType','Manual')
ai.ManualTriggerHwOn = 'Trigger';


% some extra variables
xxx_inf=1;
x3_inf=1;

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% TRIGGERING

for rr=1:r
% Set Analog Output Parameters
if (~isempty(daqfind))
    stop(daqfind)
end

shouldi=get(handles.breakkk,'value');
if shouldi==1
    break
end

%update the current frame number
set(handles.curry_frame,'string',num2str(rr));

%Start outputting data
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
trial = tn;
no=num2str(trial);
nam = ['trial' no '.mat'];

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% WAVE CREATION

%LIVE PLOTTING OPTION

ab=0;
%Start recording data
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
W=(w(rr,:))';
putdata(ao, W);
start([ai,ao])

trigger(ai)
```

```
% %delay between ai and ao
pause(delayAIao);

trigger(ao)

%Add sound
beep
%End sound
wait([ai,ao],t2+5)

data_trigger_time = ai.InitialTriggerTime;
[data2, time] = getdata(ai);

stop(ai);
%clear daqfile ai.LogFileName ai.LoggingMode
pause(tnumber);
% delete(ao)

if rr==1
    data=data2;
    data4=data2*0;
    data5=data2*0;
    zx_index=1;
else
data=vertcat(data,data2);
end

%plotting option statement
if normal_option==1
    data3=data2;
elseif avg_option==1
    data4=data4+data2;
    data3=data4/zx_index;
    zx_index=zx_index+1;
end

%LIVE PLOTTING FUNCTION
if xxx_inf&&(normal_option||avg_option==1)==1
    a=figure('Color',[0.8 1 0],'name','Channel Plots',
'numbertitle','off');
    xxx_inf=xxx_inf-1;
end

if normal_option||avg_option==1
   NSTA_live_plot_func(data3,row_numb,
column_numb,cr2,max_amp,sr2,rr,w,x,phasic1)
end

end

toc
%to save the recorded data
save (nam, 'data');
%to save parameter file
```

```
nox=num2str(trial);
namx = ['trial_' nox '_Parameters' '.mat'];

stepp=str2num(get(handles.step_size,'string'));          %step size
max_amp=str2num(get(handles.max_amp_mod,'string'));      % max level of
amplitude

save (namx,'meena_on','fr', 'pw', 't1', 'oww', 'sr1', 'am1', 'cr1',
'phasicm', 'phasicb', 'phasicl', 't2', 'r', 'KK_yo', 'amf', 'amm',
'ff', 'tn', 't_RP', 'sr2', 'MaV', 'MiV', 'cr2', 'max_amp', 'stepp',
'normal_option', 'avg_option', 'row_numb', 'column_numb', 'delayAIao',
'configS', 'configD', 'configN','none_option','pul_val','sig_val');

clear ai;
set(handles.breakkk,'value',0);


% --- Executes on button press in phasic_mono.
function phasic_mono_Callback(hObject, eventdata, handles)

% --- Executes on button press in phasic_bi.
function phasic_bi_Callback(hObject, eventdata, handles)

% --- Executes on button press in phasic_laser.
function phasic_laser_Callback(hObject, eventdata, handles)

% --- Executes on button press in checkbox2.
function checkbox2_Callback(hObject, eventdata, handles)

% --- Executes on button press in checkbox3.
function checkbox3_Callback(hObject, eventdata, handles)

% --- Executes on button press in checkbox4.
function checkbox4_Callback(hObject, eventdata, handles)

function frequency_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function frequency_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


function update_rate_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function update_rate_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end


function pulse_width_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function pulse_width_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function amplitude_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function amplitude_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function frame_time_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function frame_time_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function no_frames_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function no_frames_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function time_train_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function time_train_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function channel_range_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function channel_range_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function trial_no_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function trial_no_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function time_recording_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function time_recording_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function samplingRate_recording_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function samplingRate_recording_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit18_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit18_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function max_voltage_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function max_voltage_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function min_voltage_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function min_voltage_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit21_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit21_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit22_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit22_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in config_single.
function config_single_Callback(hObject, eventdata, handles)

% --- Executes on button press in config_diff.
function config_diff_Callback(hObject, eventdata, handles)

% --- Executes on button press in config_nonref.
function config_nonref_Callback(hObject, eventdata, handles)


function channelRange_recording_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
```

```
function channelRange_recording_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on key press with focus on load and none of its
controls.
function load_KeyPressFcn(hObject, eventdata, handles)


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
text12.
function text12_ButtonDownFcn(hObject, eventdata, handles)

% --- Executes on button press in breakkk.
function breakkk_Callback(hObject, eventdata, handles)

% --- Executes on slider movement.
function slider2_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function slider2_CreateFcn(hObject, eventdata, handles)

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end

% --- Executes on selection change in sig_funct.
function sig_funct_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function sig_funct_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Lamp_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Lamp_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit29_Callback(hObject, eventdata, handles)
```

```matlab
% --- Executes during object creation, after setting all properties.
function edit29_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit30_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function edit30_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in pulse_mod.
function pulse_mod_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function pulse_mod_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function K_yo_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function K_yo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function freq_wav_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function freq_wav_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function amplitude_modulation_Callback(hObject, eventdata, handles)
```

```
% --- Executes during object creation, after setting all properties.
function amplitude_modulation_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function interpulse_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function interpulse_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function amplitude_function_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function amplitude_function_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function max_amp_mod_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function max_amp_mod_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function step_size_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function step_size_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in norm_plot1.
```

```
function norm_plot1_Callback(hObject, eventdata, handles)

% --- Executes on button press in avgg_plot1.
function avgg_plot1_Callback(hObject, eventdata, handles)

function row_plott_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function row_plott_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function column_plott_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function column_plott_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in las_vol.
function las_vol_Callback(hObject, eventdata, handles)

pow_val=str2num(get(handles.anyPower,'string'));
llxl=pow_val;
llyl=(113.4-llxl)/25.39;
llrl=.001;
llryl=llrl*round(llyl/llrl);
set(handles.Laser_volt,'String',llryl);

function Laser_volt_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function Laser_volt_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function curry_frame_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function curry_frame_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function aIoDelay_Callback(hObject, eventdata, handles)


% --- Executes during object creation, after setting all properties.
function aIoDelay_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in none_plot.
function none_plot_Callback(hObject, eventdata, handles)

% --- Executes on button press in record_CR.
function record_CR_Callback(hObject, eventdata, handles)

message_record_CR = sprintf('Enter the Analog Input channel number(s)
in the format below: \n\n  "0 1 2"   or   "0:2"\n ');
uiwait(msgbox(message_record_CR));


% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)

message_pushbutton12 = sprintf('Enter the trial number. This identifier
can either contain *numbers* or *alphabets* or a combination of the
two. \n\nThe following symbols can be included: \n(dash)   -
\n(underscore)   _\n(period)   .\n\nThe following should not be used in
the value entered:\n(colon)   :\n(semi colon)   ;\n(comma)
,\n(backslash, front slash)\n(exclamation)  !, @, #....etc\n\n All the
data recorded will be saved as: "trial(specified value).mat"
\n\nAdditionally, the parameters used during the acquisition will be
saved as trial_(specified value).mat\n\nExample:\n\nSpecified value
(value entered):101\nUsing this value, the two files generated are:\n1:
trial101.mat\n2:   trial_101_parameters.mat');
uiwait(msgbox(message_pushbutton12,'Trial Number description',
'help'));


% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
```

```
message_pushbutton13 = sprintf('Enter the total recording time for the
channel(s)\n\n Unit: seconds ');
uiwait(msgbox(message_pushbutton13,'Recording Time Description',
'help'));


% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)

message_pushbutton14 = sprintf('Enter the sampling rate for the Analog
Input channel(s)\n\n Unit: samples/second ');
uiwait(msgbox(message_pushbutton14,'AI Channel Description', 'help'));


% --- Executes on button press in pushbutton15.
function pushbutton15_Callback(hObject, eventdata, handles)
message_pushbutton15 = sprintf('Enter the maximum voltage expected for
the input channel(s)\n\n Unit: Volt');
uiwait(msgbox(message_pushbutton15,'AI Max Voltage Description',
'help'));

% --- Executes on button press in pushbutton16.
function pushbutton16_Callback(hObject, eventdata, handles)
message_pushbutton16 = sprintf('Enter the minimum voltage expected for
the input channel(s)\n\n Unit: Volt');
uiwait(msgbox(message_pushbutton16,'AI Min Voltage Description',
'help'));


% --- Executes on button press in pushbutton17.
function pushbutton17_Callback(hObject, eventdata, handles)
message_pushbutton17 = sprintf('This box will display the current frame
number as the code is being executed');
uiwait(msgbox(message_pushbutton17,'Frame Number Display Description',
'help'));


% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
message_pushbutton18 = sprintf('Enter the amount of delay you would
like to create between the start of the Analog Input channels and the
start of the Analog Output channels\n\n Enter "0" if you want both AI
and AO to start simultaneously \n\n Unit: Seconds ');
uiwait(msgbox(message_pushbutton18,'AI - AO Delay Description',
'help'));


% --- Executes on button press in pushbutton19.
function pushbutton19_Callback(hObject, eventdata, handles)
message_pushbutton19 = sprintf('Enter the number of times you would
like stimulate\n\n Note: the value entered must be a whole number');
uiwait(msgbox(message_pushbutton19,'Number of Frames Description',
'help'));
```

```
% --- Executes on button press in pushbutton20.
function pushbutton20_Callback(hObject, eventdata, handles)
message_pushbutton20 = sprintf('Enter the total time for the duration
of the frame\n\nThis value entails time for one continuous stimulation
and/or recording time\n\nUnit: Seconds');
uiwait(msgbox(message_pushbutton20,'Frame Time Description', 'help'));


% --- Executes on button press in pushbutton21.
function pushbutton21_Callback(hObject, eventdata, handles)
message_pushbutton21 = sprintf('Enter the Analog Output channel
number(s) in the format below: \n\n  "0 1 2"   or   "0:2"\n ');
uiwait(msgbox(message_pushbutton21,'AO Channel Description', 'help'));


% --- Executes on button press in pushbutton22.
function pushbutton22_Callback(hObject, eventdata, handles)
message_pushbutton22 = sprintf('Enter the sampling rate for the Analog
Output channel(s)\n\n Unit: samples/second ');
uiwait(msgbox(message_pushbutton22,'AO Channel Sampling Rate
Description', 'help'));


% --- Executes on button press in pushbutton23.
function pushbutton23_Callback(hObject, eventdata, handles)
message_pushbutton23 = sprintf('Enter the total time for the duration
of the train\n\nThis value entails time for one continuous
signal(monophasic, biphasic or laser mode)\n\nUnit: Millisecond');
uiwait(msgbox(message_pushbutton23,'AO Train Time Description',
'help'));


% --- Executes on button press in pushbutton24.
function pushbutton24_Callback(hObject, eventdata, handles)
message_pushbutton24 = sprintf('For the monophasic mode:this value is
the delay between two anodic pulses\n\nFor the biphasic mode: the value
is the delay between the anodic pulse and the following cathodic
pulse\n\nFor the laser mode: this value is the delay between two
pulses\n\nUnit: Millisecond');
uiwait(msgbox(message_pushbutton24,'AO Train Delay Description',
'help'));


% --- Executes on button press in pushbutton25.
function pushbutton25_Callback(hObject, eventdata, handles)
message_pushbutton25 = sprintf('This value is the time for the first
part of the pulse\n\nThe first part of the pulse is defined
as:\nMonophasic & Biphasic Mode   ---> Anodic part\nLaser mode
----> Initial 5V anodic part\n\nUnit: Millisecond');
uiwait(msgbox(message_pushbutton25,'AO Pulse Width Description',
'help'));
```

```
% --- Executes on button press in pushbutton26.
function pushbutton26_Callback(hObject, eventdata, handles)
message_pushbutton26 = sprintf('This value is the frequency of each
pulse in the train\n\nUnit: Hertz');
uiwait(msgbox(message_pushbutton26,'AO Pulse Frequency Description',
'help'));


% --- Executes on button press in pushbutton27.
function pushbutton27_Callback(hObject, eventdata, handles)
message_pushbutton27 = sprintf('For Monophasic & Biphasic modes: \n-
This value is the amplitude of the anodic part of the signal\n\nFor
Laser mode:\n-This value is the amplitude of the last part of the
pulse\n\nUnit: Volt (for monophasic and biphasic mode)');
uiwait(msgbox(message_pushbutton27,'AO Amplitude / Percent power
Description', 'help'));


% --- Executes on button press in pushbutton28.
function pushbutton28_Callback(hObject, eventdata, handles)
message_pushbutton28 = sprintf('Specify analog input hardware channel
configuration\n\nFor National Instruments devices, InputType can be the
following:\n\nDifferential---> Channels are configured for
differential input\n\nSingleEnded---> Channels are configured for
single-ended input\n\nNonReferencedSingleEnded---> This channel
configuration is used when the input signal has its own ground
reference, which is tied to the negative input of the instrumentation
amplifier');
uiwait(msgbox(message_pushbutton28,'AI Channel Config Description',
'help'));


% --- Executes on button press in pushbutton29.
function pushbutton29_Callback(hObject, eventdata, handles)
message_pushbutton29 = sprintf('This option is applicable when the
*frame* or the *averaged frame* option is chosen from the Plotting
Options section\n\nThis option allows the user to define how the data
from each of the input channels will be displayed on the graph\n\nFor
example, if 5 input channels are chosen, the user has the option to
display the data from the channels in a (1x5) or (5x1) or (2x3) or
(3x2)etc format\n\nNote: If the "No Plot" option is choosen, the
"Plotting order" option is disabled');
uiwait(msgbox(message_pushbutton29,'Plotting Order Description',
'help'));


% --- Executes on button press in pushbutton30.
function pushbutton30_Callback(hObject, eventdata, handles)
message_pushbutton30 = sprintf('Plotting Options during the time of
acquisition\n\nNo Plot---> This option does not create a plot of the
data acquired after every frame\n\nFrame---> This option plots the data
acquired from each of the input channels after each frame. The data
displayed in the plot changes after each frame\n\nAveraged Frame---> In
this option, after the end of every frame, the data collected is
averaged with the data collected from the previous frames\n\nNote: If
```

```
the "No Plot" option is choosen, the "Plotting order" option is
disabled');
uiwait(msgbox(message_pushbutton30,'Live Plot Description', 'help'));


% --- Executes on button press in pushbutton31.
function pushbutton31_Callback(hObject, eventdata, handles)

    fr=str2num(get(handles.frequency,'string'));            %frequency
    pw=str2num(get(handles.pulse_width,'string'));          %time of the
first part of the Pulse (ms)
    t1=str2num(get(handles.time_train,'string'));           %time of the
Train (ms)
    oww=str2num(get(handles.interpulse,'string'));          %time of the
interpulse delay (ms)
    sr1=str2num(get(handles.update_rate,'string'));         %sampling
rate for the Pulse, Train, Frame
    am1=str2num(get(handles.amplitude,'string'));           %amplitude of
the pulse
    cr1=str2num(get(handles.channel_range,'string'));       %Channel
Range of the analog output
    phasicm=get(handles.phasic_mono,'value');               %choose
monophasic
    phasicb=get(handles.phasic_bi,'value');                 %choose
biphasic
    phasicl=get(handles.phasic_laser,'value');              %choose laser

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~
~~%
    % FRAME PARAMETERS
    t2=str2num(get(handles.frame_time,'string'));            %time of the
Frame (s)
    r=str2num(get(handles.no_frames,'string'));              %# of frames


%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~
~~%
    % SIGNAL MODULATION PARAMETERS

    amf=str2num(get(handles.amplitude_function,'string'));
%function amplitude
    amm=str2num(get(handles.amplitude_modulation,'string'));
%modulation amplitude
    ff=str2num(get(handles.freq_wav,'string'));
%modulation frequency

    switch get(handles.sig_funct,'Value')
        case 1                  % ---> normal
            cchoice=1;
        case 2                  % ---> cosine
            cchoice=2;
        case 3                  % ---> sawtooth
            cchoice=3;
        case 4                  % ---> sine
            cchoice=4;
        case 5                  % ---> square
```

```
            cchoice=5;
    end

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
    % PULSE MODULATION
    KK_yo=str2num(get(handles.K_yo,'string'));

    switch get(handles.pulse_mod,'Value')
        case 1
            funt_choice=1;
        case 2
            funt_choice=2;
        case 3
            funt_choice=3;
        case 4
            funt_choice=4;
        case 5
            funt_choice=5;
        case 6
            funt_choice=6;
        case 7
            funt_choice=7;
        case 8
            funt_choice=8;
    end

stepp=str2num(get(handles.step_size,'string'));          %step size
max_amp=str2num(get(handles.max_amp_mod,'string'));      % max level of
amplitude

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% error check

%Dummy values:
MaV=1;
MiV=-1;
tn=1;
t_RP=1;
normal_option=1;
avg_option=1;
row_numb=1;
column_numb=1;
meena_on=1;


[main_msg tot]=NSTA_error_check_func(am1, KK_yo, oww, pw, t1,t2, MaV,
MiV, tn, t_RP, fr, amf, amm, ff, normal_option, avg_option, row_numb,
column_numb,r, meena_on, cchoice, phasicb);
if isempty(tot)
    [w x train_mod sig_mod func_mod,max_amp]=
NSTA_modulation_func(fr,pw,t1,sr1,am1,phasicm,phasicb,phasicl,oww,t2,r,
amf,amm,ff,cchoice,funt_choice,KK_yo,max_amp,stepp);
    figure,plot(x,w(1,:),'r');      %plot of the signal
```

```
    xlabel('Time (s)');
    ylabel('Amplitude (V)');

    grid on
else
    final_msg=[main_msg tot];
    uiwait(msgbox(final_msg,'List of Parameter Errors', 'error'));
end


% --- Executes on button press in pushbutton32.
function pushbutton32_Callback(hObject, eventdata, handles)

message_pushbutton32 = sprintf('This option allows the user to modulate
the train (1 frame) as per the specified parameters: \n\nFunction
(sine, cosine...etc)\nFrequency\nAmplitude of the function\nOffset of
the signal\n\nNote: If the "normal" option of the drop down menu is
chosen, then this section is not applicable');
uiwait(msgbox(message_pushbutton32,'Signal Modulation Description',
'help'));


% --- Executes on key press with focus on pushbutton28 and none of its
controls.
function pushbutton28_KeyPressFcn(hObject, eventdata, handles)

% --- Executes on button press in pushbutton33.
function pushbutton33_Callback(hObject, eventdata, handles)
message_pushbutton33 = sprintf('This option determines the frequency of
the function chosen on the drop down menu\n\nNote: If the "normal"
option is chosen then this frequency is not applicable\n\nUnit:
Hertz');
uiwait(msgbox(message_pushbutton33,'Modulation Frequency Description',
'help'));


% --- Executes on button press in pushbutton34.
function pushbutton34_Callback(hObject, eventdata, handles)
message_pushbutton34 = sprintf('This option determines amplitude of
highest region of the function (peak of the wave)\n\nNote: If the
"normal" option is chosen then this value is not applicable\n\nUnit:
Volt');
uiwait(msgbox(message_pushbutton34,'Normal Depth Description',
'help'));


% --- Executes on button press in pushbutton35.
function pushbutton35_Callback(hObject, eventdata, handles)
message_pushbutton35 = sprintf('This option determines offset of the
signal\n\nNote: If the "normal" option is chosen then this value is not
applicable\n\nUnit: Volt');
uiwait(msgbox(message_pushbutton35,'Offset Description', 'help'));
```

```
% --- Executes on button press in pushbutton36.
function pushbutton36_Callback(hObject, eventdata, handles)
message_pushbutton35 = sprintf('This option determines the shape of the
individual pulses in the train. Instead of a normal rectangular wave,
the pulse can be modified using the following functions:\n\nLinear
Increase\nLinear Decrease\nExponential Increase\nExponential
Decrease\nGaussian\nSinusoidal\n\n This section also allows the user to
determine the amplitude of the pulse\n\nNote: If the "normal" option of
the drop down menu is chosen, then this section is not applicable');
uiwait(msgbox(message_pushbutton35,'Pulse Modulation Description',
'help'));


% --- Executes on button press in pushbutton37.
function pushbutton37_Callback(hObject, eventdata, handles)
message_pushbutton37 = sprintf('This option determines the amplitude of
the function used to shape the pulse\n\nThis value is multiplied with
the Amplitude value specified under the "Train Parameter section"
\n\nNote: If the "normal" option is chosen then this value is not
applicable\n\nUnit: Volt');
uiwait(msgbox(message_pushbutton37,'K Description', 'help'));



% --- Executes on button press in pushbutton39.
function pushbutton39_Callback(hObject, eventdata, handles)
message_pushbutton39 = sprintf('This section can be used for two
purposes:\n\n1.-----Save-----------------------------------------------
\n~Click on "Save"\n\n~This operation opens a window where the user can
specify the file path and the title of the parameter file and then
click on the save file\n\nThis action results in a ".fig" file that can
be loaded in future using the "load" button\n\n\n2.-----Load-----------
-----------------------------------\n~The user can load a previously
loaded parameter file by clicking on the load button\n\nA click results
in a window where the user has to specify which file is to be
opened\n\nAfter this action the previous GUI will be closed');
uiwait(msgbox(message_pushbutton39,'Save/Load Description', 'help'));


% --- Executes on button press in pushbutton40.
function pushbutton40_Callback(hObject, eventdata, handles)
message_pushbutton40 = sprintf('This section can be used for modulating
the amplitude of consecutive frames. This can be achieved by using the
following parameters:\n\n1.-----Maximum Amplitude----------------------
------------------------\n~This parameter sets the maximum amplitude
that the frame can reach. The starting amplitude value is determined by
the amplitude under the "Train Parameters" and K under the "Pulse
Modulation section (in some cases)\n\n\n2.-----# of Steps--------------
--------------------------------\n~This determines the increament of
the amplitude from the specified minimum amplitude and the maximum
amplitude\n\n~Example:\n\n****Minimum Amplitude=1\nMaximum
Amplitude=5\n# of Steps=5\n\n*****These parameters will generate a list
of values:1 2 3 4 5. These values will be the (max) amplitudes of the
corresponding frames\n\n*****For the 1st frame the maximum amplitude
```

```
will be 1 volt\n*****For the 2nd frame the maximum amplitude will be 2
volt\n*****For the 3rd frame the maximum amplitude will be 3
volt\n*****For the 4th frame the maximum amplitude will be 4
volt\n*****For the 5th frame the maximum amplitude will be 5
volt\n\n*Note: In order to disable FRAME BY FRAME AMPLITUDE MODULATION,
either enter 0 or leave the box blank for the two parameters');
uiwait(msgbox(message_pushbutton40,'Frame by Frame Amplitude Modulation
Description', 'help'));


% --- Executes on button press in pushbutton41.
function pushbutton41_Callback(hObject, eventdata, handles)
message_pushbutton41 = sprintf('Enter a single numeric value\n\nNote:
This value must be greater than:\n\n [Amplitude (under Train
Parameters)]  x  [K (under Pulse modulation)]\n\nUnit: Volt');
uiwait(msgbox(message_pushbutton41,'Maximum Amplitude Description',
'help'));


% --- Executes on button press in pushbutton42.
function pushbutton42_Callback(hObject, eventdata, handles)
message_pushbutton42 = sprintf('Enter a single numeric value\n\nNote:
This value can be greater than the *# of frames*, however the final
frame may not reach the maximum amplitude');
uiwait(msgbox(message_pushbutton42,'# of Steps Description', 'help'));


% --- Executes on button press in pushbutton43.
function pushbutton43_Callback(hObject, eventdata, handles)
message_pushbutton43 = sprintf('Enter a single numeric value for the
number of rows desired');
uiwait(msgbox(message_pushbutton43,'Row Description', 'help'));


% --- Executes on button press in pushbutton44.
function pushbutton44_Callback(hObject, eventdata, handles)
message_pushbutton44 = sprintf('Enter a single numeric value for the
number of columns desired');
uiwait(msgbox(message_pushbutton44,'Column Description', 'help'));


% --- Executes on button press in pushbutton46.
function pushbutton46_Callback(hObject, eventdata, handles)

am1=str2num(get(handles.amplitude,'string'));         %amplitude of the
pulse
KK_yo=str2num(get(handles.K_yo,'string'));

oww=str2num(get(handles.interpulse,'string'));        %time of the
interpulse delay (ms)
pw=str2num(get(handles.pulse_width,'string'));        %time of the
first part of the Pulse (ms)
t1=str2num(get(handles.time_train,'string'));         %time of the
Train (ms)
```

```
t2=str2num(get(handles.frame_time,'string'));          %time of the
Frame (s)

MaV=str2num(get(handles.max_voltage,'string')) ;       %Maximum Voltage
Range
MiV=str2num(get(handles.min_voltage,'string'));        %Minimum Voltage
Range
tn=(get(handles.trial_no,'string'));             %trial #
t_RP=str2num(get(handles.time_recording,'string'));    %recording time

fr=str2num(get(handles.frequency,'string'));          %frequency

amf=str2num(get(handles.amplitude_function,'string'));         %norm
depth
amm=str2num(get(handles.amplitude_modulation,'string'));       %offset
ff=str2num(get(handles.freq_wav,'string'));
%modulation frequency

normal_option=get(handles.norm_plot,'Value');
avg_option=get(handles.avgg_plot,'Value');
row_numb=str2num(get(handles.row_plott,'string'));
column_numb=str2num(get(handles.column_plott,'string'));

r=str2num(get(handles.no_frames,'string'));            %# of frames
meena_on=(get(handles.devnam,'string'));       %DAQ name

switch get(handles.sig_funct,'Value')
    case 1                 % ---> normal
        cchoice=1;
    case 2                 % ---> cosine
        cchoice=2;
    case 3                 % ---> sawtooth
        cchoice=3;
    case 4                 % ---> sine
        cchoice=4;
    case 5                 % ---> square
        cchoice=5;
end
phasicb=get(handles.phasic_bi,'value');               %choose biphasic

[main_msg tot]=NSTA_error_check_func(am1, KK_yo, oww, pw, t1,t2, MaV,
MiV, tn, t_RP, fr, amf, amm, ff, normal_option, avg_option, row_numb,
column_numb,r, meena_on, cchoice, phasicb);

if isempty(tot)
    tot=sprintf('No Errors!\n\nReady to Go!');
end

final_msg=[main_msg tot];

uiwait(msgbox(final_msg,'List of Parameter Errors', 'error'));


% --- Executes on button press in pushbutton47.
```

```
function pushbutton47_Callback(hObject, eventdata, handles)

% PULSE PARAMETERS
meena_on=(get(handles.devnam,'string'));         %DAQ name
fr=str2num(get(handles.frequency,'string'));            %frequency
pw=str2num(get(handles.pulse_width,'string'));          %time of the
first part of the Pulse (ms)
t1=str2num(get(handles.time_train,'string'));           %time of the
Train (ms)
oww=str2num(get(handles.interpulse,'string'));           %time of the
interpulse delay (ms)
sr1=str2num(get(handles.update_rate,'string'));         %sampling rate
for the Pulse, Train, Frame
am1=str2num(get(handles.amplitude,'string'));           %amplitude of the
pulse
cr1=str2num(get(handles.channel_range,'string'));       %Channel Range of
the analog output
phasicm=get(handles.phasic_mono,'value');               %choose
monophasic
phasicb=get(handles.phasic_bi,'value');                 %choose biphasic
phasicl=get(handles.phasic_laser,'value');              %choose laser

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% FRAME PARAMETERS
t2=str2num(get(handles.frame_time,'string'));                    %time of
the Frame (s)
r=str2num(get(handles.no_frames,'string'));
delayAIao=get(handles.aIoDelay,'string');               %get the delay
betw. ai and ao

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% PULSE MODULATION PARAMETERS
KK_yo=str2num(get(handles.K_yo,'string'));
pul_val=get(handles.pulse_mod,'Value');

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% SIGNAL MODULATION PARAMETERS
sig_val=get(handles.sig_funct,'Value');                          %signal
modulation drop down menu
amf=str2num(get(handles.amplitude_function,'string'));
%function amplitude
amm=str2num(get(handles.amplitude_modulation,'string'));
%modulation amplitude
ff=str2num(get(handles.freq_wav,'string'));
%modulation frequency
%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% AMPLITUDE MODULATION PARAMETERS
stepp=str2num(get(handles.step_size,'string'));            %step size
max_amp=str2num(get(handles.max_amp_mod,'string'));        % max level of
amplitude
```

```
%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% RECORDING PARAMETERS
tn=(get(handles.trial_no,'string'));            %trial #
t_RP=str2num(get(handles.time_recording,'string'));    %recording time
sr2=str2num(get(handles.samplingRate_recording,'string'));
%sampling rate
MaV=str2num(get(handles.max_voltage,'string')) ;        %Maximum Voltage
Range
MiV=str2num(get(handles.min_voltage,'string'));         %Minimum Voltage
Range
cr2=str2num(get(handles.channelRange_recording,'string'));     %Channel
Range of the analog input

configS=get(handles.config_single,'value');
configD=get(handles.config_diff,'value');
configN=get(handles.config_nonref,'value');

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% LIVE PLOTTING PARAMETERS
normal_option=get(handles.norm_plot,'Value');
avg_option=get(handles.avgg_plot,'Value');
none_option=get(handles.none_plot,'Value');

row_numb=str2num(get(handles.row_plott,'string'));
column_numb=str2num(get(handles.column_plott,'string'));

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%

[filename,pathname] = uiputfile('*.mat','Save Current Parameters As
....');
if pathname == 0
    return
end
save (filename,'meena_on','fr', 'pw', 't1', 'oww', 'sr1', 'am1', 'cr1',
'phasicm', 'phasicb', 'phasicl', 't2', 'r', 'KK_yo', 'amf', 'amm',
'ff', 'tn', 't_RP', 'sr2', 'MaV', 'MiV', 'cr2', 'max_amp', 'stepp',
'normal_option', 'avg_option', 'row_numb', 'column_numb', 'delayAIao',
'configS', 'configD', 'configN','none_option','pul_val','sig_val');

% --- Executes on button press in pushbutton48.
function pushbutton48_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton48 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%allow the user to choose which settings to load
[filename, pathname] = uigetfile('*.mat', 'Load Parameter file...');
if pathname == 0
    return
end
load(filename)

 set(handles.devnam,'string',(meena_on));              %DAQ name
```

```
 set(handles.frequency,'string',num2str(fr));          %frequency
 set(handles.pulse_width,'string',num2str(pw));        %time of the
first part of the Pulse (ms)
 set(handles.time_train,'string',num2str(t1));         %time of the
Train (ms)
 set(handles.interpulse,'string',num2str(oww));        %time of the
interpulse delay (ms)
 set(handles.update_rate,'string',num2str(sr1));       %sampling rate
for the Pulse, Train, Frame
 set(handles.amplitude,'string',num2str(am1));         %amplitude of
the pulse
 set(handles.channel_range,'string',num2str(cr1));     %Channel Range
of the analog output
 set(handles.phasic_mono,'value',phasicm);             %choose
monophasic
 set(handles.phasic_bi,'value',phasicb);               %choose biphasic
 set(handles.phasic_laser,'value',phasicl);            %choose laser
 % FRAME PARAMETERS
 set(handles.frame_time,'string',num2str(t2));                  %time
of the Frame (s)
 set(handles.no_frames,'string',num2str(r));
 set(handles.aIoDelay,'string',num2str(delayAIao));
 % PULSE MODULATION PARAMETERS
 set(handles.K_yo,'string',num2str(KK_yo));
 set(handles.pulse_mod,'Value',pul_val);

% SIGNAL MODULATION PARAMETERS
set(handles.sig_funct,'Value',sig_val);                         %signal
modulation drop down menu
set(handles.amplitude_function,'string',num2str(amf));
%function amplitude
set(handles.amplitude_modulation,'string',num2str(amm));
%modulation amplitude
set(handles.freq_wav,'string',num2str(ff));
%modulation frequency
% AMPLITUDE MODULATION PARAMETERS
set(handles.step_size,'string',num2str(stepp));             %step size
set(handles.max_amp_mod,'string',num2str(max_amp));         % max level
of amplitude
 % RECORDING PARAMETERS
 set(handles.trial_no,'string',num2str(tn));            %trial #
 set(handles.time_recording,'string',num2str(t_RP));    %recording time
 set(handles.samplingRate_recording,'string',num2str(sr2));
%sampling rate
 set(handles.max_voltage,'string',num2str(MaV)) ;       %Maximum
Voltage Range
 set(handles.min_voltage,'string',num2str(MiV));        %Minimum
Voltage Range
 set(handles.channelRange_recording,'string',num2str(cr2));
%Channel Range of the analog input
 set(handles.config_single,'value',configS);
 set(handles.config_diff,'value',configD);
 set(handles.config_nonref,'value',configN);
 % LIVE PLOTTING PARAMETERS
 set(handles.row_plott,'string',num2str(row_numb));             %row
specification
```

```
 set(handles.column_plott,'string',num2str(column_numb));
%column specification
 set(handles.norm_plot,'value',normal_option);
 set(handles.avgg_plot,'value',avg_option);
 set(handles.none_plot,'value',none_option);

function devnam_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function devnam_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton49.
function pushbutton49_Callback(hObject, eventdata, handles)

message_pushbutton49 = sprintf('Name of Data Acquisition Card\n\nThe
default name of the hardware usually starts with "Dev" followed by a
number\n\nThis value can be accessed by referring to the National
Instruments "Measurement and Automation Explorer"\n\nAdditionally the
use the following code to access the DAQ card name thorough
Matlab:\n\nnote: replace the double quote with a single quote in the
code below\n\nhw =
daqhwinfo("nidaq");\n\nhw.InstalledBoardIds\n\nhw.BoardNames');
uiwait(msgbox(message_pushbutton49,'Device Name', 'help'));

% --- Executes on button press in avgg_plot.
function avgg_plot_Callback(hObject, eventdata, handles)

% --- Executes on button press in norm_plot.
function norm_plot_Callback(hObject, eventdata, handles)

% --- Executes on button press in pushbutton53.
function pushbutton53_Callback(hObject, eventdata, handles)

function anyPower_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.
function anyPower_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Source codes for NSTA_modualtion_func.m :

```
function [w,x,train_mod, sig_mod,func_mod,max_amp]=
NSTA_modulation_func(fr,pw,t1,sr1,am1,phasicm,phasicb,phasicl,oww,t2,r,
amf,amm,ff,cchoice,funt_choice,KK_yo,max_amp,stepp)

% ----------------------------------------------------------------------
---------
% NeuralSTA - A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING
APPLICATIONS
%                              WITH LASER CONTROL.
% ----------------------------------------------------------------------
---------
% Note: This function generates the stimulation signal after all the
parameters
%       are specified


%preallocation
igobb=0;
max_amp1=max_amp;

%normalizing for lazer
if phasicl==1    %this code (next 6 lines) will allow one to convert
power% to voltage
    %%%%%%%%%%conversion
    llxl=KK_yo*am1;
    llyl=(113.4-llxl)/25.39;   %laser formula
    llrl=.01;
    llryl=llrl*round(llyl/llrl);
    amp_initial=llryl;
    %%%%%%%%%%conversion
    allxl=max_amp1;
    allyl=(113.4-allxl)/25.39;   %laser formula
    allryl=llrl*round(allyl/llrl);
    max_amp=allryl;
    %%%%%%%%%%conversion
    bllxl=amf;
    bllyl=(113.4-bllxl)/25.39;   %laser formula
    bllryl=llrl*round(bllyl/llrl);
    amf=bllryl;
    %%%%%%%%%%conversion
    cllxl=amm;   %offset
    cllyl=(113.4-cllxl)/25.39;   %laser formula
    cllryl=llrl*round(cllyl/llrl);
    amm=cllryl;

    if isempty(max_amp)|| isempty(stepp) || max_amp1<amp_initial ||
stepp==0
        max_amp=amp_initial;
        stepp=r;
    end

    if max_amp>5
        max_amp=5;
    end
```

```
end

%preallocation
igobb=0;

if phasicb==1 || phasicm==1
    amp_initial=KK_yo*am1;
    if isempty(max_amp)|| isempty(stepp) || max_amp==0 || stepp==0
        max_amp=KK_yo*am1;
        stepp=r;
        igobb=1;
    end
end

%-----------------------------------------------------------------------
----

%amplitude calculation for frame by frame modulation
am1a= linspace(amp_initial,max_amp,stepp);                 %PLACE OF
MIN. AMP. CACLCULATION

if r>stepp
    am1a(stepp:r)=max_amp;
end

amp1=am1a;
%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
% WAVE CREATION
%normalizing calculations:
t1=t1/1000;             %total time of the Train (s)
pw=pw/1000;             %pulse width of the first part of the Pulse (s)
pww=oww/1000;           %pulse width of the second part of the Pulse

%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~~~~~~%~~~~~~~~~~~~~
~~%
%PULSE CALCULATIONS

T=1/fr;                           %Time period of the Pulse (s)
tp=round(T*sr1);                  %# of samples in the Pulse
tt=round(t1*sr1);                 %# of samples in the Train
tf=round(t2*sr1);               %# of samples in the Frame
jj=round(tt/tp)+2;                %loop amount
pws=pw*sr1;                      %# of samples in the first part of
Pulse
l2=round(pww*sr1);                %# of samples in the second part of
Pulse
l4=round(tp-((3*pws)+l2));        %# of samples in the fourth part of
Pulse
if l4<1
    l4=0;
end
%ttRP_length=length(ttRP)
ttRP_addition=zeros(1,(tf-tt));  %pause at the end of the Train
```

```
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
%SIGNAL MODULATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
pwf=sr1*t1;             %# of samples for the function
rxx=1:pwf;
time1=rxx/sr1;

if cchoice==1;                % ---> normal
    sig_mod='normal';
    ys=ones(1,pwf);
elseif cchoice==2             % ---> cosine
    sig_mod='cosine';
    ys=(amf*cos(2*pi*ff*time1))+amm;
elseif cchoice==3             % ---> sawtooth
    sig_mod='sawtooth';
    ys=(amf*sawtooth(2*pi*ff*time1))+amm;
elseif cchoice==4             % ---> sine
    sig_mod='sine';
    ys=(amf*sin(2*pi*ff*time1))+amm;
elseif cchoice==5             % ---> square
    sig_mod='square';
    ys=(amf*square(2*pi*ff*time1))+amm;
end

ys(ys<0) = 0;
 if phasicl==1
     ys(ys>5) = 5;
 end


%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
%PULSE MODULATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%parameters

t_mod=linspace(1/sr1,pw,pws);
t_mod1=linspace(1/sr1,1,pws);

% preallocation
funt_moo=(ones(1,uint16(pws)));

%~~Anodic Calculations~~~
for moo=1:pws
    if funt_choice==1                  %normal  ---> 1
        func_mod='normal';
        funt_moo(moo)=1;              % positive part
    elseif funt_choice==2              %linear increase  ---> 2
        func_mod='linear increase';
        funt_moo(moo)=(t_mod1(moo));
    elseif funt_choice==3              %linear decrease  ---> 3
        func_mod='linear decrease';
        funt_moo(moo)=(1-t_mod1(moo));
    elseif funt_choice==4              %exponential increase ---> 4
        func_mod='exponential increase';
        calc_ei=(5*(pw-t_mod(moo))/(pw));
        funt_moo(moo)=(exp(-1.*calc_ei));
```

```
    elseif funt_choice==5              %exponential decrease ---> 5
        func_mod='exponential decrease';
        calc_ed=(5*(t_mod(moo))/(pw));
        funt_moo(moo)=(exp(-1.*calc_ed));
    elseif funt_choice==6              % gaussian  ---> 6
        func_mod='gaussian';
        funt_moo(moo)=exp(-1*((((t_mod(moo))-
(pw/2))/((2^.5)*(pw/5)))^2));
    elseif funt_choice==7              %sinusoidal  ---> 7
        func_mod='sinusoidal';
        funt_moo(moo)=sin((pi*t_mod(moo))/(pw));
    end
end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~signal amplitude
calculations~~~~~~~~~~~~~~~~

% preallocation
wax=zeros(stepp,tt);
waxx=zeros(stepp,(tt+(tf-tt)));
w=zeros(r,(tt+(tf-tt)));
ys1=ys;

for sp=1:r;
    if cchoice~=1 && phasicl~=1
        ys(ys>amp1(sp))=amp1(sp);
        ys(ys<0)=0;
    elseif cchoice~=1 && phasicl==1
        ys(ys<amp1(sp))=amp1(sp);
        ys(ys>5)=5;
    elseif cchoice==1
        ys=ys*amp1(sp);
    end

%conditions for laser mode
if phasicl==1
    if funt_choice==1
        rimiss=funt_moo;
    elseif funt_choice>=6
        rimiss=1-funt_moo;
    else
        rimiss=1+funt_moo;
    end
    if cchoice~=1 && funt_choice~=1
        rimiss=2-funt_moo;
    end

end


%~~Cathodic Calculations~~~
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
%area calculations
rasa=sum(funt_moo); %length of the anode
```

```
samp3=2*pws;  %width of the anode
am3=(rasa/samp3); % length of cathode

%pulse width calculations
pw1a=funt_moo;              % positive part
pw2=zeros(1,l2);                    % zero break
pw3=(-1*am3)*(ones(1,2*uint16(pws)));        % negative part
pw4=zeros(1,l4);                    % rest of the frame which is zero
[m,n]=size(pw4);

if m==1 && n==0
    pw4=[];
end

yys=pws;
tts=tt;

%~~PULSE Build~~~
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
if phasicm==1                   %  --->  Monophasic
    train_mod='Monophasic';
    pw5=zeros(1,(tp-pws));
    pw1=pw1a;
    signal=[ pw1 pw5 ];

else if phasicb==1              %  --->  Biphasic
        train_mod='Biphasic';
        pw1=pw1a;
        signal_1=[ pw1 pw2 pw3 pw4 ];
        signal=signal_1(1:tp);

    else if phasicl==1          %  --->  Laser
            train_mod='Laser';
            pw1=rimiss;                 % positive part

            pw5=(ones(1,(tp-pws)))*5;
            signal=[ pw5 pw1 ];
            ttRP_addition=5*ones(1,(tf-tt));
        end
    end
end

%~~TRAIN Calculations~~~
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~
wave=signal;
ttx=1;
pwsg=1;
suum=tp;
ttxx=(suum-pws);

rever_sal=ones(1,pwf);  %just for laser

wavet=wave;
```

```
jcm=ys;

% loop to create train
for i=1:jj-1
    wavet=[wavet wave];
    if phasicl==1
    jcm(pwsg:ttxx)=1;
    rever_sal(pwsg:ttxx)=0;
    pwsg=pwsg+suum;
    ttxx=ttxx+suum;
    end
    ys=jcm;
end

if phasicl==1
     if cchoice==1 && funt_choice~=1   %for pulse mod
        gaga=ys.*rever_sal;
        wax(sp,:)=wavet(1:tt)+gaga(1:tt);
        wax(wax>5)=5;
     elseif cchoice~=1 && funt_choice==1  %for signal mod
%           gaga=ys.*rever_sal;
         wax(sp,:)=wavet(1:tt).*ys(1:tt);
         wax(wax>5)=5;
     elseif cchoice==1 && funt_choice==1   %for none
         wax(sp,:)=wavet(1:tt).*ys(1:tt);
         wax(wax>5)=5;
     elseif cchoice~=1 && funt_choice~=1  %for both

         wax(sp,:)=wavet(1:tt).*ys(1:tt);
         wax(wax>5)=5;
         wax(wax<amp1(sp))=amp1(sp);
     end
else
    wax(sp,:)=wavet(1:tt).*ys(1:tt);   %signal modulation integration
end

%~~FRAME Calculations~~~
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~

waxx(sp,:)=[wax(sp,:) ttRP_addition];
w=waxx;

lwav=length(waxx(sp,:));     %length of the entire signal

x=linspace(0,t2,lwav);

if igobb==1

    break;
end

ys=ys1;
end
```

```
%condition for no frame by frame modulation
if igobb==1
    for spx=1:r;
        w(spx,:)=waxx(1,:);
    end
else
    w=waxx;
end
```

Source codes for NSTA_error_check_func.m:

```
function [main_msg,tot] = NSTA_error_check_func(am1, KK_yo, oww, pw,
t1,t2, MaV, MiV, tn, t_RP, fr, amf, amm, ff, normal_option, avg_option,
row_numb, column_numb,r, meena_on, cchoice, phasicb)

% ----------------------------------------------------------------------
---------
% NeuralSTA - A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING
APPLICATIONS
%                              WITH LASER CONTROL.
% ----------------------------------------------------------------------
---------
% Note: This function generates a list of errors in the parameters
entered by the user.


main_msg=sprintf('* * * * * * * ERROR SUMMARY * * * * * * *\n\n\n\n');

%1. Amplitude selection
if am1==0
    msg1=sprintf('TRAIN PARAMETERS --> AMPLITUDE \n\nThe amplitude is
zero. Make sure that it is the value that you desire\n\n*     Note: The
final amplitude is a product of the amplitude(under Train Parameters)
and the K value\n\n      *Suggestion: You can use the "Plot Initial
Signal" button to see the frame-view of the signal\n-------------------
---------------\n\n\n');
else if isempty(am1)
        msg1=sprintf('TRAIN PARAMETERS --> AMPLITUDE\nEnter a numeric
value\n----------------------------------\n\n\n');
    else
        msg1=[];
    end
end

if KK_yo==0
    msg2=sprintf('PULSE MODULATION --> K \n\nThe value of "K" is zero.
Make sure that it is the value that you desire\n\n     *Note: The final
amplitude is a product of the amplitude(under Train Parameters) and the
K value\n\n     *Suggestion: You can use the "Plot Initial Signal"
button to see the frame-view of the signal\n---------------------------
------\n\n\n');
else if isempty(KK_yo)
```

```
        msg2=sprintf('PULSE MODULATION --> K\nEnter a numeric value\n--
-------------------------------\n\n\n');
    else
        msg2=[];
    end
end

%2. Time selection
if isempty(oww)
    msg3=sprintf('TRAIN PARAMETERS --> Inter-Pulse Width\nEnter a
numeric value\n---------------------------------\n\n\n');
else
    msg3=[];
end
%..............
if isempty(pw)
    msg4=sprintf('TRAIN PARAMETERS --> Pulse Width\nEnter a numeric
value\n---------------------------------\n\n\n');
else
    msg4=[];
end
%..............
if isempty(t1)
    msg5=sprintf('TRAIN PARAMETERS --> Train Duration\nEnter a numeric
value\n---------------------------------\n\n\n');
else if (oww+pw)>t1
        t1
        oww+pw
        msg5=sprintf('PULSE MODULATION --> Pulse Width, Inter-Pulse
Width, Train Duration\n\n*The sum of the Pulse Width and Inter-Pulse
width should not exceed the Train Duration\n---------------------------
------\n\n\n');
    else
    msg5=[];
    end
end
%..............
if isempty(t2)
    msg6=sprintf('FRAME PARAMETERS --> Frame Duration\nEnter a numeric
value\n---------------------------------\n\n\n');
else
    msg6=[];
end
%..............
if t1>(t2*1000)
    msg8=sprintf('PULSE MODULATION --> Train Duration\nFRAME PARAMETERS
--> Frame Duration\n\n*The Train Duration cannot exceed the Frame
Duration\n---------------------------------\n\n\n');
else
    msg8=[];
end
%-------------RECORDING PARAMETERS section-----------------------------
---
if MiV>MaV
    msg9=sprintf('RECORDING PARAMETERS --> Max Input Voltage\nRECORDING
PARAMETERS --> Min Input Voltage\n\n*The Minimum Voltage value must not
```

```
exceed or equal the Maximum Voltage Value\n----------------------------
-----\n\n\n');
else
    msg9=[];
end

if isempty(tn)
%     tn
    msg15=sprintf('RECORDING PARAMETERS --> Trial #\nEnter a numeric
value\n---------------------------------\n\n\n');
else
    msg15=[];
end

if isempty(t_RP)
    msg16=sprintf('RECORDING PARAMETERS --> Recording Time\nEnter a
numeric value\n---------------------------------\n\n\n');
else
    msg16=[];
end

%-------------FREQUENCY AND TIME PARAMETERS section--------------------
------------
if isempty(fr)
    msg10=sprintf('TRAIN PARAMETERS --> Pulse Frequency\nEnter a
numeric value\n---------------------------------\n\n\n');
else if (1/fr)>(t1/1000)
        freq_min=1000/t1;
        t1_min=((1/fr)*1000);
        msg10=sprintf('TRAIN PARAMETERS --> Pulse Frequency\nTRAIN
PARAMETERS --> Train Duration\n\n*The Pulse Frequency value must not
exceed the value of %d\n     or\n*The Train Duration value must not
exceed the value of %d\n---------------------------------
\n\n\n',freq_min,t1_min);
    else
    msg10=[];
    end
end

%-------------SIGNAL MODULATION PARAMETERS section---------------------
-----------
if cchoice~=1
    if ff<1
        msg11=sprintf('SIGNAL MODULATION --> Modulation
Frequency\n\nThe Modulation Frequency value cannot be less than
1\n\n*Suggestion: You can choose the "Normal" option from the drop down
menu (SIGNAL MODULATION section) to disable signal modulation\n--------
-------------------------\n\n\n');
    else
        msg11=[];
    end

    if isempty(ff)
        msg12=sprintf('SIGNAL MODULATION --> Modulation
Frequency\nEnter a numeric value\n---------------------------------
\n\n\n');
```

```
    else
        msg12=[];
    end
    if isempty(amf)
        msg13=sprintf('SIGNAL MODULATION --> Norm. Depth\nEnter a 
numeric value\n---------------------------------\n\n\n');
    else
        msg13=[];
    end
    if isempty(amm)
        msg14=sprintf('SIGNAL MODULATION --> Offset\nEnter a numeric 
value\n---------------------------------\n\n\n');
    else
        msg14=[];
    end
else
        msg11=[];msg12=[];msg13=[];msg14=[];
end

%-------------LIVE PLOT section--------------------------------
if normal_option==1 || avg_option==1;
    if isempty(row_numb)
        msg17=sprintf('LIVE PLOT --> Plotting order --> Row(s)\nEnter a 
numeric value\n---------------------------------\n\n\n');
    else
        msg17=[];
    end
    if isempty(column_numb)
        msg18=sprintf('LIVE PLOT --> Plotting order --> 
Column(s)\nEnter a numeric value\n---------------------------------
\n\n\n');
    else
        msg18=[];
    end
else
    msg17=[];msg18=[];
end

%-------------FRAME PARAMETER section--------------------------------
if isempty(r)|| r<1
    msg19=sprintf('FRAME PARAMETERS --> # of Frames\n\nEnter a numeric 
value greater than or equal to one\n---------------------------------
\n\n\n');
else
    msg19=[];
end

%-------------OTHER sectionS--------------------------------
if isempty(meena_on)
    msg20=sprintf('TRAIN PARAMETERS --> Device Name\n\nEnter the DAQ 
card name\n---------------------------------\n\n\n');
else
    msg20=[];
end

if phasicb==1
```

```
    gxx=((3*pw)+oww); %if the pulse duration exceeds
    gyy=(1/fr*1000);
    if gxx>gyy
        msg21=sprintf('PULSE PARAMETERS --> Pulse Width\n\nThe sum of
the pulse width and the inter-pulse width must not be greater than
%d\n---------------------------------\n\n\n',gyy);
    else
        msg21=[];
    end
else
     msg21=[];
end

%putting together
tot=[msg1 msg2 msg3 msg4 msg5 msg6 msg8 msg9 msg10 msg11 msg12 msg13
msg14 msg15 msg16 msg17 msg18 msg19 msg20 msg21];
```

Source codes for NSTA_live_plot.m:

```
function NSTA_live_plot_func(data3,row_numb,
column_numb,cr2,max_amp,sr2,rr,w,x,phasicl)

% --------------------------------------------------------------------
---------
% NeuralSTA - A SOFTWARE TOOL FOR NEURAL STIMULATION AND RECORDING
APPLICATIONS
%                              WITH LASER CONTROL.
% --------------------------------------------------------------------
---------
% Note: This function generates live plot of the data saved


rowx=row_numb;
columnx=column_numb;
ttotalx=rowx*columnx+columnx;
xv=size(data3);
%
yy=xv(2);



figure(1)
subplot(rowx+2,columnx,[1:columnx]); plot(x,(w(rr,:)),'r')
xlabel('Time (s)', 'fontweight','b');
ylabel('Amplitude (V)', 'fontweight','b');
title('Initial Signal', 'fontweight','b');

%preallocation
jjt=0;
llx=length(data3(:,1));
lengthx=1/sr2:1/sr2:llx/sr2;

wmax_amp=-max_amp;
if phasicl==1
```

```
    wmax_amp=.5;
end

for jj=columnx+1:ttotalx
    jjt=jjt+1;         %title display
    jjtx=cr2(jjt);     %title display
    figure(1)
    subplot(rowx+2,columnx,jj); plot(lengthx,data3(:,jjt))  % jj gives
the row number (also the frame)
    xlabel('Time (s)');
    ylabel('Amp. (V)');
    title(['Channel # ',num2str(jjtx)]);
    axis([0 llx/sr2 wmax_amp (max_amp+1)]) %setting axis of the channel
plots

    if yy==jjt
        break
    end
end
```

REFERENCES

[1] Butson, C. R., & McIntyre, C. C. (2007, August). Differences among implanted pulse generator waveforms cause variations in the neural response to deep brain stimulation. *Clinical Neurophysiology*, *118*(8), 1889–1894. doi:10.1016/j.clinph.2007.05.061

[2] Rubinstein, J. T., Miller, C. A., Hiroyuki, M., & Abbas, P. J. (2001, October). Analysis of monophasic and biphasic electrical stimulation of nerve. *IEEE Transactions on Biomedical Engineering*, *48*(10), 1065-1070. doi:10.1109/10.951508

[3] Field-Fote, Anderson, Robertson, & Spielholz. (2003, August). Monophasic and biphasic stimulation evoke different responses. *Muscle & Nerve*, *28*(2), 239-41. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/12872331

[4] Piallat. B., Chabardes, S., Devergnas, A., Torres, N., Allain, M., Barrat, E., Benabid, AL. (2009, January). Monophasic but not biphasic pulses induce brain tissue damage during monopolar high-frequency deep brain stimulation. *Neurosurgery*, 64(1), 156-62. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/19145164

[5] *Data Acquisition Toolbox*. (2011). Retrieved November 15, 2011, from The Mathworks, Inc. website: http://www.mathworks.com/products/daq/

[6] *Legacy Serial Interfaces*. (2011). Retrieved November 15, 2011, from National Instruments Corporation website: http://sine.ni.com/nips/cds/view/p/lang/en/nid/10081

[7] R2011b Documentation → Data Acquisition Toolbox. (2011). *Analog Input Subsystem*. Retrieved November 15, 2011, from The Mathworks, Inc. website: http://www.mathworks.com/help/toolbox/daq/f5-24516.html

[8] CompactDAQ. (2011). *What Is NI CompactDAQ?* Retrieved November 15, 2011, from National Instruments Corporation website: http://www.ni.com/compactdaq/whatis/

[9] R2011b Documentation → Data Acquisition Toolbox. (2011). *Sampling*. Retrieved November 15, 2011, from The Mathworks, Inc. website: http://www.mathworks.com/help/toolbox/daq/f5-24516.html#f5-23538

[10] R2011b Documentation → Data Acquisition Toolbox. (2011). *Quantization*. Retrieved November 15, 2011, from The Mathworks, Inc. website: http://www.mathworks.com/help/toolbox/daq/f5-24516.html#f5-24501

[11] R2011b Documentation → Data Acquisition Toolbox. (2011). *Transferring Data from Hardware to System Memory.* Retrieved November 15, 2011, from The Mathworks, Inc. website: http://www.mathworks.com/help/toolbox/daq/f5-24516.html#f5-36873

[12] Sahin M, Tie Y. (2007). Non-rectangular waveforms for neural stimulation with practical electrodes. *Journal of Neural Engineering, 4*(3), 227-33. doi:10.1088/1741-2560/4/3/008

[13] Costenoble, S. R., & Waner, S. (1997). This Section: 1. Modeling with the Sine Function. In *The Trigonometric Functions.* Retrieved September 23, 2010, from http://people.hofstra.edu/Stefan_Waner/trig/trig1.html

[14] Costenoble, S. R., & Waner, S. (1997). This Section: 2. The Six Trigonometric Functions. In *The Trigonometric Functions.* Retrieved September 23, 2010, from http://people.hofstra.edu/Stefan_Waner/trig/trig2.html

[15] Abdo, A., Sahin, M., Freedman, D. S., Cevik, E., Spuhler, P. S., & Unlu, M. S. (2011). Floating light-activated microelectrical stimulators tested in the rat spinal cord. *Journal of Neural Engineering, 8*(5), 056012 (9pp). doi:10.1088/1741-2560/8/5/056012