REAL-TIME PROCESSING OF ELECTROMYOGRAMS IN AN AUTOMATED HAND-FOREARM DATA COLLECTION AND ANALYSIS SYSTEM

by

PHILLIP ANTHONY KUEHL

B.S., Kansas State University, 2013

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Electrical & Computer Engineering

College of Engineering

Kansas State University

Manhattan, Kansas

2015

Approved by

Steve Warren

Major Professor

Steve Warren

ABSTRACT

Handgrip contractions are a useful exercise for assessing muscle fatigue in the forearm musculature. Most conventional hand-forearm ergometer systems require the researcher to manually guide subject activity, collect subject data, and assess subject fatigue after it has occurred. Since post-processing tools are not standardized for this type of experiment, researchers resort to building their own tools. This process can make comparing results between research groups difficult.

This thesis presents updates to a hand-forearm ergometer system that automate the control, data-acquisition, and data-analysis mechanisms. The automated system utilizes a LabVIEW virtual instrument as the system centerpiece; it provides the subject/researcher interfaces and coordinates data acquisition from both traditional and new sensors. The system also processes the hand-forearm data within the LabVIEW environment as the data are collected. This allows the researcher to better understand the onset of subject fatigue while an experiment is in progress.

System upgrades relative to prior work include the addition of new parameters to the researcher display, a change in the subject display from a binary up-down display to a sliding bar for better control over subject grip state, and a software update from a simple data acquisition and display system to a real-time processing system.

The toolset has proven to be a viable support resource for experimental studies performed in the Kansas State University Human Exercise Physiology Laboratory that target muscle fatigue in human forearms. Initial data acquired during these tests indicate the viability of the system to acquire consistent and physiologically meaningful data while providing a useable toolset for follow-on data analyses.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I appreciate the support and assistance of my major professor, Dr. Steve Warren, and the other members of my graduate committee: Dr. Dwight Day, Dr. Punit Prakash, and Dr. Thomas Barstow.

I also extend my gratitude to Carl Ade, Ryan Broxterman, and the other students that support the Kansas State University Human Exercise Physiology Laboratory for their advice and feedback regarding the LabVIEW interface design and its usability.

In addition, I would like to thank my parents, my brother, and my fiancée for supporting me through this project.

This work was supported by the NASA Human Research Program, *Research and Technology Development to Support Crew Health and Performance in Space Exploration Missions*, Exploration Systems Mission Directorate, Johnson Space Center, Houston, TX. Any opinions, findings and conclusions or recommendations in this material are those of the author(s) and do not necessarily reflect the views of NASA.

CHAPTER 1: INTRODUCTION

A. Motivation

The work described in this thesis was supported by the Kansas State University (KSU) Human Exercise Physiology Laboratory (KSU Department of Kinesiology) and the KSU Department of Electrical & Computer Engineering (ECE) with funding from the NASA Human Research Program – *Research and Technology Development to Support Crew Health and Performance in Space Exploration Missions*. The NASA grant aims to provide tools that can help to assess astronaut health during space exploration. One focus of the KSU work is assessing subject fatigue in forearm musculature during intense or repetitive tasks. The studies that motivated the technical work presented in this document focused on monitoring muscle fatigue in human forearms using a two-hand ergometer system. The reliability, accuracy, and integrity of that experimental work depended on how data were collected, processed, and displayed [1]. Typically, data collection and processing, as well as subject movement cues, are all addressed manually by the researcher. This can make comparing and correlating data difficult even within a single research group. One solution is to automate these processes to improve data quality and experimental consistency [2]. This was a primary driver for the work presented here.

B. Traditional Hand-Forearm Ergometer Systems

Handgrip contractions are a standard means to evaluate the intact muscular system [3, 4] and mechanisms of fatigue [5]. Traditional hand-forearm ergometers are limited to a subject pulling against a suspended weight or spring with a single limb. This type of resistance allows the researcher to test the subject only at a single work rate per session. The researcher must prompt the subject with continual squeeze-release-rest cues while also manually acquiring data and identifying subject fatigue [6, 7]. Recent hand-forearm ergometer advancements have automated the data collection process [8, 9] and offer limited controls for data acquisition and post-processing. However, these systems tend to be experimentally specific and do not offer the researcher a central interface to control data collection and processing [8]. The work presented here represents an advancement in hand-forearm ergometry, where an automated hand-forearm ergometer previously developed at Kansas State University [10-12] has been updated to incorporate real-time data processing.

C. Fatigue Parameters

An electromyogram (EMG) is an electrode-acquired, time-domain signal that represents the electrical activity in the musculature nearest to an electrode pair. A typical electromyographic signal and its accompanying frequency-domain power spectrum are depicted in Figure 1.1. EMG parameters traditionally used to track the progression of subject fatigue include mean and median power frequency [13, 14], rectified, integrated EMG [15], and the RMS value of the EMG [16, 17]. Most studies employ sustained muscle contractions when measuring fatigue, where the extended muscle 'bursts' are divided up by time and parameters are calculated for the specific time blocks. By definition, these types of experiments are robust due to the minimization of movement artifacts. However, most real-world activities do not involve sustained muscle contractions but rather a series of muscle contractions. This makes identifying fatigue relatively difficult in comparison to prior work, which ignored the effects of the rest cycle on muscle behavior. Until these effects are better understood, researchers that work in this area are constrained to apply the parameters used in sustained contractions to studies that involve repetitive contractions.

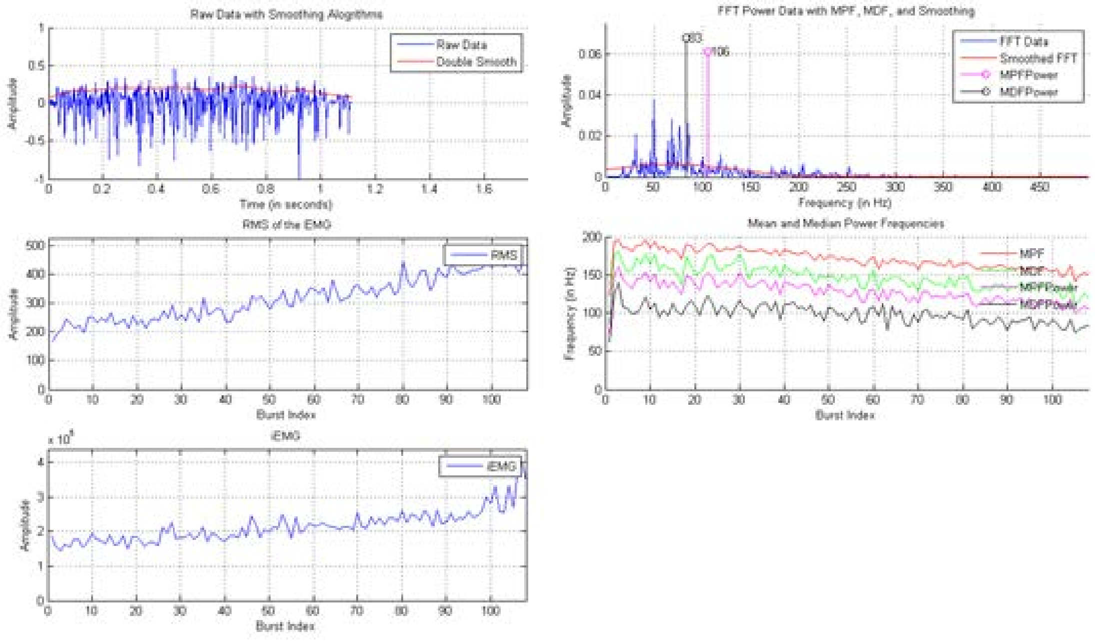


Figure 1.1 Typical EMG burst with fatigue parameter calculations.

The mean and median power frequencies are intuitive fatigue parameters. Two types of muscle fibers exist: fast- and slow-twitch muscle fibers [18]. In fresh muscles, both types of muscle fibers engage to produce the force required to complete a task. As the subject continues the activity, the fast-twitch muscle fibers start to disengage, while the slow-twitch muscle fibers continue to contribute. This results in a shift in the EMG frequency spectrum which can be observed by tracking the changing mean and median power frequencies. These frequencies are calculated per burst using the following formulas [13]:

$$\text{mean frequency} = \frac{\sum_{n=f_l}^{f_h} n * |FFT(n)|}{\sum_{n=f_l}^{f_h} |FFT(n)|}$$

and

$$\text{median frequency} = \sum_{n=f_l}^{f_h} |FFT(n)| > 0.5 * \sum |FFT(n)|$$

where n is a frequency index, $|FFT(n)|$ represents the magnitude of the Fast Fourier Transform [19] of the EMG burst, and f_l and f_h are indices that relate to the lowest and highest frequencies, respectively.

The RMS value (a.k.a., RMS voltage) of an EMG is another intuitive fatigue parameter. It is a measurement of the average amplitude of the EMG and represents the amount of effort the muscle fibers must put forth to complete the task at hand. As a subject experiences fatigue, one of two things occurs: 1) more muscle units are recruited to maintain the desired force or 2) the motor unit firing rates increase [20]. Both cases result in an increase in EMG amplitude and therefore an increase in the RMS value of the EMG. The RMS value is calculated per burst using the following formula [17]:

$$\text{RMS} = \sqrt{\frac{1}{k}\ \sum_n \mathrm{x}_n^2}\ ,$$

where *n* is an index value for the data (*n* = 1,2,3….), *k* is the total number of data points (a constant), and x is the value of the signal at a given *n*.

The rectified, integrated EMG is a less intuitive fatigue parameter. Similar to the RMS value, iEMG is a measurement of the amount of energy in a muscle burst. As the subject experiences fatigue, the force-generating capacity of the muscle decreases, which leads to an increase in EMG amplitude at submaximal contractions [20]. The iEMG is calculated per burst using the formula

$$iEMG[|m(t)|] = \int_0^t |m(t)|\, dt$$

where *m(t)* is the rectified EMG signal [15].

D. Non-Traditional Fatigue Parameters

Non-traditional fatigue parameters are parameters that are thought to be influenced by fatigue but are not agreed upon within the community. Zero crossing rate [13, 14, 21, 22], average rectified value [14, 20, 21], total spectral magnitude [14, 22], skewness, and kurtosis are such parameters that will also be discussed in this thesis.

The zero crossing rate, useful when estimating muscle conduction velocity, relates to how fast the muscle fibers are firing. It has been suggested that muscle conduction velocity decreases as a subject fatigues [20, 23]. However, muscle conduction velocity has also been shown to correlate with the mean power frequency, which would reduce its effectiveness in a multivariate approach to detecting muscle fatigue [24]. Zero crossing rate is sensibly calculated by looking for the number of times the raw EMG signal crosses the time axis (both positive and negative slopes).

The average rectified value (ARV) is similar to the RMS value of a muscle burst, but it differs in an important way. Instead of requiring the square root of a sum of squares, the ARV utilizes the mean of the rectified, raw EMG signal. The ARV is calculated with the following formula [14]:

$$Average\ Rectified\ Value(i) = \frac{1}{N}\sum_{n=1}^{N}|x_i(n)|,$$

where *i* is a burst index and *n* is an index variable used to step through a sequence of N values that comprise a burst.

The total spectral magnitude (TSM) is similar to the iEMG, except that the iEMG is a time-domain parameter and the TSM is a frequency-domain parameter. The TSM is calculated by finding the area under the FFT magnitude spectrum [14]:

$$Total\ Spectral\ Magnitude[|m(f)|] = \int_0^f |m(f)|\, df$$

The skewness and kurtosis values are fatigue parameters that have not been addressed in the literature with regard to muscle fatigue, yet their use seems natural given the change in EMG spectral shape that occurs with fatigue. In this thesis, the skewness and kurtosis values are calculated from the coefficients of the raw power spectrum (LabVIEW implementation – see Section 5.B) as well as from a Gaussian curve that is fit to the raw FFT power spectrum (MATLAB movie – see Section 5.B) [25]:

$$skewness(i) = \frac{\sum_{n=1}^{N}(Y_i(n)-\bar{Y})^3}{s^3}/N$$

and

$$kurtosis(i) = \frac{\sum_{n=1}^{N}(Y_i(n)-\bar{Y})^4}{s^4}/N\ ,$$

where *i* is a burst index, $Y_i(n)$ is a spectral value, $\bar{Y}$ is the average spectral value, *s* is the standard deviation of the spectral values, and *n* and *N* are defined as above.

CHAPTER 2: BACKGROUND

A. Previous Hardware Design

As illustrated in Figure 2.1 and Figure 2.2, the previous hardware design consisted of the existing hand-forearm ergometer, a pressurized force-tank system, an NI USB-6211 data acquisition card, a Delsys EMG sensor, and a personal computer running LabVIEW version 10.0 in Windows 7 [10-12]. The pressure tank system uses a nitrogen gas mixture to pressurize the cylinder. As the subject squeezes and releases the metal bars of the ergometer, the pressurized force-tank system collects and transmits the force and displacement data via the USB-6211 unit, while the Delsys EMG sensor collects and transmits forearm EMG data to the computer via the same USB-6211 unit. The squeezing force exerted on the bars by the subject equates to the current pressure in the cylinder. The experiment continues until the subject can no longer squeeze the two bars together. The previous hardware/software design was comprised of five main parts:

1. ergometer,
2. pressure cylinder and manual control box,
3. potentiometers and load cells,
4. digital metronome and stopwatch (not depicted), and
5. researcher interface.

The ergometer has been utilized when experiments observing the forearm musculature are conducted. A typical experiment consists of the subject squeezing the two metal bars of the ergometer together repeatedly until the subject can no longer keep pace with the metronome. The researcher can manually control the pressure inside the cylinder via a potentiometer dial on the front of the manual control box, depicted in Figure 2.3, or with the LabVIEW interface depicted in Figure 2.4. The latter allows for both constant and ramp force experiments to be performed [10]. The hardware design has been kept the same for this work.

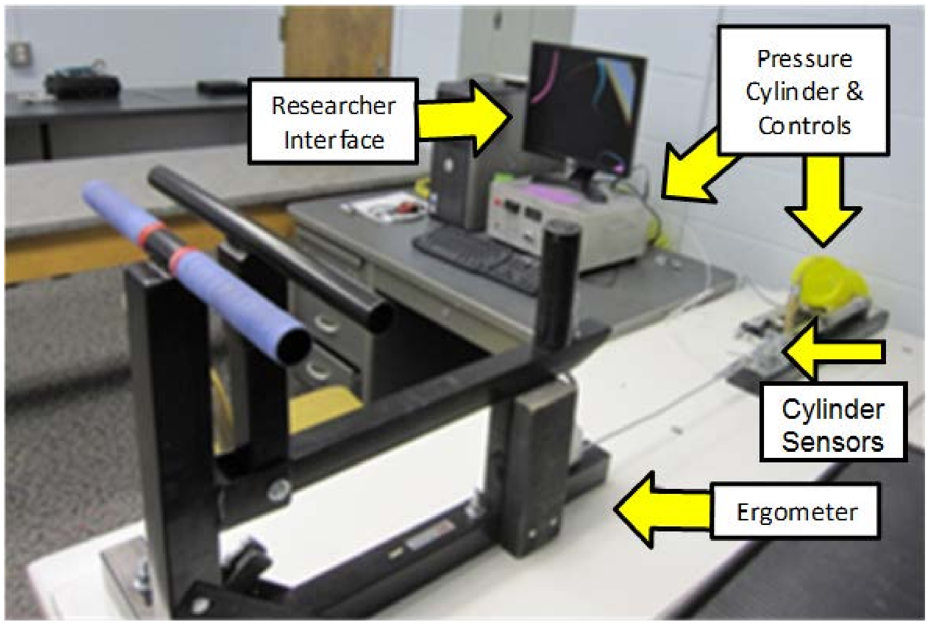


Figure 2.1 Previous hardware design.

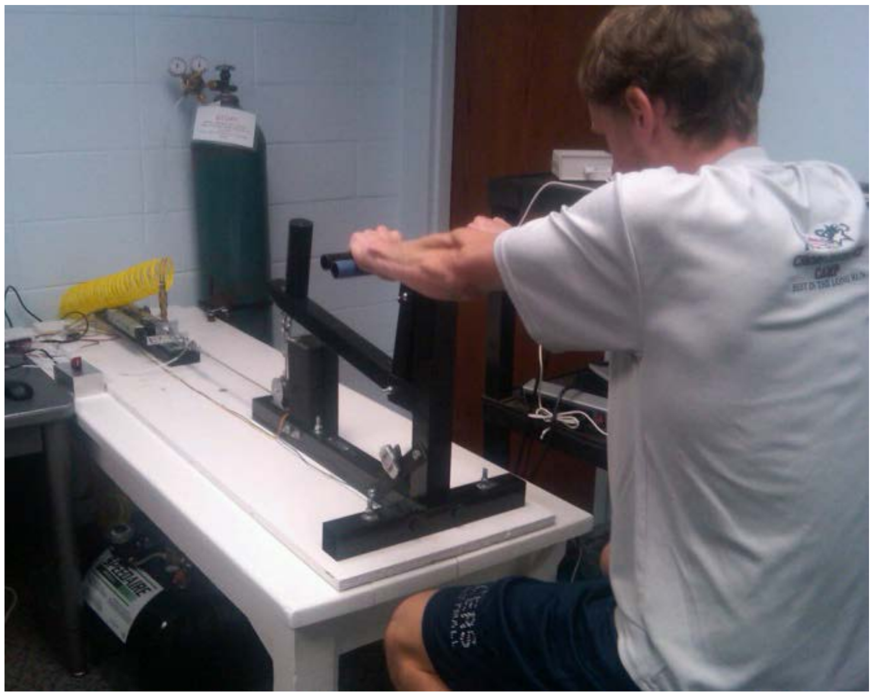


Figure 2.2 Demonstration of the hand-forearm ergometer system.

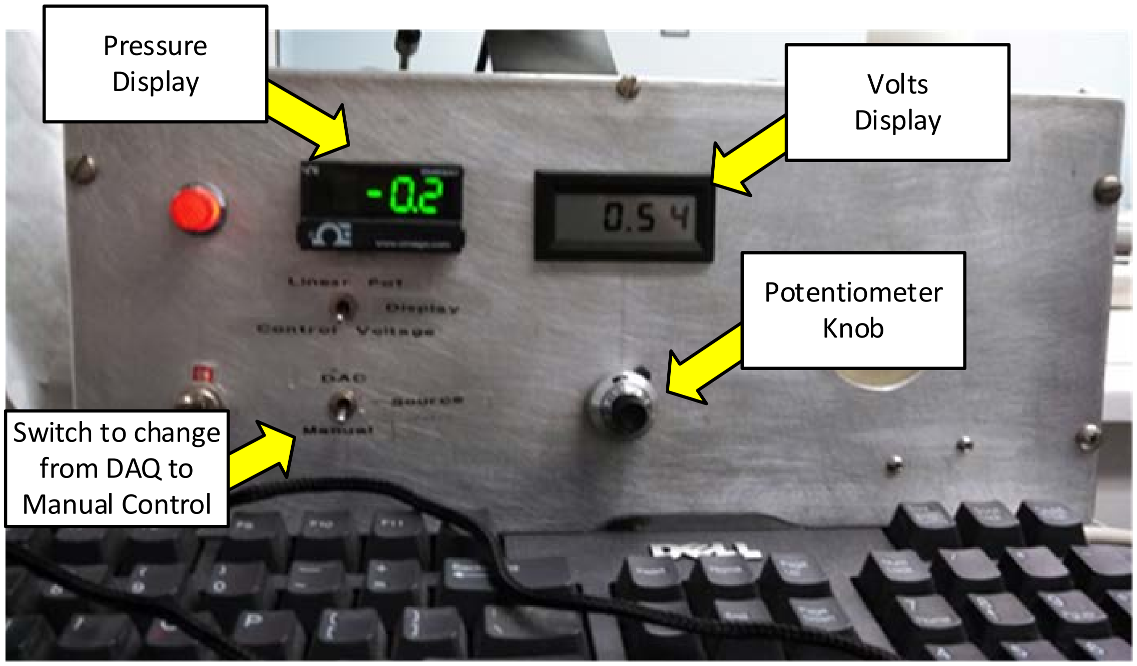


Figure 2.3 Front panel of the manual cylinder-pressure control box.

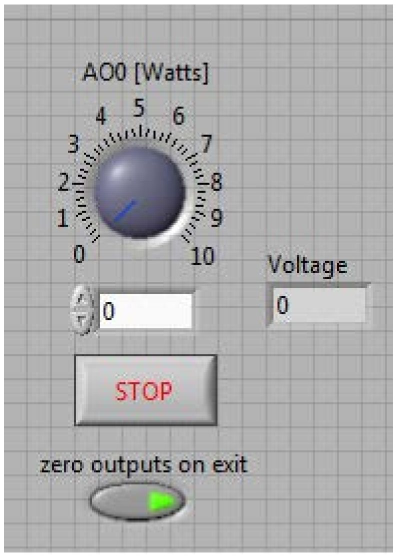


Figure 2.4 LabVIEW interface for the cylinder-pressure control box.

B. Previous Software Design

The previous software design [10-12] consisted of three simple LabVIEW virtual instruments (VIs): one VI to collect subject data, one VI to prompt the subject's forearm movements, and one VI to manipulate the pressure inside the force-tank system. As illustrated in Figure 2.5 and Figure 2.6, the data collection VI utilized a DAQ assistant to retrieve the data from the correct ports and convert them to useful units: force in kilograms, displacement in centimeters, and EMG in microvolts. The subject interface, depicted in Figure 2.7, gave the subject visual cues for when to grip the ergometer handles. Depending on which experiment was in progress, the pressure of the force-tank system could be kept the same, increased, or decreased throughout the experiment. At the conclusion of the experiment, these data were processed using multiple MATLAB scripts to identify the muscle bursts, calculate various fatigue parameters, and compile a burst movie to display burst sequences over time.

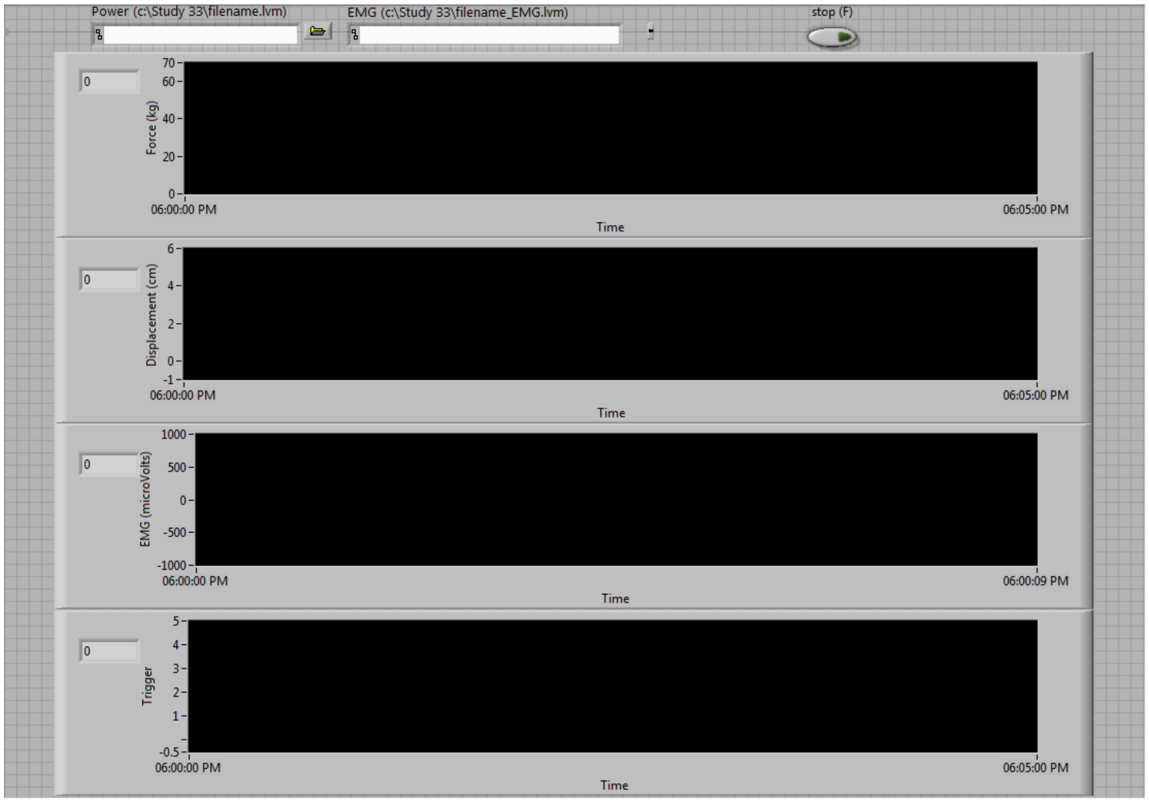


Figure 2.5 Previous researcher interface.

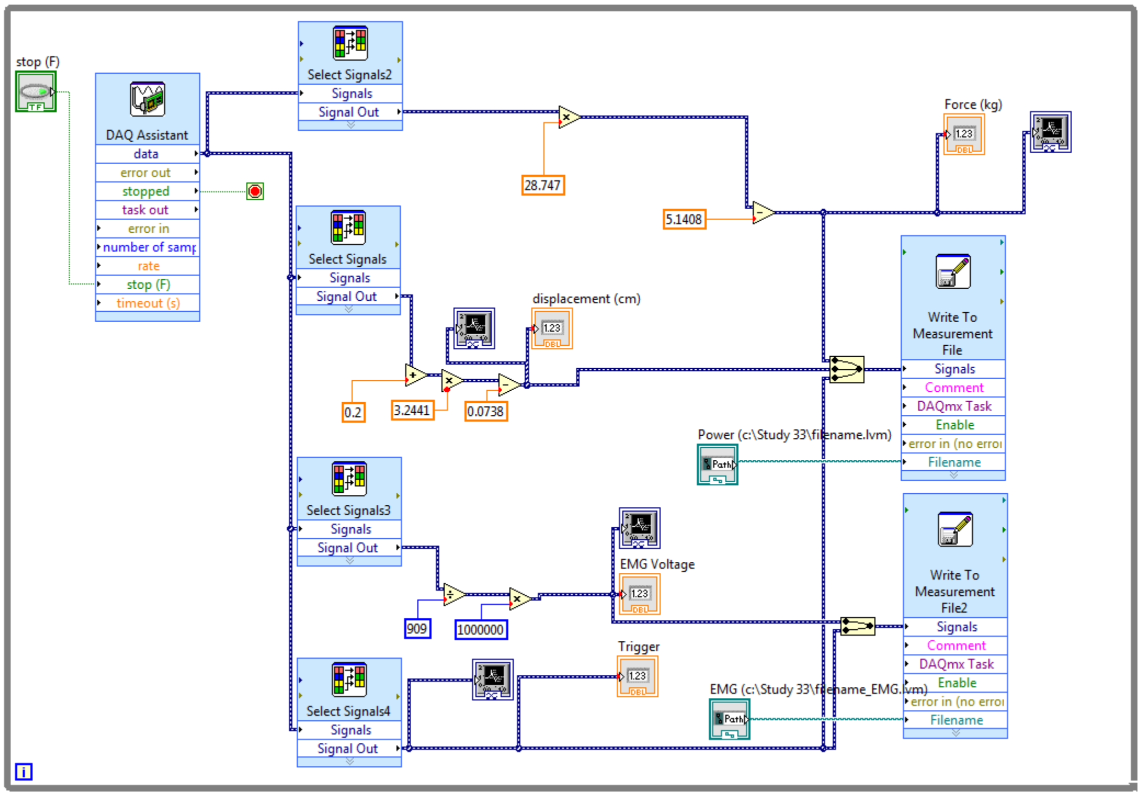


Figure 2.6 Previous software design.

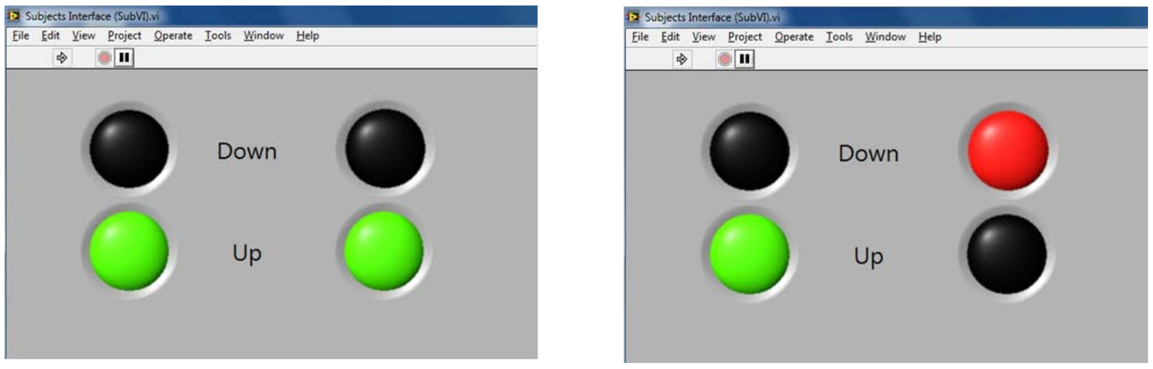

Subjects Interface (SubVI).vi
File Edit View Project Operate Tools Window Help
Down
Up
Subjects Interface (SubVI).vi
File Edit View Project Operate Tools Window Help
Down
Up

Figure 2.7 Previous subject display. The left image is an example of correct subject movement, and the right image is an example of incorrect subject movement.

CHAPTER 3: REAL-TIME PROCESSING OF EMG SIGNALS

A. Producer and Consumer Loop Functionality

To collect and analyze data at the same time can be an arduous task when computing power is limited. Analysis techniques are often complex and intensive, interfering with data collection processes. This effect is compounded as delayed collected data move into analysis processes, further bogging down the system. To prevent this situation, one can separate the data collection and analysis processes from each other and then further prioritize the data collection efforts.

One option is to implement a producer-consumer loop system which contains two `while()` loops connected by a queueing system. The first `while()` loop manages the data collection process and the second loop manages the analysis techniques. Once data are collected, they are sent to the queueing system for the consumer loop to process. Since (a) LabVIEW uses data flow to control which blocks execute first and (b) the blocks are connected as producer—queue—consumer, the producer loop must be executed prior to the consumer loop before data are available to process in the producer loop. So by design, the producer-consumer loop system separates the data acquisition process from the data-analysis process, allowing the data acquisition functionality to run uninhibited from computer-intensive analyses.

Figure 3.1 presents a flowchart that describes the high-level implementation of the producer-consumer loop employed for this work. The system first identifies whether the user wants to run a live experiment or run a replayed experiment, using a given text file, through a Boolean control variable on the researcher screen. Once the type of experiment is determined, the system then enters the producer loop. If the user has indicated that the experiment is a replay, then the system conditions the file by removing the header of the text file. As stated earlier, the producer loop consists solely of data collection processes, ensuring the timing integrity of these data. The consumer loop contains data-analysis processes, such as identifying muscle bursts and calculating fatigue parameters. The functional elements of these producer and consumer loops are described in more detail in the following sections.

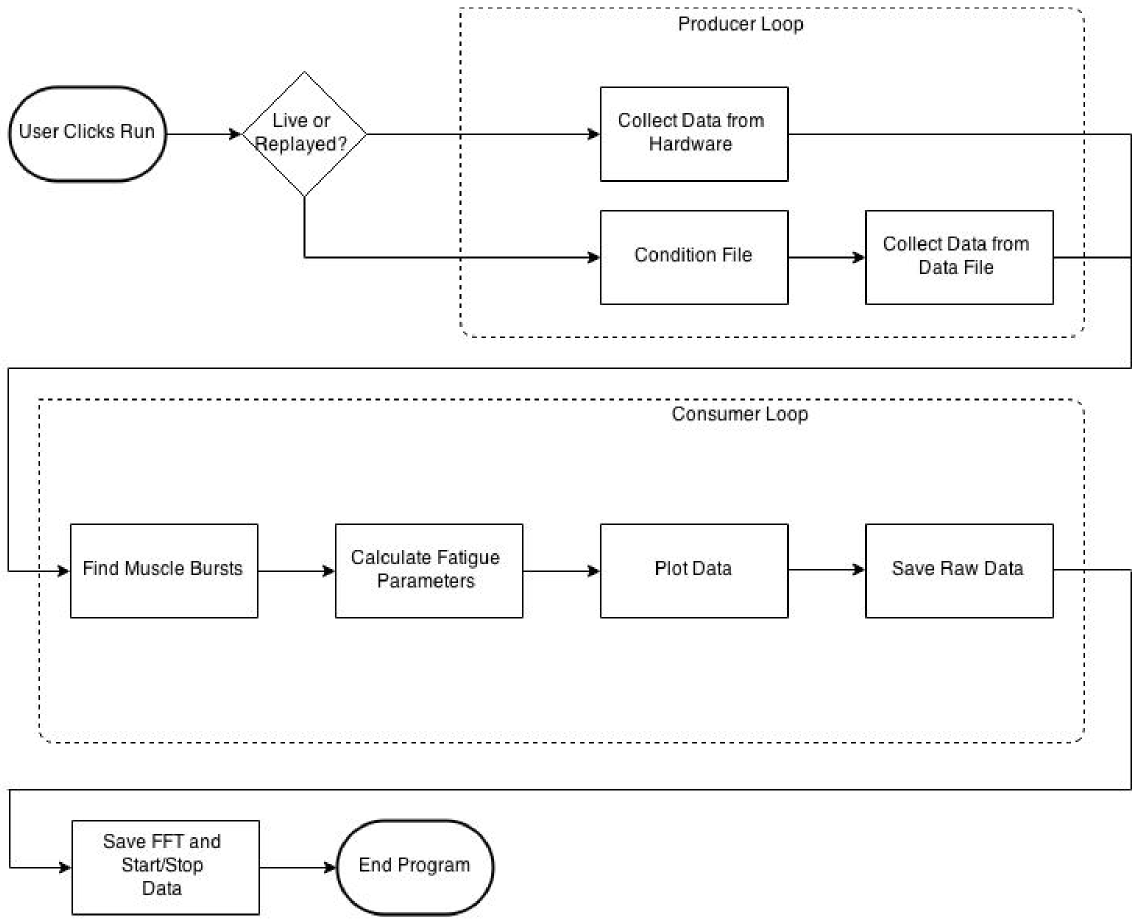


Figure 3.1 Flowchart for the overall producer-consumer loop design.

B. Top-Level VI

B.1 Overview

The top-level VI contains the producer and consumer loops and manages the queueing system that connects both loops. As illustrated in Figure 3.2, the VI takes in the user inputs (contained in the clusters) and returns three data files containing the raw data (force, displacement, and EMG), the frequency–domain spectrum (EMG power coefficients), and parameter values. The data saving procedure is illustrated in Figure 3.3. The producer and consumer loops are discussed in more detail in the following sections.

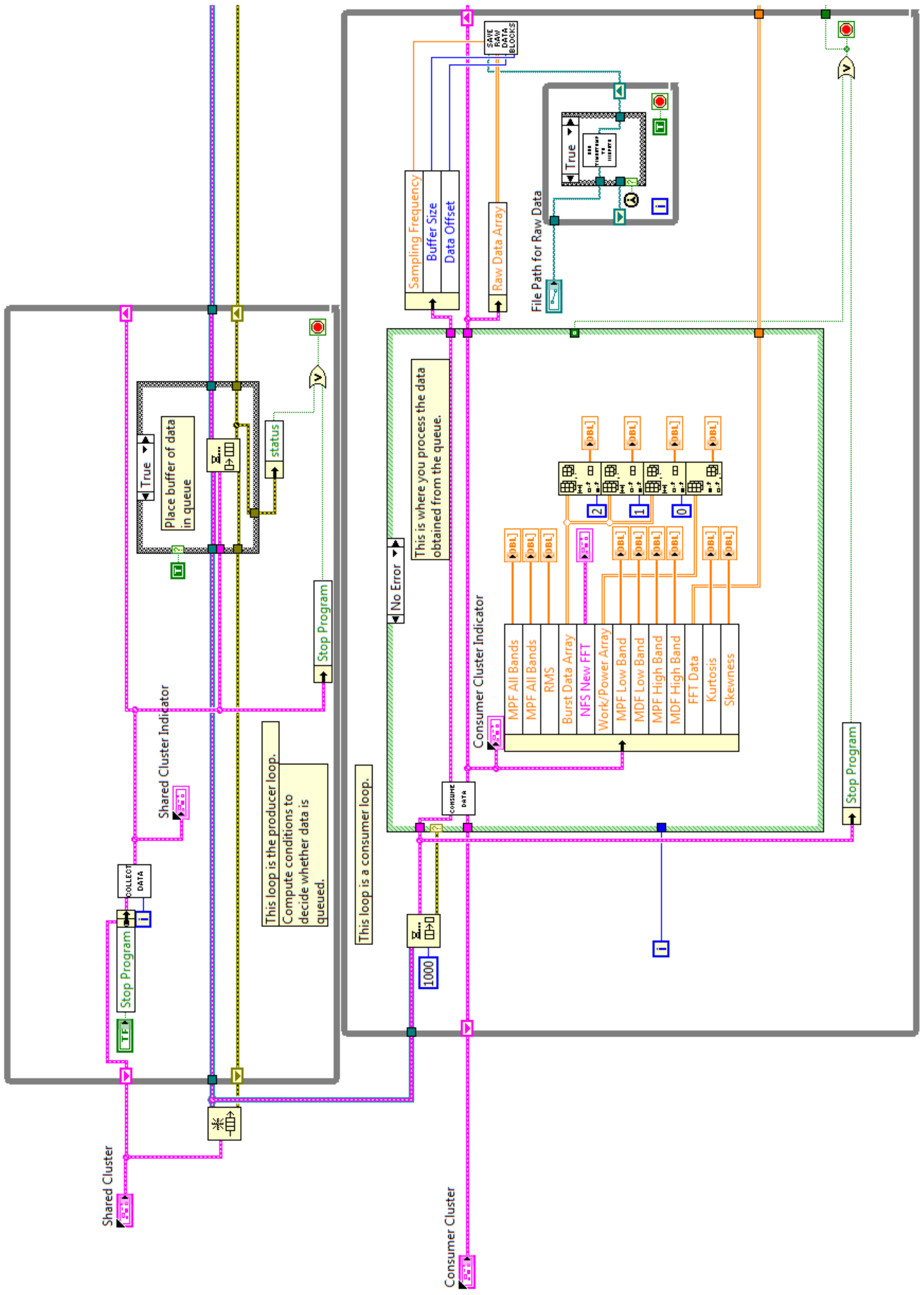


Figure 3.2 Top-Level VI that contains the producer and consumer loops.

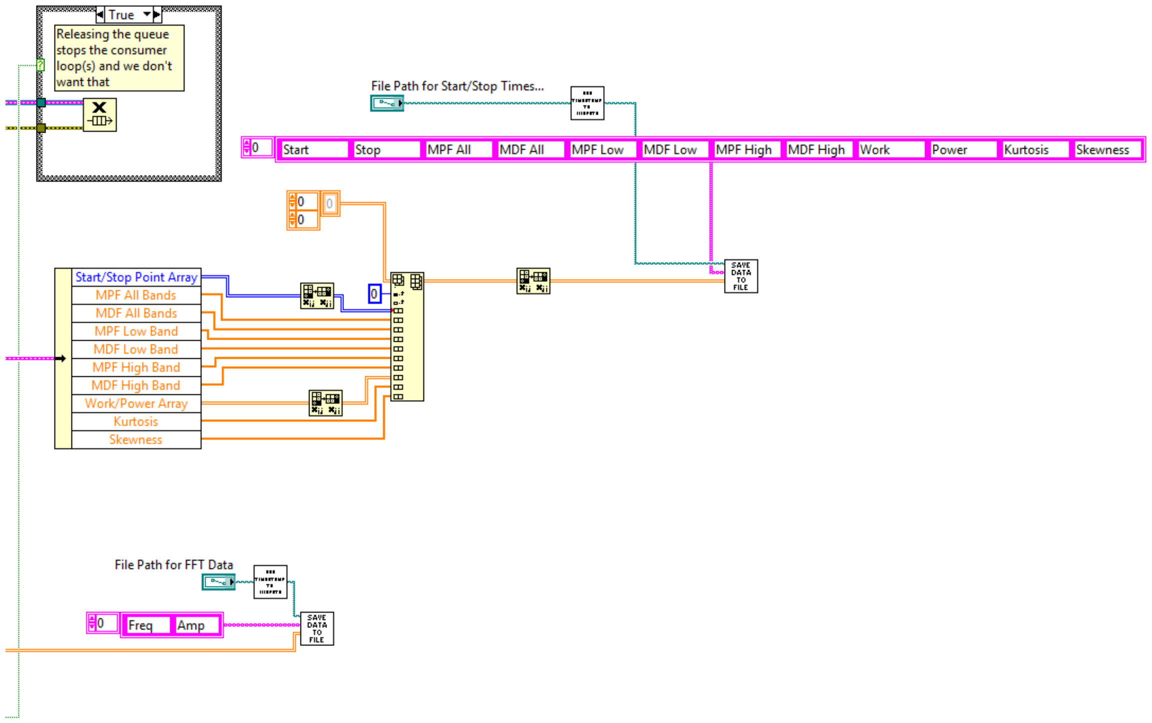


Figure 3.3 Top-Level VI that contains the data saving process.

B.2 Add Timestamp to Filepath VI

The Add Timestamp to Filepath VI adds a timestamp to the end of an input filename. As illustrated in Figure 3.4, the VI receives the file path and then returns the new file path with the timestamp appended. First, the filepath is separated from the filename itself. Then, the current time and date are added to the filename. Lastly, the filename is appended to the end of the file path.

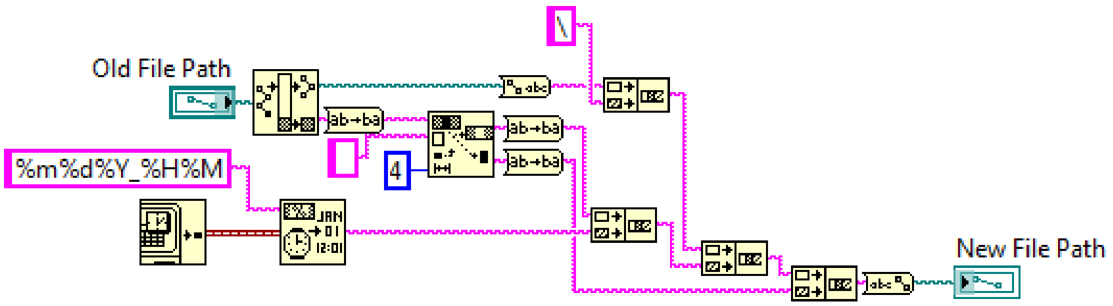


Figure 3.4 Add Timestamp to Filename VI.

B.3 Save Raw Data Blocks VI

The Save Raw Data Blocks VI saves the data to the raw data file. As illustrated in Figure 3.5, the VI receives the file path, the raw data array, the index to save, the buffer size, and the sampling frequency. It then writes the raw data with time values to the appropriate spot in the raw data file. The Index to Save variable is an integer value that represents the current row in the raw data array to start a number of rows of data (equal to Buffer Size) that will be stored in the raw data file. First, the VI writes the header to the file if the current call of the VI is the first call. It then generates the appropriate time values using a `for()` loop. Lastly, the time values and raw data values are written to the end of the raw data file.

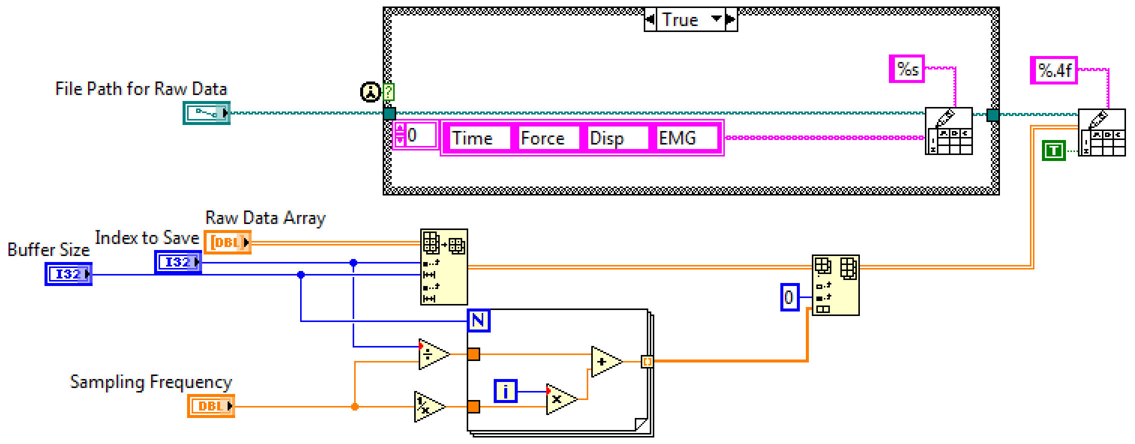


Figure 3.5 Save Raw Data Blocks VI.

B.4 Save Data Files VI

The Save Data Files VI writes a data array and a header to the given file path. As illustrated in Figure 3.6, the VI receives the file path, the header array, and the data array. It then writes the data to the end of the given file. This VI is used to write both the FFT (power) coefficients and the parameter data to their respective files.

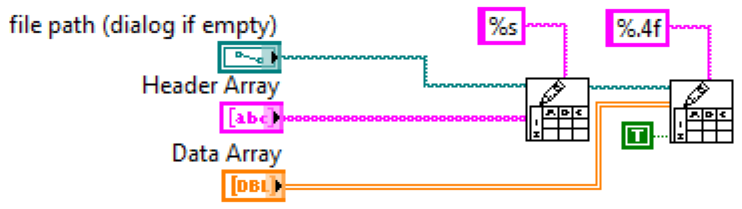


Figure 3.6 Save Data Files VI.

C. Producer Loop

C.1. Overview

As stated above, the primary function of the producer loop is to acquire data and pass them into the queueing system. These data can either be captured from the hardware or procured from a text document with a known format. Data are captured in data blocks whose sizes are dictated by the user and then sent to the consumer loop.

C.2. Collect Data VI

The roles of the Collect Data VI are (a) to determine if the data are to be gathered from a text document or acquired with hardware (Figure 3.7) and (b) to collect the data at the desired sampling rate (Figure 3.8). As illustrated in Figure 3.7 and Figure 3.8, the VI receives the buffer size, the sampling frequency, the file path (if applicable), a Boolean variable that indicates if the data are from a file or from hardware, a Boolean variable that indicates if the user wants to stop the program, and the file offset. The VI then returns the force, displacement, and EMG data as three separate arrays of dynamic data. If the data are collected with hardware, the VI simply uses a DAQ Assistant to gather the data from the various DAQ ports and scales these data into units of kilograms (force), centimeters (displacement), and microvolts (EMG). These data are passed into the queueing system as dynamic data. If data are collected from a file, the file path is passed into the Remove Header VI (see the next section), which identifies the start of the data and provides that file offset as an output. If the Remove Header VI has been called already this run, the VI passes the file offset through instead of recalculating it each time. Once the file offset is found, the Collect Data VI grabs the next set of data to be passed into the queueing system. The VI then converts the double-precision data into the dynamic data type by first converting the data to a waveform. (‘Dynamic’ data are used by LabVIEW as a general data type for data generation or data acquisition and the data can easily and immediately be graphed in LabVIEW using any of the basic graphing tools.) However, many built-in LabVIEW VIs do not accept the dynamic data type, so the data must be converted to a different data type prior to analysis. After the final conversion, the data are sent into the queueing system.

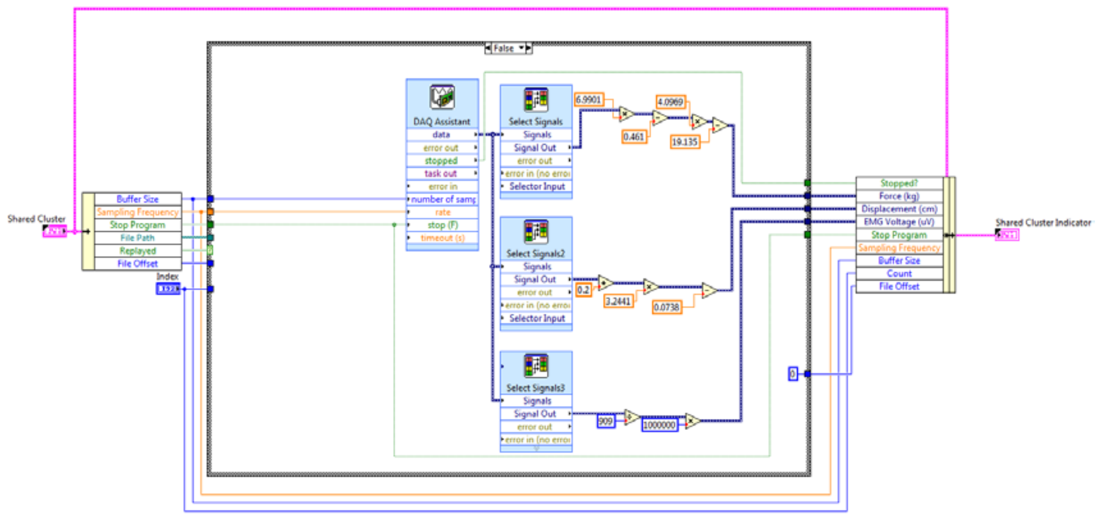


Figure 3.7 Collect Data VI acquiring data from hardware.

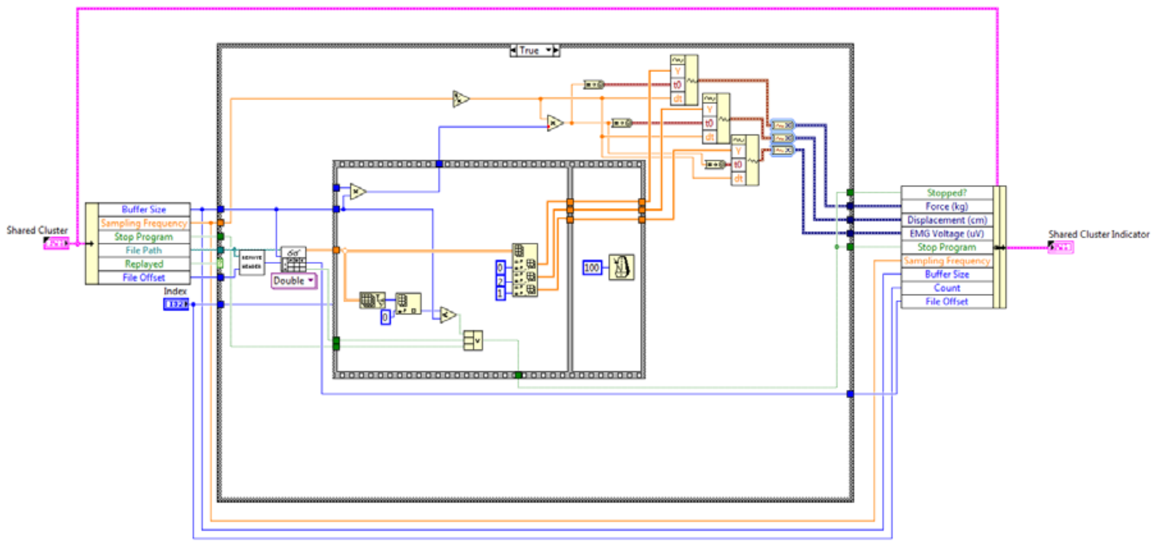


Figure 3.8 Collect Data CI acquiring data from data file.

C.2.1. Remove Header VI

The Remove Header VI removes any non-data text at the beginning of the file that resides at the given path. As illustrated in Figure 3.9, the VI receives the file path and the file offset. It then returns the header offset. The file offset input contains the position of the next character to be read in the file. This file offset is output by the Read From Spreadsheet File VI (which is native to LabVIEW), and its value updates each time the Read From Spreadsheet File VI is called. The first time the Remove Header VI is called, the VI determines the size of the header in terms of the number of characters and passes that value out as the header offset. For any subsequent call, the Remove Header VI passes through the file-offset input as the header offset, which is illustrated in Figure 3.10.

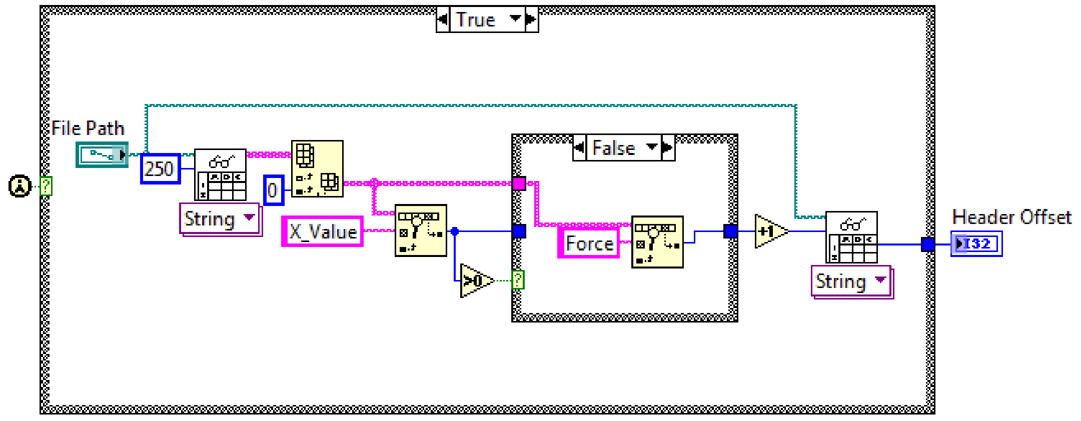


Figure 3.9 Remove Header VI removing the header.

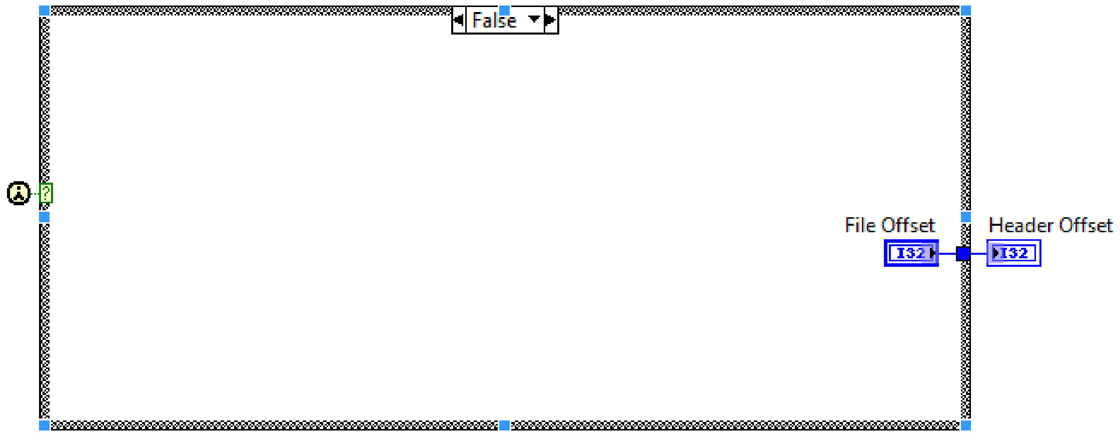


Figure 3.10 Remove Header VI passing the file offset through.

D. Consumer Loop

D.1. Overview

As stated in Section A, the role of the consumer loop is to receive the electromyographic data, identify the muscle bursts, and calculate the desired parameters of each muscle burst. As illustrated in Figure 3.11, the consumer loop smooths the incoming data and uses a threshold method to separate the muscle bursts from the rest of the signal. Once a muscle burst is found, the consumer loop then calculates its power spectrum (via an FFT), from which the median and mean power frequencies, skew, and kurtosis are then determined. The researcher’s screen is continuously updated with the results. The consumer loop is made up of multiple VIs: Baseline Calculation VI, Find Start & Stop Times VI, Power FFT VI, Mean and Median Power Frequencies VI, and Skew and Kurtosis VI. These VIs are discussed in more detail in the following sections.

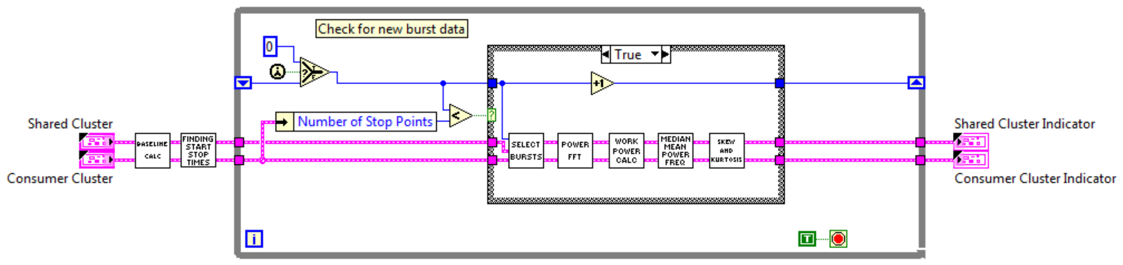


Figure 3.11 Consumer Loop VI – top level.

D.2. Baseline Calculation VI

The Baseline Calculation VI finds the baseline value of the force, displacement, and EMG data. As illustrated in Figure 3.12 and Figure 3.13, the VI receives the buffer size, the force, displacement, and EMG data, the time used to calculate the preliminary baseline (known as delay time), and the current iteration of the producer loop. The delay time represents a set number of buffers of data points for the system to collect before allowing the consumer loop to perform any analysis. This lets the system calculate baseline values for force, displacement, and EMG to improve the threshold for detecting muscle bursts. The VI returns the updated baseline array, the updated raw data array, and the signal threshold. The baseline array is the array that holds the “DC” values of the force, displacement, and EMG data. This array is essentially the result of a moving average calculation applied to a length of data equal to the delay multiplied by the buffer size. The raw data array holds a number of data values that can be up to delay-multiplied-by-buffer-size in length. This reduces the amount of data held in memory at any given time, which speeds up the execution of the consumer loop.

During the first call of the VI, the VI creates the raw data and baseline arrays, then inserts the current data into the raw data array. Each member of the baseline array is filled with the sum of the respective data values divided by the length of the moving average filter. While the current iteration of the producer loop is less than the specified delay time, the VI keeps appending values to the raw data array (from oldest to newest) and updates the baseline array by adding in the sum of the respective data values divided by the length of the moving average filter. Once the current count is greater than or equal to the delay length, the old data are removed from the raw data array and the new data are placed into the raw data array. The baseline EMG value is adjusted by subtracting off the old data points and adding in the new data points, both of which are divided by the full length of the moving average filter. The signal threshold is set equal to the baseline of the EMG multiplied by 1.5. This value was chosen using trial and error across several test sets. Currently, this value does not update itself, as that can cause problems when future sections of code compare previous burst data to the signal threshold. At this time, the force and displacement baseline values are not updated because they are not useful for later calculations.

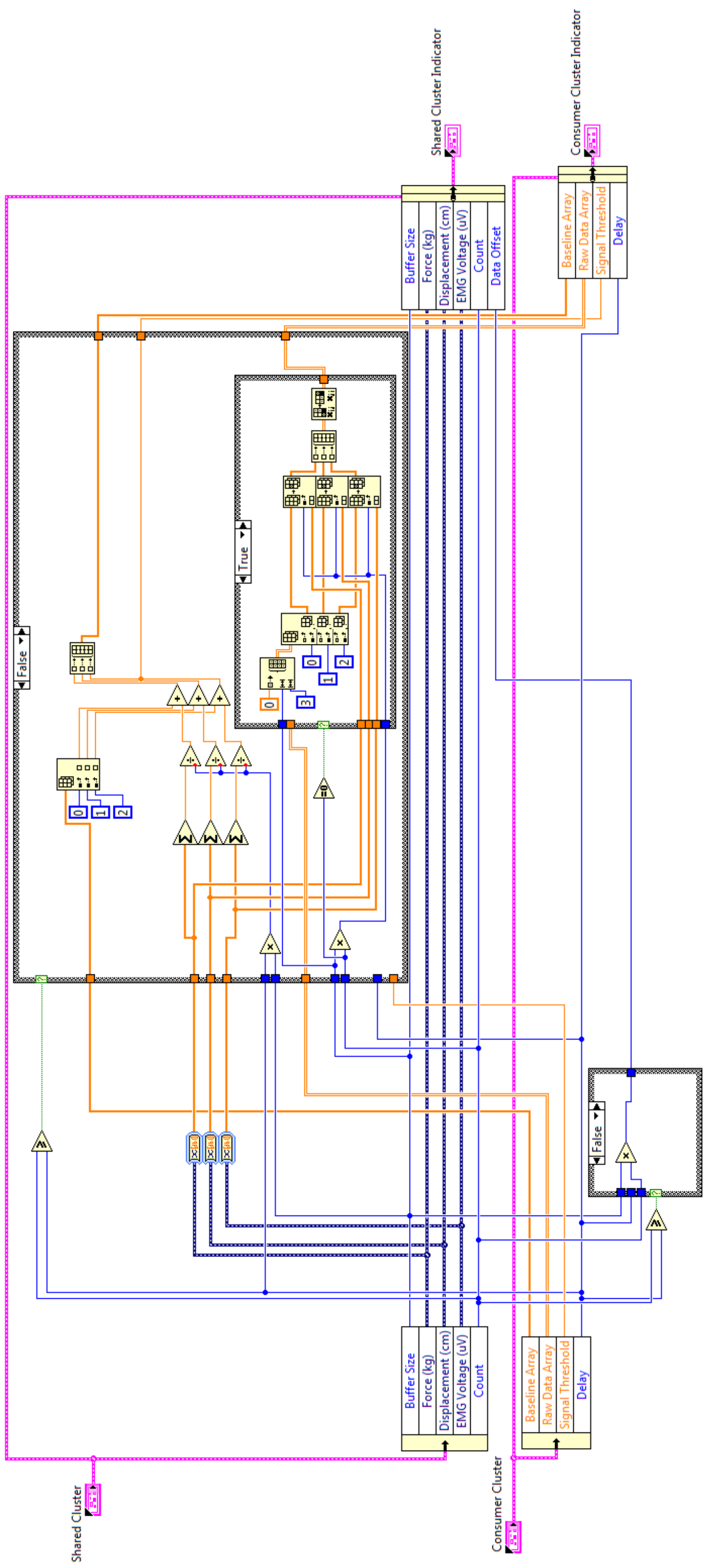


Figure 3.12 Baseline Calculation VI during the delay period.

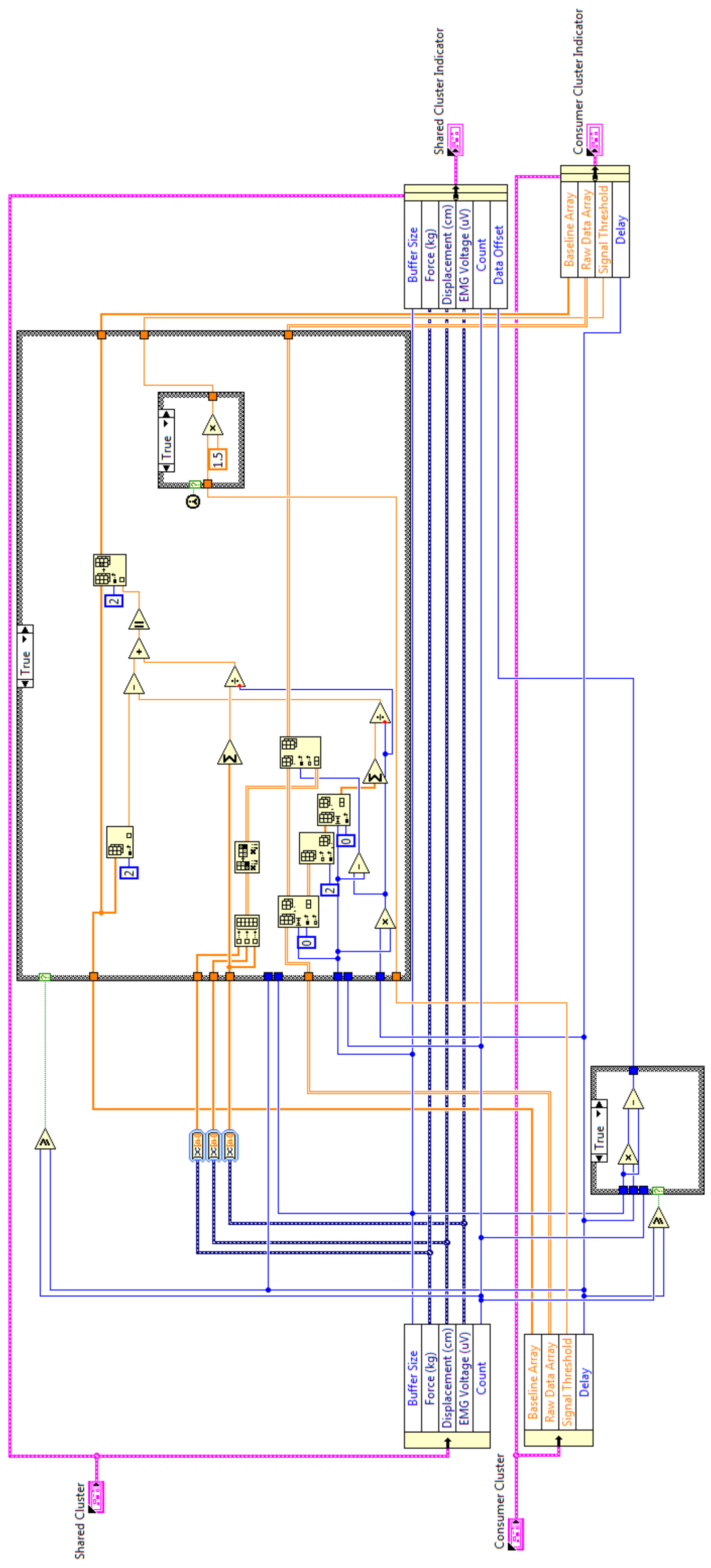


Figure 3.13 Baseline Calculation VI after the delay period.

D.3. Find Start & Stop Times VI

D.3.1. Overview

The Find Start & Stop Times VI determines the start and stop times of the muscle bursts during an experiment. As illustrated in Figure 3.14 and Figure 3.15, the VI receives the current raw data block and the DC value of the EMG found in the Baseline Calculation VI, then it returns start and stop points in an array format. The VI first filters the raw data using a 50-point-wide sliding median filter (to remove random outliers) followed by a 50-point-wide moving average filter (to remove broader-band noise). Next, the VI uses the filtered data set along with the DC value of the EMG to determine the location of any muscle bursts that have occurred. The filtering and location functions are contained in their own VIs: Prepare Signal VI and Find Muscle Bursts VI, respectively.

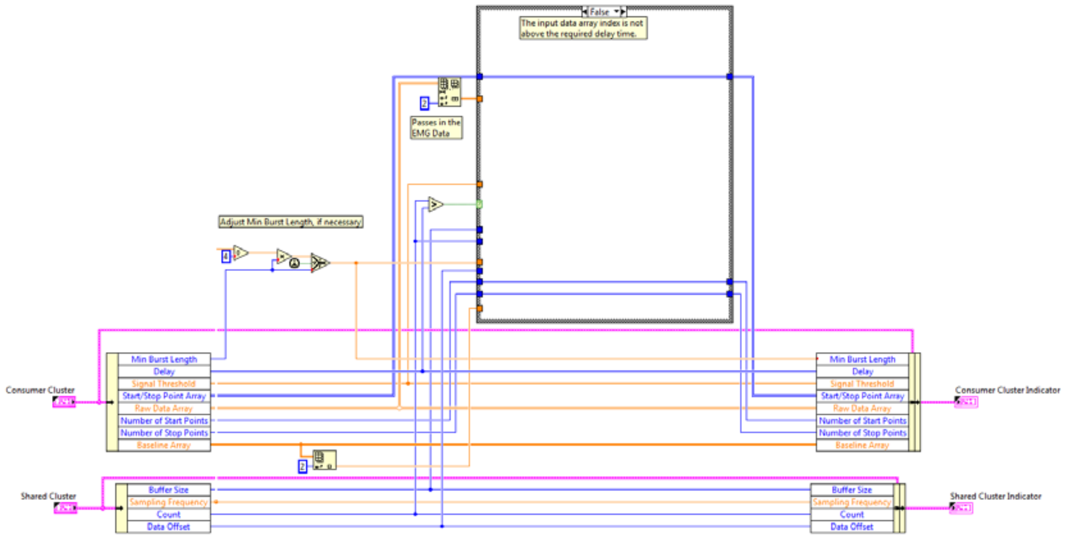


Figure 3.14 Find Start and Stop Times VI during the delay period.

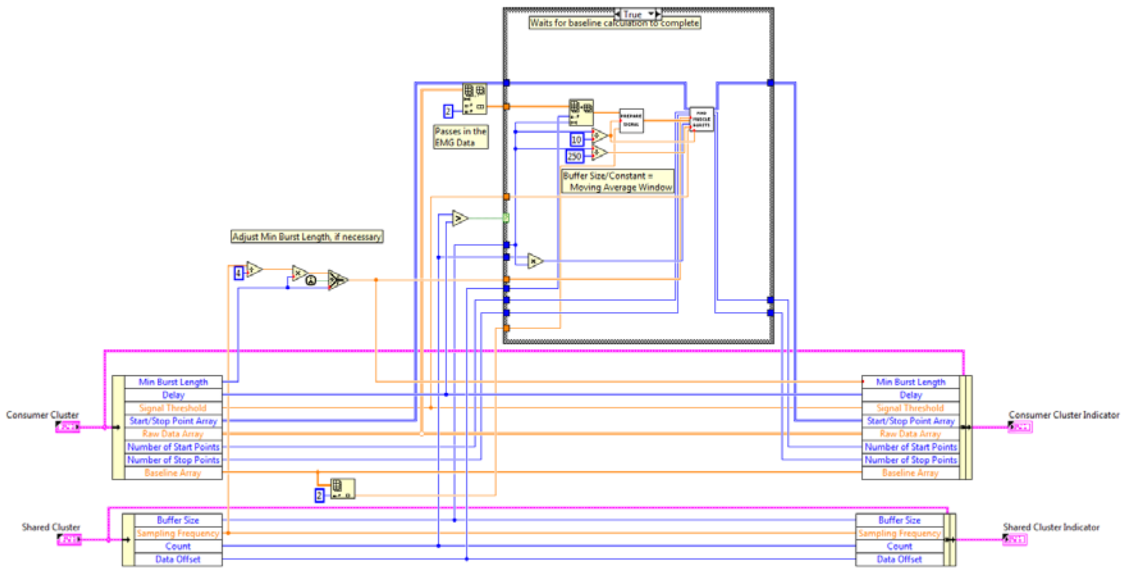


Figure 3.15 Find Start and Stop Times VI after the delay period.

D.3.2. Prepare Signal VI

The Prepare Signal VI (see Figure 3.16) receives raw data and conditions these data in a way that eliminates the startup times for the smoothing algorithms. The VI accomplishes this by holding onto a set of raw data from the previous block whose length corresponds to the combined sizes of the median filter and moving average filter windows.

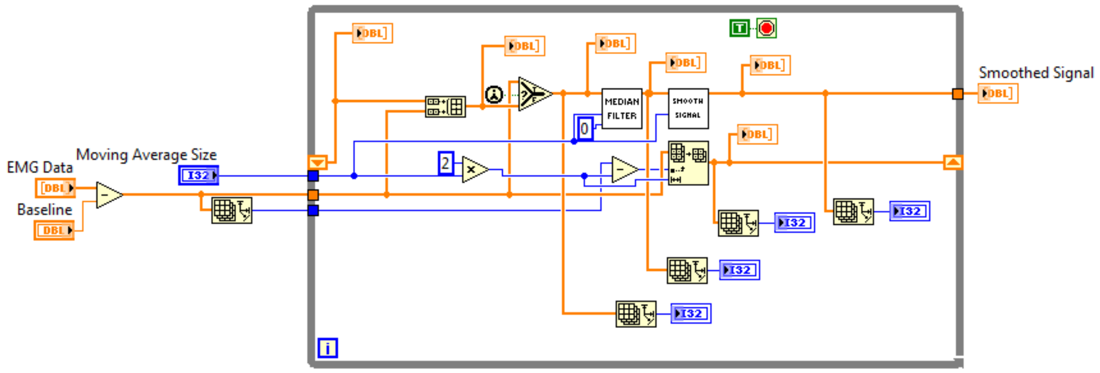


Figure 3.16 Prepare Signal VI

D.3.2.1. Median Filter VI

The Median Filter VI (see Figure 3.17) receives raw data from the Prepare Signal VI and returns data that have been smoothed using a 50-point-wide sliding median filter. The median filter’s window has a right rank of 0 and a left rank of the desired window size,

W. This means that the first W filtered data points will be zero, which represents the startup time of the filter. Once the data have passed through the median filter, the startup data are deleted. The filtered data are shifted to be time aligned with the start of the raw data and returned as the output of the Median Filter VI.

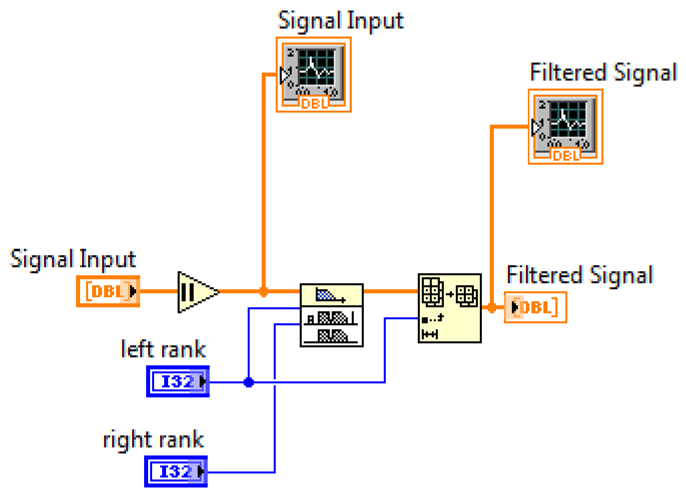


Figure 3.17 Median Filter VI.

D.3.2.2. Smooth Signal VI

The Smooth Signal VI (see Figure 3.18) receives data that have been output by the Median Filter VI described in the section above and returns data that have been smoothed using a 50-point-wide moving average filter. Before the data are filtered, they are rectified so that the smoothing operation more readily leads toward a waveform envelope that can be used to discern start/stop times for the individual EMG bursts. The VI then implements a moving average filter of size W to process the data. Again, this means that the first W filtered data points will be zeros that represent the startup time of the filter. These startup data are again deleted as above. The filtered data are shifted to be time aligned with the start of the raw data, and then they are returned as the output of the Smooth Signal VI.

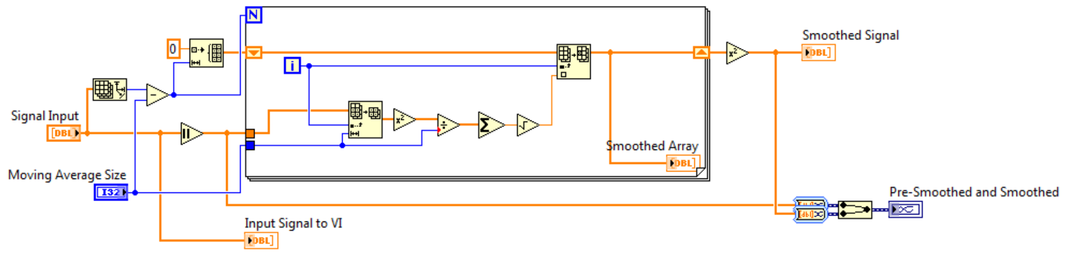


Figure 3.18 Smooth Signal VI

D.3.3. Find Muscle Bursts VI

D.3.3.1. Overview

The Find Muscle Bursts VI (see Figure 3.19, Figure 3.20, Figure 3.21, and Figure 3.22) receives the smoothed signal from the Prepare Signal VI, a different window size for burst detection, the signal threshold found in the Baseline Calculation VI, the start/stop array, the minimum burst length, the number of both start and stop points, and the index of the block of raw data relative to the start of the experiment. It then returns the start and stop points as well as an updated count for the start and stop points. The VI first calculates a new signal threshold using the smoothed data through the Update Threshold VI. The VI also contains a `for()` loop that allows it to step through the smoothed signal with a window of size S while it holds the start/stop array and the number of start and stop points. A Boolean shift register keeps track of the burst activity, allowing identification of a false start or stop point. A shift register of type 'double' also keeps track of the previous S-1 data points from the previous data block. This allows the system an uninterrupted stream of data, meaning that the system still looks for muscle burst start and stop times across consecutive data blocks. Once inside the loop, the VI concatenates the old data to the front of the new data and then sends them on to the start/stop decision tree. The VI then compares the data points against the updated signal threshold. To avoid undesired outcomes such as missing stop points, start points only a few data points away, etc., the VI keeps the previous signal threshold to compare with the old data. The decision tree checks for muscle burst activity by looking at the first and last data points of the S-sized window and checking if the data cross the signal threshold. The four conditions checked are 1) if the first and last points are both below the threshold, 2) if the first point is below the threshold and the last point is above the threshold, 3) if the first point is above the threshold and the last point is below the threshold, and 4) if the first and last points are both above the threshold.

As illustrated in Figure 3.19, the first condition indicates that no burst activity exists in the current S EMG points. The muscle burst activity variable is then set to ‘false.’ As illustrated in Figure 3.20, the second condition indicates the potential start of a muscle burst. The loop then saves the potential start point and increments the number of start points variable. The muscle burst activity variable is then set to ‘true.’ As illustrated in Figure 3.21, the third condition indicates the potential end of a muscle burst. The VI compares the length of the potential burst with the minimum burst length. If the burst is too short, the start point is removed and the number of start points variable is decremented. If the burst length is longer than the minimum burst length, the stop point is added to the start/stop point array, and the number of stop points variable is incremented. The muscle burst activity variable is then reset to ‘false.’ As illustrated in Figure 3.22, the fourth condition indicates that potential burst activity is underway. The muscle burst activity variable is then set to ‘true.’

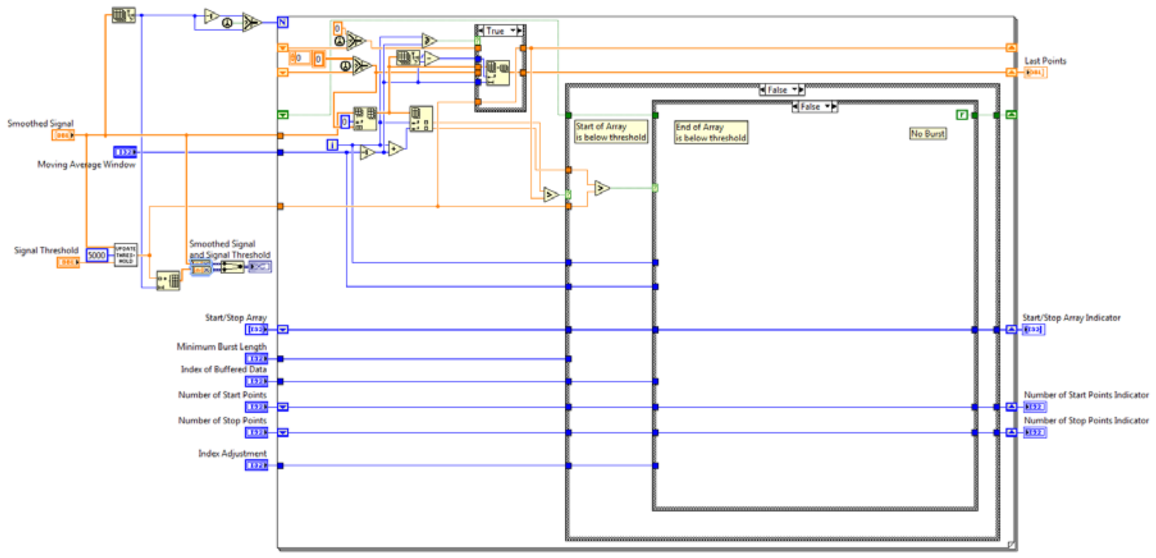


Figure 3.19 Find Muscle Bursts VI condition 1.

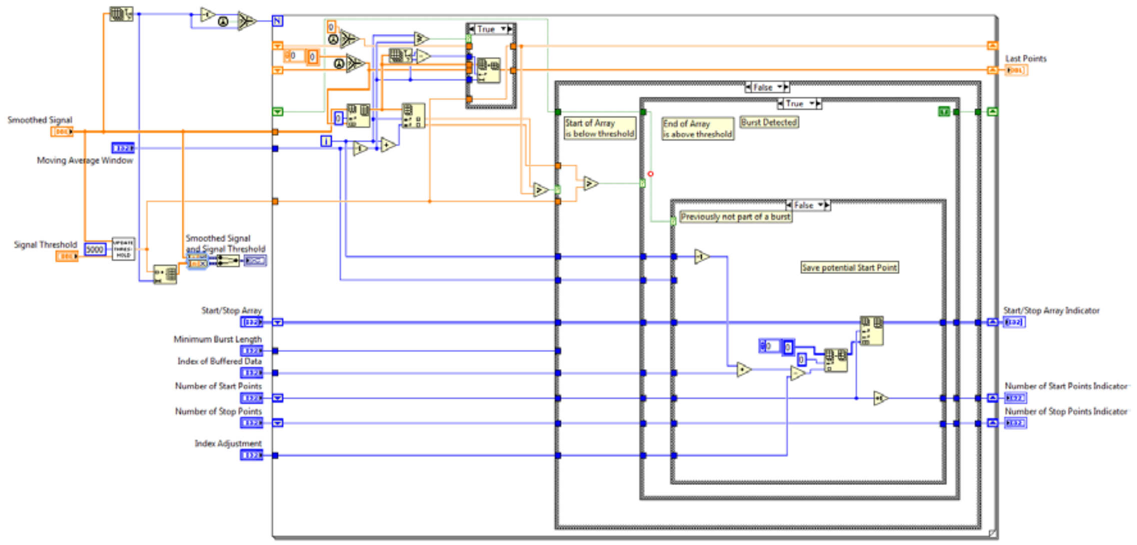

Smoothed Signal
Moving Average Window
Signal Threshold
5000
Smoothed Signal and Signal Threshold
Start/Stop Array
Minimum Burst Length
Index of Buffered Data
Number of Start Points
Number of Stop Points
Index Adjustment
True
False
True
False
Start of Array is below threshold
End of Array is above threshold
Burst Detected
Previously not part of a burst
Save potential Start Point
Last Points
Start/Stop Array Indicator
Number of Start Points Indicator
Number of Stop Points Indicator

Figure 3.20 Find Muscle Bursts VI condition 2.

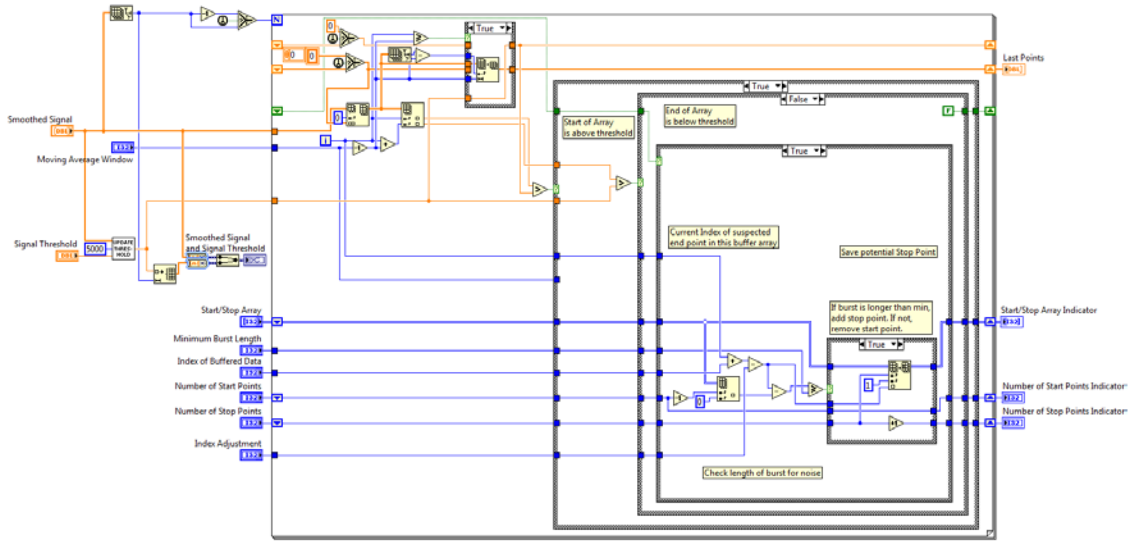

Smoothed Signal
Moving Average Window
Signal Threshold
5000
Smoothed Signal and Signal Threshold
Start/Stop Array
Minimum Burst Length
Index of Buffered Data
Number of Start Points
Number of Stop Points
Index Adjustment
True
True
False
True
True
Start of Array is above threshold
End of Array is below threshold
Current Index of suspected end point in this buffer array
Save potential Stop Point
If burst is longer than min, add stop point. If not, remove start point.
Check length of burst for noise
Last Points
Start/Stop Array Indicator
Number of Start Points Indicator
Number of Stop Points Indicator

Figure 3.21 Find Muscle Bursts VI condition 3.

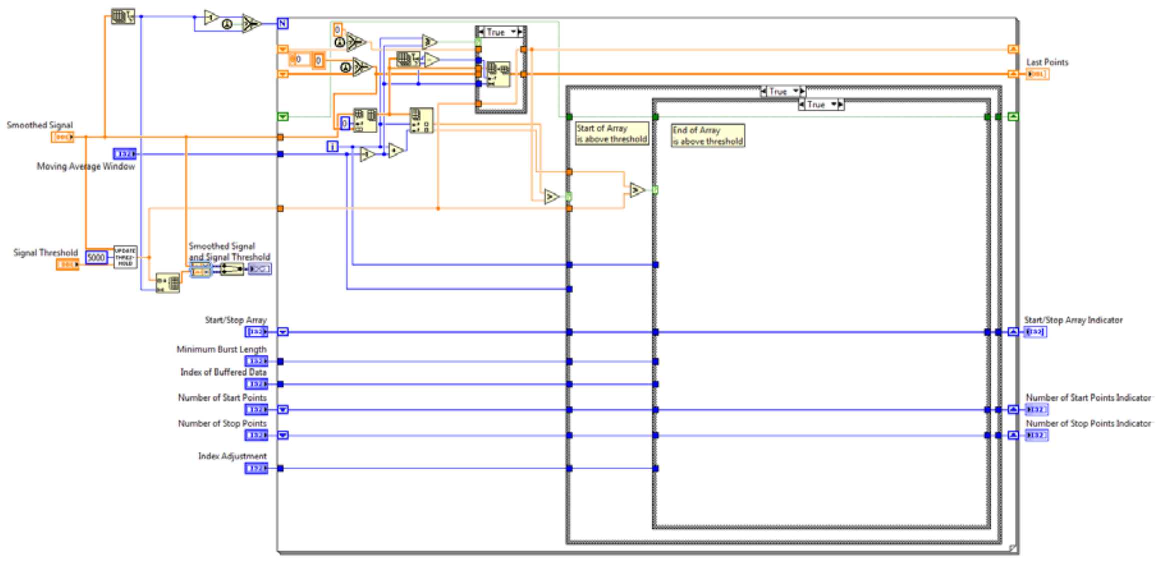

Smoothed Signal
Moving Average Window
Signal Threshold
Smoothed Signal and Signal Threshold
Start/Stop Array
Minimum Burst Length
Index of Buffered Data
Number of Start Points
Number of Stop Points
Index Adjustment
Start of Array is above threshold
End of Array is above threshold
True
Last Points
Start/Stop Array Indicator
Number of Start Points Indicator
Number of Stop Points Indicator

Figure 3.22 Find Muscle Bursts VI condition 4.

D.3.3.2. Update Threshold VI

The Update Threshold VI scales the threshold found with the raw data to be better suited for the smoothed data. As illustrated in Figure 3.23, Figure 3.24, and Figure 3.25, the VI receives an array of smoothed data, the number of points over which to calculate, and the raw data signal threshold. It then returns the updated threshold. The VI has two shift registers that store the size of the holder array and the holder array itself. This allows the VI to perform calculations over a range of data that is larger than the buffer size. To update the threshold, the VI looks at two Boolean outputs: the first call function's output and the output of a comparison function that determines whether the placeholder array is filled. The first call function is a native LabVIEW function that determines if the current VI has been called since the start of the overall program. These two outputs relate to four possible outcomes, three of which are useful: 1) both outputs are false, 2) the first call function is true, and the second output is false, and 3) the first call function is false and the second output is true. The fourth outcome, where both outputs are true, would result if the desired size of the placeholder array was smaller than the buffer size. Since this is an undesirable result, it has been hardcoded not to happen.

The first condition indicates that the placeholder array is not quite full and that this is not the first call of the VI. The VI then updates the size of the placeholder array and inserts the new smoothed data into the placeholder array. The VI then takes the median value of the array and adds it to the raw data array offset to get the updated threshold value. Adding in the raw data signal threshold allows the median threshold to be placed just above the peak of the noisy smoothed data. The second condition indicates that the placeholder array is not quite full and that this is the first call of the VI. The VI then creates a placeholder array and adds the median of the smoothed signal to the raw data threshold to create the updated threshold. The third condition indicates that the placeholder array is full and that this is not the first call of the VI. The VI then adjusts the size of the placeholder array and, if necessary, removes the old data from the array, adds in the new data from the smoothed data array, and then takes the median of the placeholder array and adds that value to the raw data threshold, which becomes the updated threshold.

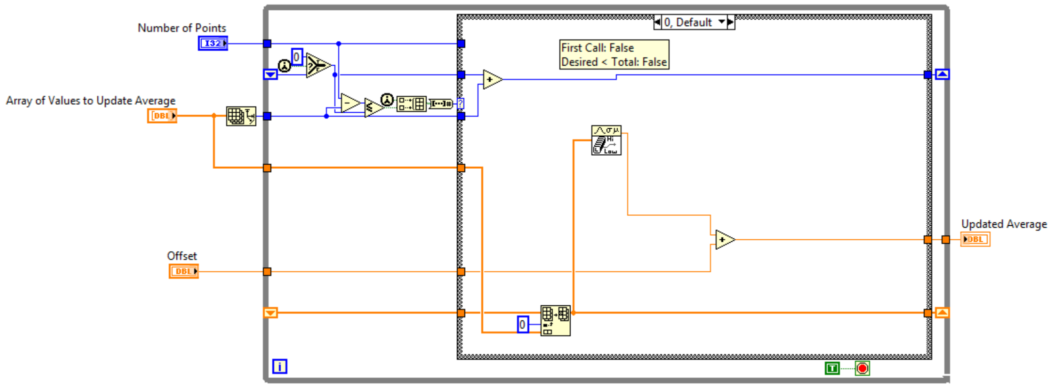


Figure 3.23 Updated Threshold VI condition 1.

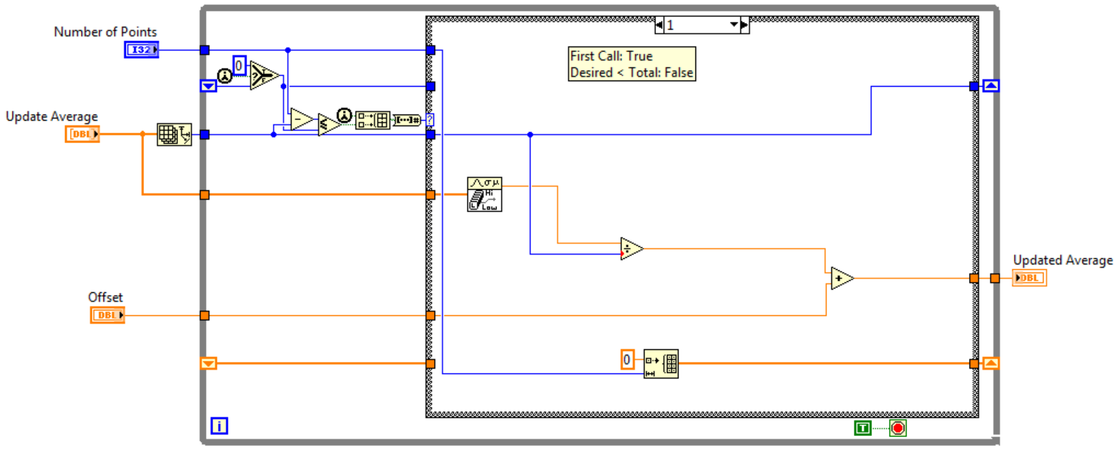


Figure 3.24 Updated Threshold VI condition 2.

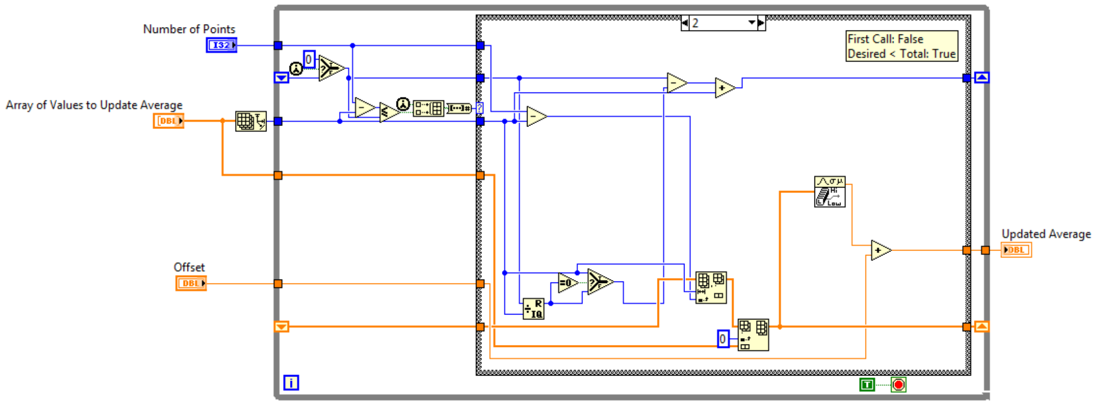


Figure 3.25 Updated Threshold VI condition 3.

D.4. Select Bursts VI

The Select Bursts VI selects the current muscle burst and calculates the RMS value of the current muscle burst. As illustrated in Figure 3.26, the VI receives the sampling frequency, the buffer size, the current iteration of the producer loop, the current iteration of the consumer loop, the baseline array, the raw data array, the start/stop point array, the delay time, the number of stop points, and the RMS array. It then returns the burst data array, the two-dimensional burst-and-time-data array, and an updated RMS array. First, the VI decides which burst is the current burst and places that section of the raw data block into the burst data array. The VI also calculates when the burst occurs relative to the beginning of the experiment. It then uses the sampling frequency to generate the time values (starting at 0) and places the time values alongside the burst values into the burst-and-time-data array. Finally, the VI calculates the RMS value of the current burst and places it into the RMS array.

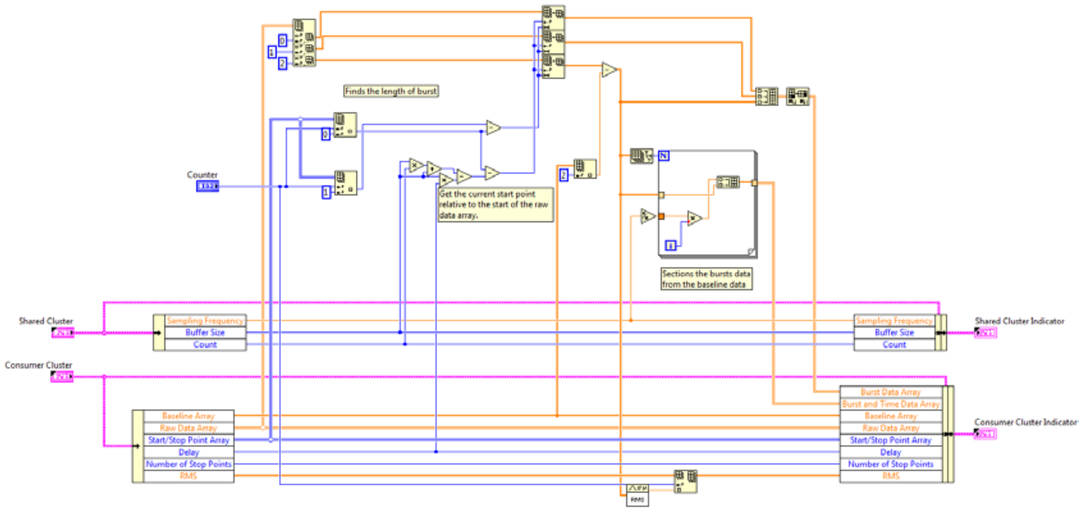

Finds the length of burst
Counter
Get the current start point relative to the start of the raw data array.
Sections the bursts data from the baseline data
Shared Cluster
Sampling Frequency
Buffer Size
Count
Shared Cluster Indicator
Consumer Cluster
Baseline Array
Raw Data Array
Start/Stop Point Array
Delay
Number of Stop Points
RMS
Burst Data Array
Burst and Time Data Array
Consumer Cluster Indicator

Figure 3.26 Select Bursts VI.

D.5. Power FFT VI

D.5.1. Overview

The Power FFT VI calculates the power spectrum of each muscle burst and splits the FFT spectrum into high (95+ Hz) and low (5-30 Hz) frequency bands. As illustrated in Figure 3.27 and Figure 3.28, the VI receives the current muscle-burst-and-time-data array, the FFT array (magnitudes squared), and the sampling frequency. The VI returns an updated FFT array, the power spectrum, and the high and low frequency bands of the FFT spectrum. First, the burst data must be converted into a waveform data type so that the native FFT spectrum VI can compute the real and imaginary parts of the burst coefficients. The FFT is then computed, and the real and imaginary parts are combined (making a complex value for each frequency) and then squared to create the power coefficients. The VI then takes the spectrum and splits it into the corresponding bands of interest. Since the LabVIEW software is unable to enlarge an array if a larger array is inserted into a current array, the Insert Large Row Array VI checks to see if the new spectrum is longer (in terms of the number of coefficients) than the previous spectrum and adjusts the array length if necessary.

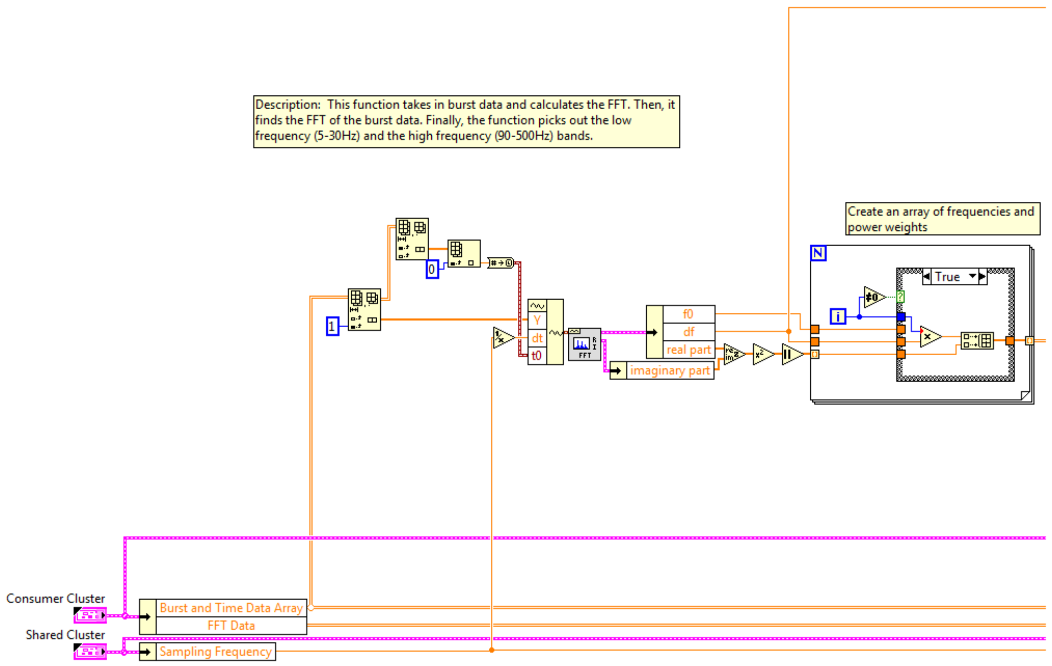

Description: This function takes in burst data and calculates the FFT. Then, it finds the FFT of the burst data. Finally, the function picks out the low frequency (5-30Hz) and the high frequency (90-500Hz) bands.
Create an array of frequencies and power weights
Consumer Cluster
Shared Cluster
Burst and Time Data Array
FFT Data
Sampling Frequency
f0
df
real part
imaginary part
True

Figure 3.27 Power FFT VI – FFT calculation.

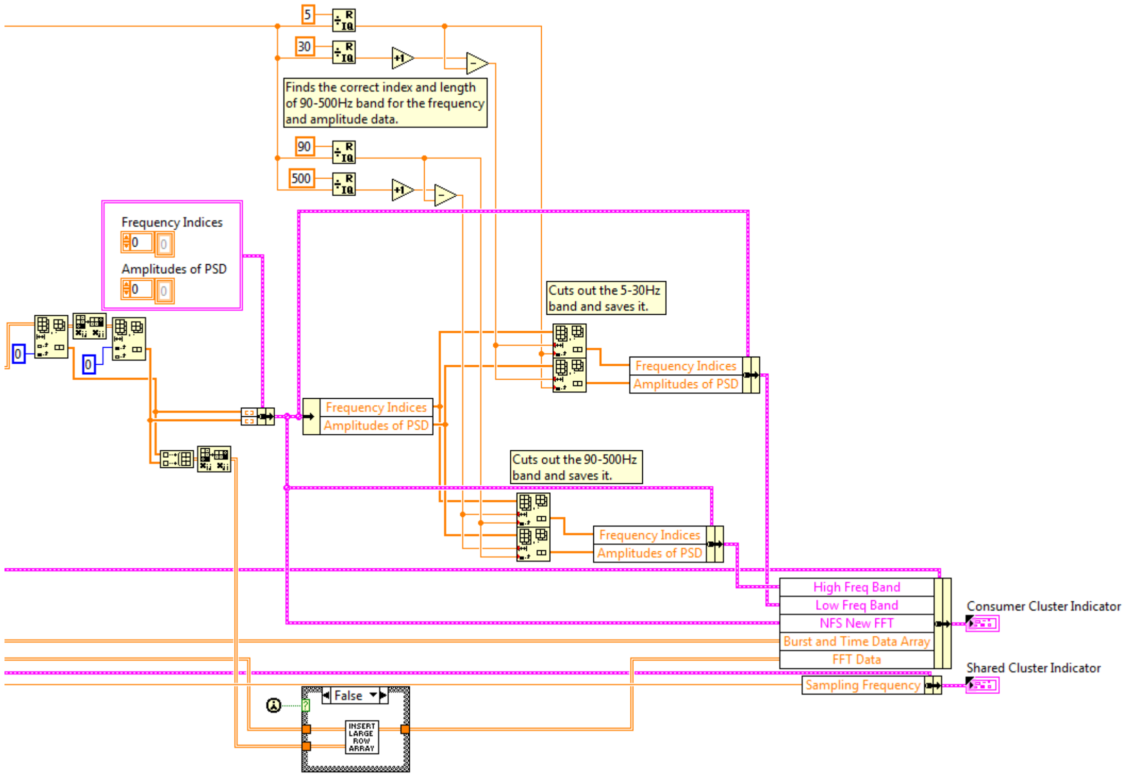

Finds the correct index and length of 90-500Hz band for the frequency and amplitude data.
Frequency Indices
Amplitudes of PSD
Cuts out the 5-30Hz band and saves it.
Cuts out the 90-500Hz band and saves it.
High Freq Band
Low Freq Band
NFS New FFT
Burst and Time Data Array
FFT Data
Sampling Frequency
Consumer Cluster Indicator
Shared Cluster Indicator
False
INSERT LARGE ROW ARRAY

Figure 3.28 Power FFT VI – frequency banding.

D.5.2. Insert Large Row Array VI

The Insert Large Row Array VI determines if any rows need to be added to the FFT array due to the greater length of an incoming spectrum. As illustrated in Figure 3.29 and Figure 3.30, the VI receives the array that contains all of the previous FFT values (frequencies and powers). It then returns an array that has the new FFT values inserted as two columns (frequencies and powers) to the end of the FFT array. First, the VI finds the sizes of the two incoming arrays and determines which has the highest number of rows. If the previous FFT array has a greater length, the VI simply inserts two new columns and places the new FFT values into those columns. If the new FFT array is longer, the VI creates a new array that has a length equal to the new FFT array, and the number of columns is kept equal to the previous FFT array's columns plus two. The VI then inserts the previous FFT values into the new array and then places the new FFT values into the array. This preserves the FFT values and prevents LabVIEW from truncating the incoming FFT values.

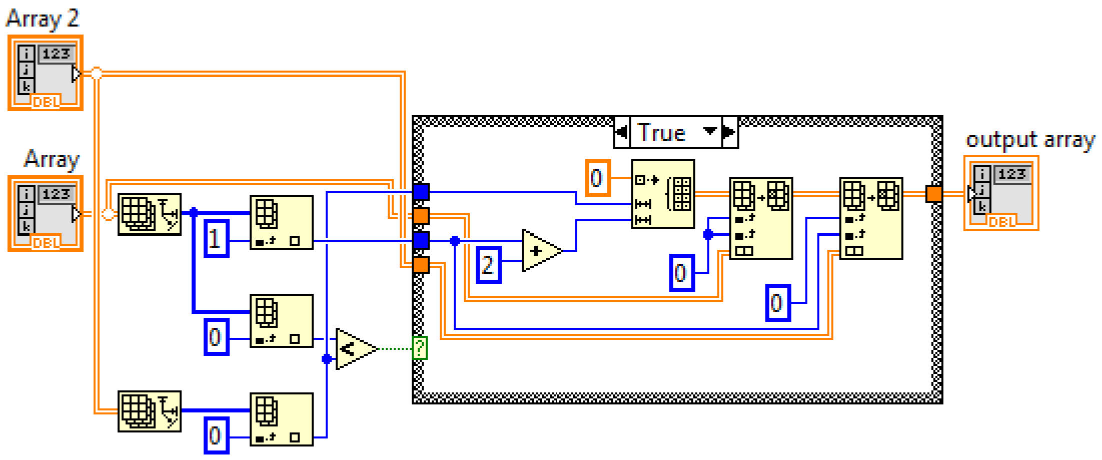


Figure 3.29 Insert Large Row Array VI with a larger new row.

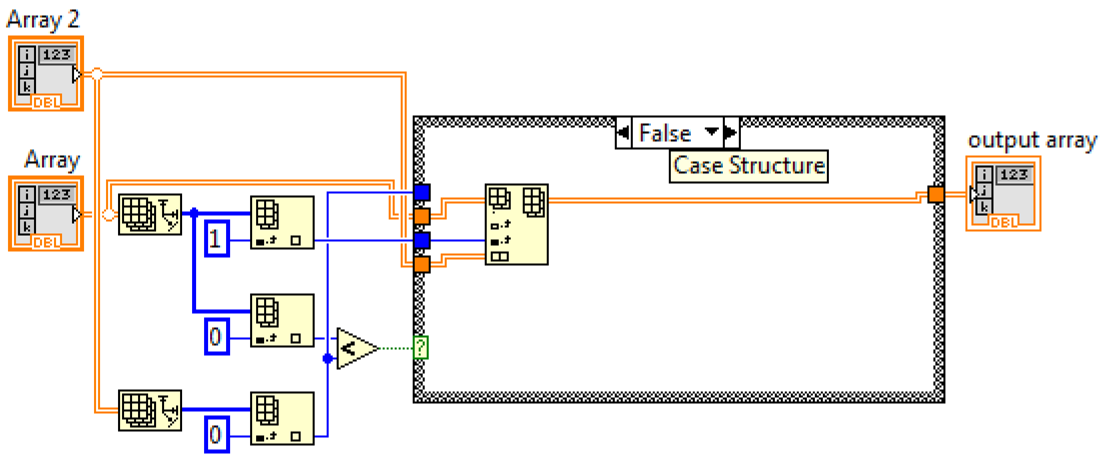


Figure 3.30 Insert Large Row Array VI with a smaller new row.

D.6. Mean and Median Power Frequency Calculation VI

D.6.1. Overview

The Mean and Median Power Frequency Calculation VI calculates the mean and median power frequencies (MPF and MDF, respectively) for a given power spectrum – see Figure 3.31. Currently, the VI calculates the MPF and MDF of the full spectrum (0-500 Hz), the low band of frequencies (5-30 Hz), and the high band of frequencies (95+ Hz). To simplify the coding structure, this VI calls another VI for each frequency band.

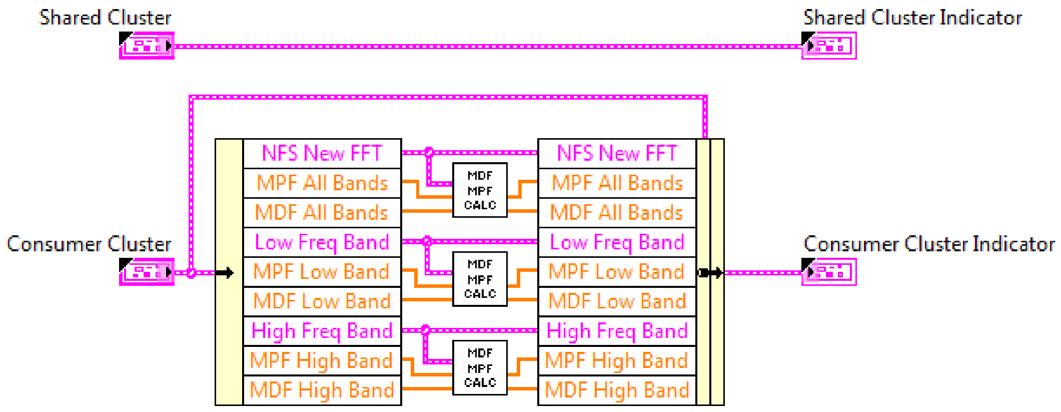


Figure 3.31 Mean and Median Power Frequency VI – top level.

D.6.2. MDF MPF Calculation VI

The MDF MPF Calculation VI calculates the mean and median power frequencies for a spectrum. As illustrated in Figure 3.32, the VI receives the FFT values for a frequency band as well as the mean and median power frequency arrays. It then returns the updated mean and median power frequency arrays. The equations used to calculate these values are given below:

mean frequency = $\frac{\sum_{n=f_l}^{f_h} n*|FFT(n)|}{\sum_{n=f_l}^{f_h} |FFT(n)|}$, where n is the frequency index of the FFT coefficients

and

$$\text{median frequency} = \sum_{n=f_l}^{f_h} |FFT(n)| > 0.5 * \sum |FFT(n)|,$$

where n is a frequency index, $|FFT(n)|$ represents the magnitude of the Fast Fourier Transform [19] of the EMG burst, and f_l and f_h are the indices that relate to the lowest and highest frequencies, respectively.

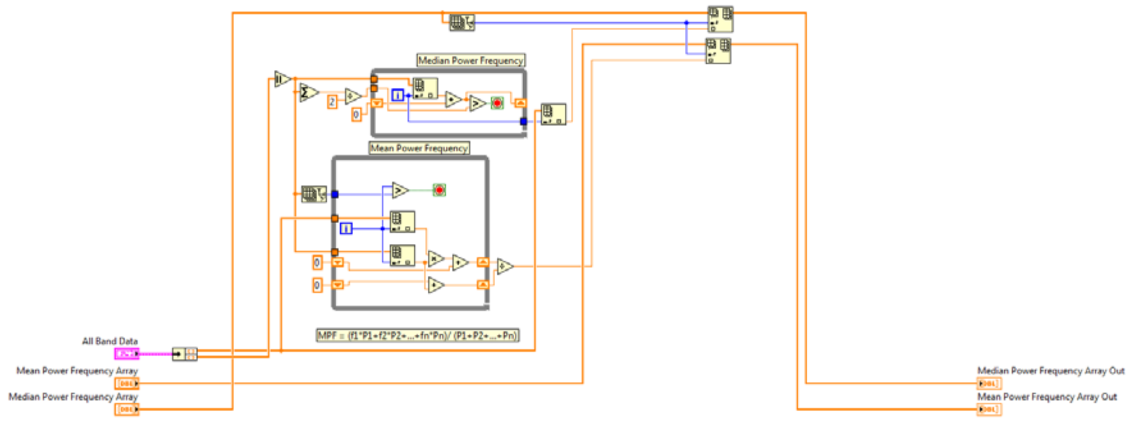

Median Power Frequency
Mean Power Frequency
MPF = (f1*P1+f2*P2+...+fn*Pn)/ (P1+P2+...+Pn)
All Band Data
Mean Power Frequency Array
Median Power Frequency Array
Median Power Frequency Array Out
Mean Power Frequency Array Out

Figure 3.32 Mean and Median Power Frequency VI calculation.

D.7. Skewness and Kurtosis Calculation VI

The Skewness and Kurtosis Calculation VI calculates the skewness and kurtosis of the spectrum of the current muscle burst. As illustrated in Figure 3.33, the VI receives the current FFT values (frequencies and power) and returns the skewness and kurtosis of that spectrum. The equations used to calculate these values are given below [25]:

$$skewness(i) = \frac{\sum_{n=1}^{N}(Y_i(n)-\bar{Y})^3}{s^3}/N$$

and

$$kurtosis(i) = \frac{\sum_{n=1}^{N}(Y_i(n)-\bar{Y})^4}{s^4}/N \ ,$$

where i is a burst index, n is a frequency index, $Y_i(n)$ is a spectral power, $\bar{Y}$ is the average spectral value, s is the standard deviation of the spectral values, and N is the number of coefficients in the spectrum.

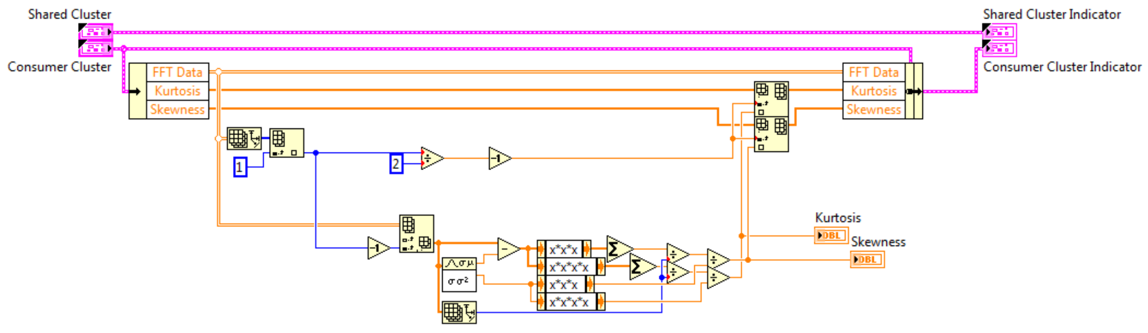


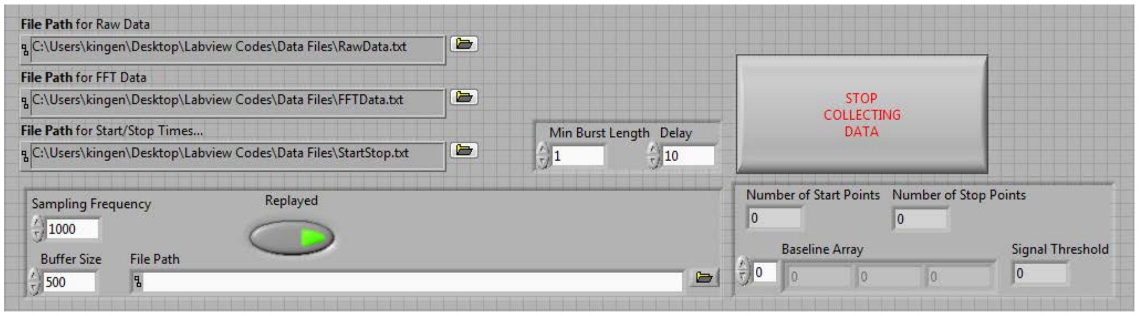
Figure 3.33 Skewness and Kurtosis VI.

F. Researcher Display

The researcher display provides the researcher with an interface to (a) manipulate acquisition variables prior to the experiment and (b) observe the raw and calculated data during the experiment. As illustrated in Figure 3.34, Figure 3.35, Figure 3.36, and Figure 3.37, the researcher display contains three main parts: 1) the input variables panel, 2) the raw data panel, and 3) the calculated data panel.

Figure 3.34 depicts the input-variables panel on the researcher display. The file paths in the upper-left-hand corner point to the directories where the raw data, FFT coefficients, and start/stop data files are stored. The bottom-left and middle windows hold fields for input variables. The bottom-right window displays output variables of interest to the researcher for troubleshooting purposes. The button in the upper-right-hand corner is used to stop the program safely.

The input variables control the sampling frequency (Hz), buffer size (number of data points), minimum burst length (seconds), delay (number of buffers), replayed versus live data, and file path for replayed experiments if troubleshooting a replayed data set (number of data points). The ‘Sampling Frequency’ is simply the number of samples the system will acquire per second during a live experiment. For a replayed experiment, the system pulls ‘Buffer Size’ number of data points and treats them as being sampled at the sampling frequency. For a live experiment, the ‘Buffer Size’ refers to how many data points the hardware can hold before these data will be transferred to the software. For a replayed experiment, the buffer size refers to the number of data points the software will take from the data file at a time before sending these data into the software analysis VIs. The ‘Replayed’ Boolean variable determines if the system will pull data from a file or accept data from hardware. If the researcher desires to pull data from a file, the ‘File Path’ variable points to that experiment file. ‘Min Burst Length’ refers to the smallest expected muscle burst length (seconds) for both live and replayed experiments. The ‘Delay’ variable refers to how many buffers of data points to take in as baseline values for both live and replayed experiments.



File Path for Raw Data: C:\Users\kingen\Desktop\Labview Codes\Data Files\RawData.txt

File Path for FFT Data: C:\Users\kingen\Desktop\Labview Codes\Data Files\FFTData.txt

File Path for Start/Stop Times...: C:\Users\kingen\Desktop\Labview Codes\Data Files\StartStop.txt

Min Burst Length: 1 Delay: 10

Sampling Frequency: 1000

Buffer Size: 500

File Path:

Number of Start Points: 0 Number of Stop Points: 0

Baseline Array: 0 | 0 | 0 | 0

Signal Threshold: 0

Figure 3.34 **Researcher display input variables.**

Figure 3.35 depicts the raw data panel of the researcher display. The panel displays the force data (kg), the displacement data (cm), the EMG data (µV), and the current count of the experiment. The force, displacement, and EMG data are either acquired from hardware or from data files depending on whether the experiment is live or replayed. The count variable displays how many buffers of data have been passed through the system during either a live and replayed experiment.

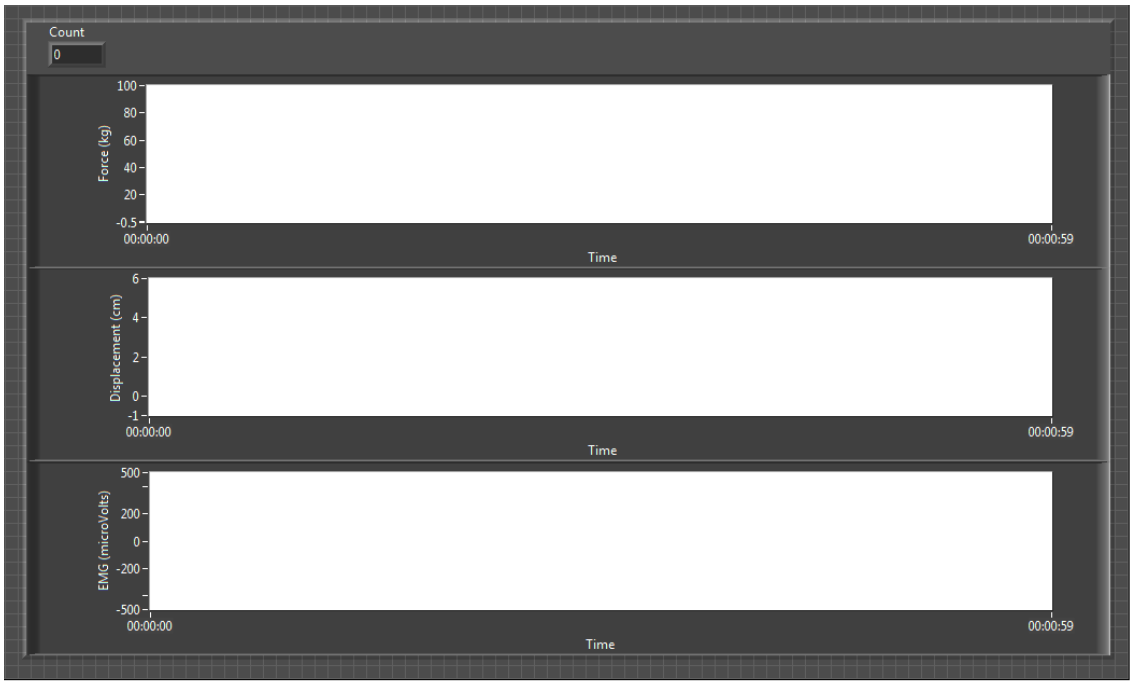


Figure 3.35 Researcher display raw data (inverted color).

Figure 3.36 presents the top half of the calculated data display:

- Upper left plot – current burst data.
- Upper center plot – burst RMS values.
- Upper right plot – EMG power spectrum for the current muscle burst.
- Middle left plot – mean power frequency calculated from the entire EMG amplitude spectrum for each individual burst.
- Center plot – median power frequency calculated from the entire EMG amplitude spectrum for each individual burst.
- Middle right plot – displacement data for the current muscle burst.
- Bottom left plot – mean power frequency calculated from the low band (5-30 Hz) of the current EMG amplitude spectrum.
- Bottom center plot – median power frequency calculated from the low band (5-30 Hz) of the current EMG amplitude spectrum.
- Bottom right plot – force data for the current muscle burst.

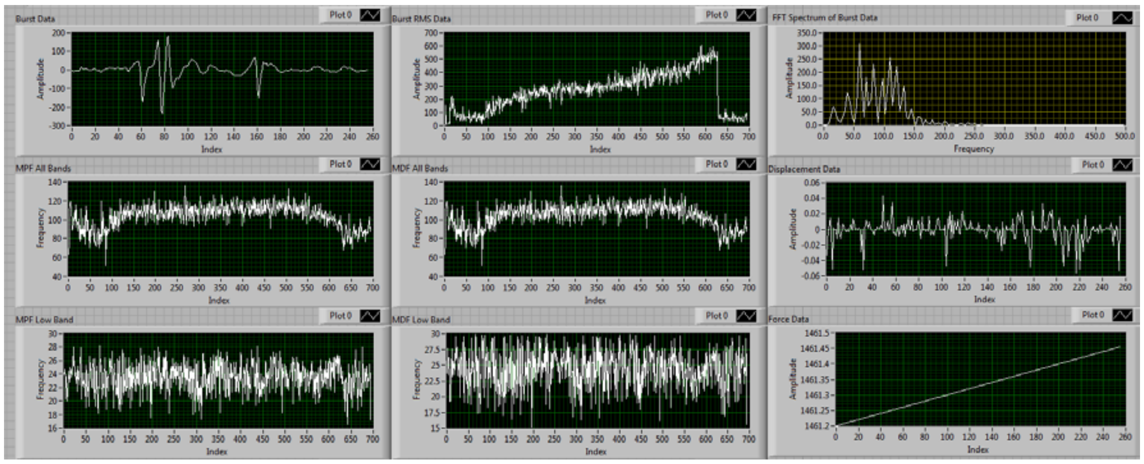


Figure 3.36 Researcher display calculated data – top half.

Figure 3.37 presents the bottom half of the calculated data display:

- Upper left plot – mean power frequency calculated from the high band (95+ Hz) of the current EMG amplitude spectrum.
- Upper right plot – median power frequency calculated from the high band (95+ Hz) of the current EMG amplitude spectrum.
- Bottom left plot – kurtosis value for each muscle burst.
- Bottom right plot – skewness value for each muscle burst.

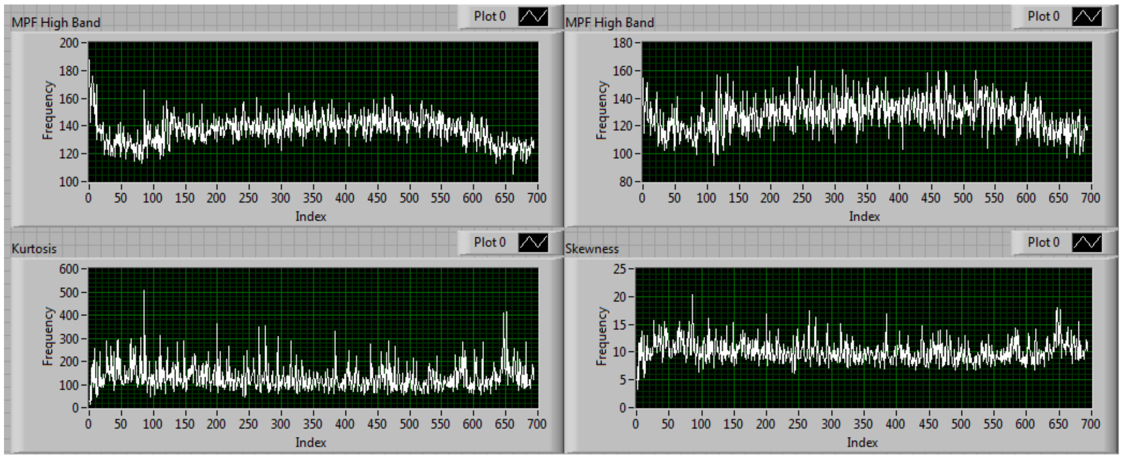


Figure 3.37 Researcher display calculated data – bottom half.

G. Subject Display

As illustrated in Figure 3.38, the subject display contains a simple slider tool native to the LabVIEW software. The purpose of the subject display is to give the subject a visual representation of their required grip sequence. To start the VI, the researcher first clicks on the white run arrow at the top of the window to allow the VI to create the array necessary to fill and empty the slider. Next, the researcher turns on the audio file that gives an audio cue to the subject that signals when to grip and release. The researcher then clicks on the start button in synchronization with the audio signal. This gives the subject both audio and visual cues for grip timing.

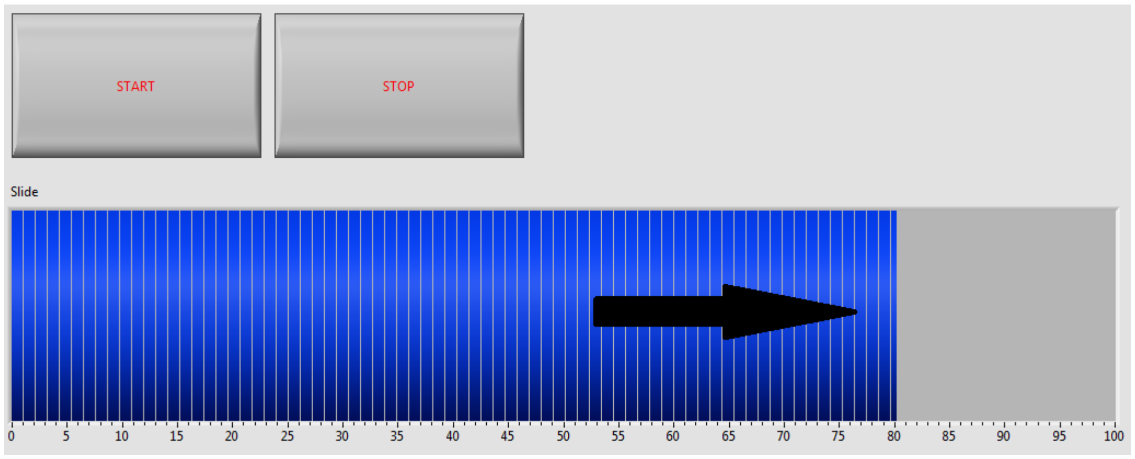


Figure 3.38 Subject display.

CHAPTER 4: BURST MOVIES

A. Overview

EMG burst shape, spectral content, and parameters change from burst to burst as a subject fatigues, and these changes can be difficult to visualize. To address this issue, a MATLAB script was developed that creates “burst movies” which allow one to pictorially replay the results of an EMG-based fatigue experiment. Each updated frame in a movie depicts data from a new EMG burst, and when a movie is played over and over, it allows a researcher to visualize trends in time-domain EMG burst shape, frequency-domain magnitude and power spectra shapes, and the associated parameters as fatigue occurs. This MATLAB script gives the researcher the flexibility to test new fatigue parameters using old data produced by the real-time analysis system. The script takes in raw data files and FFT coefficient files populated by the real-time system, and it produces two movies that document changes in EMG signals, EMG spectra, and fatigue parameters: a Time-Domain Movie and a Frequency-Domain Movie. Each movie displays six plots in a three-row by two-column grid. Before the video frames are captured, the script makes an initial pass through the input files to determine optimal axis ranges for each plot. The content of these two types of movies is described in more detail below. It should be noted that the MATLAB script displays both magnitude and power frequency spectra while the LabVIEW code only displays power frequency spectra. Further, the researcher can manually edit out non-experiment bursts in the MATLAB script, while the LabVIEW script displays all muscle bursts that appear after the delay time and are above threshold.

B. Time-Domain Movie

The time-domain movie is designed to display the time-domain EMG and fatigue parameters. These parameters include the traditional fatigue parameters, RMS and iEMG, of each muscle burst as well as the more non-traditional fatigue parameters, zero crossing rate and average rectified value. A normalization factor is also included to fit the EMG muscle burst into a *y*-axis range of [-1,1]. An example of the final frame of a time-domain burst movie is depicted in Figure 4.1.

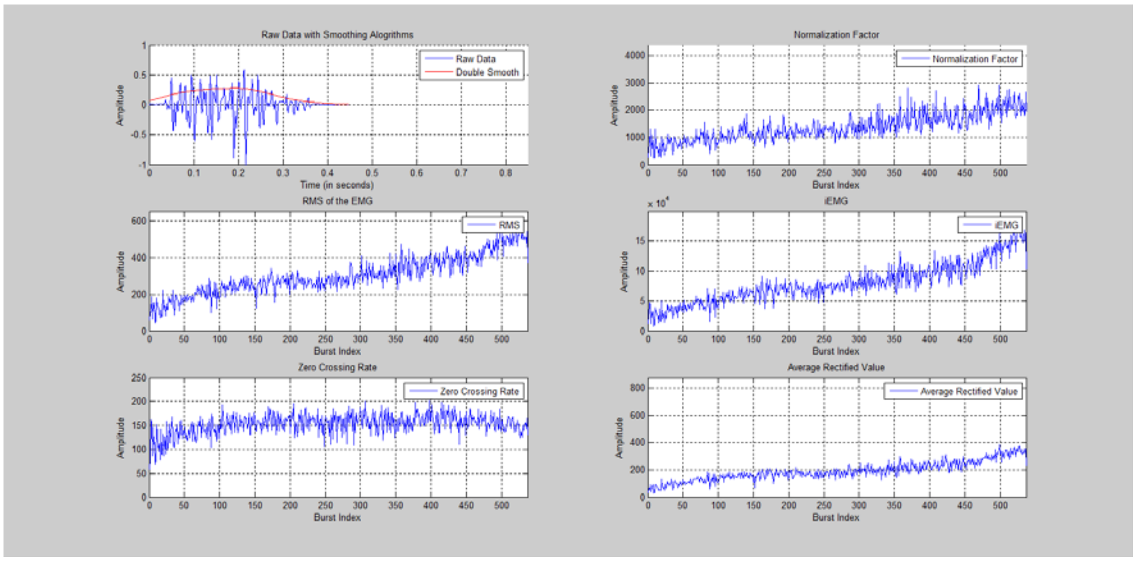


Figure 4.1 Time-domain burst movie frame.

The upper-left plot in Figure 4.1 depicts the normalized muscle burst data along with a smoothed curve that represents the general shape of the burst. The smoothed curve is created by rectifying the raw EMG burst and then processing the burst with a sliding median filter followed by a moving average filter, both of length 101 points.

The upper-right plot depicts the normalization factor trend for the entire session. This is the factor used to scale each raw muscle burst into a [-1,1] amplitude window. It is important to note that this factor is only relevant in constant-force experiments.

The middle-left plot depicts the trend of the root-mean square (RMS) value of each muscle burst for the entire session. The RMS value is found using the following equation:

$$RMS\ Value = \frac{1}{length(currentBurstData)} * \sqrt{\sum currentBurstData^2}$$

The middle-right plot depicts the trend of the integrated EMG, or iEMG for the entire session. This value is found for each burst by integrating the rectified EMG burst data.

The lower-left plot depicts the trend of the zero crossing rate for the entire session. The zero crossing rate is found for each burst by calculating how many times the raw EMG burst crosses zero. This value is useful when determining the muscle conduction velocity [13].

The lower-right plot depicts the trend of the average rectified value for the entire session. The average rectified value is found for each burst by taking the mean of the rectified EMG muscle burst data [14]:

$$Average\ Rectified\ Value(i) = \frac{1}{N}\sum_{n=1}^{N}|x_i(n)|,$$

where *i* is a burst index and *n* is an index variable used to step through a sequence of *N* values that comprise a burst.

C. Frequency Domain Movie

The Frequency Domain movie is designed to display burst-to-burst changes in the EMG magnitude and power spectra as well as the fatigue parameters associated with the frequency domain. These parameters include traditional fatigue parameters (mean and median power frequency) of each muscle burst as well as non-traditional fatigue parameters – total spectral magnitude, skewness, and kurtosis. The skewness and kurtosis of each burst was calculated from a Gaussian curve fit to the respective power spectrum. This slightly reduced the amount of jitter in the skewness and kurtosis values when compared to calculations performed using the power spectra directly. An example of the final frame of an FFT coefficient burst movie is depicted in Figure 4.2.

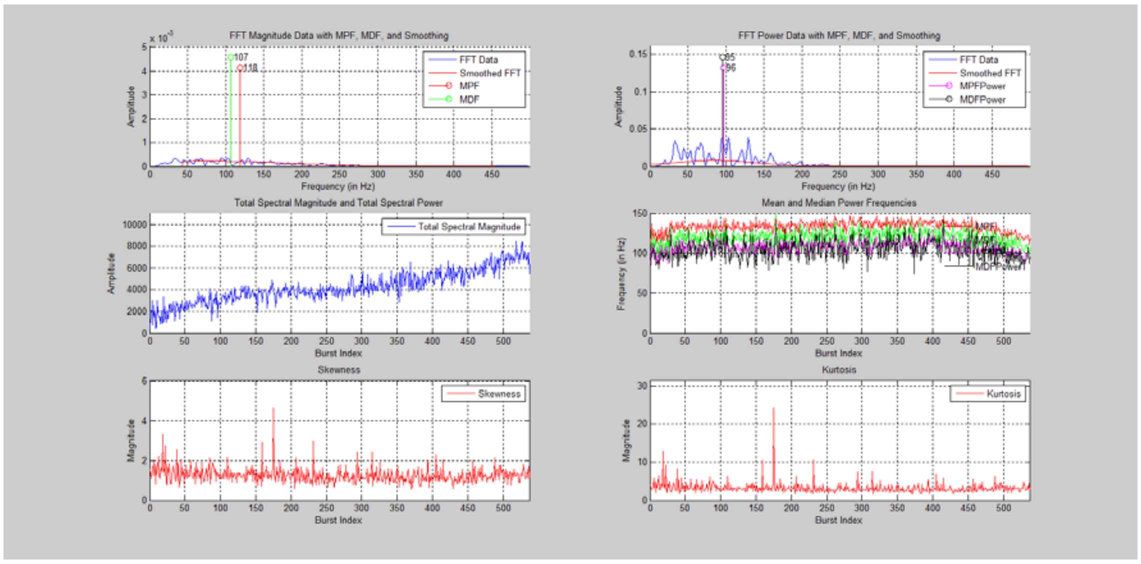


Figure 4.2 FFT coefficient burst movie frame.

The upper-left plot in Figure 4.2 depicts the magnitude spectrum for the current burst along with the mean and median frequencies of the spectrum. The magnitude spectrum is found for each burst by taking the square root of the power spectrum produced by the real-time analysis system. The plot also displays a smoothed version of the magnitude spectrum, obtained by applying a 101-point sliding median filter followed by a 101-point moving average filter. The mean and median frequencies are found for each burst using the following formulas [13]:

$$\text{mean frequency} = \frac{\sum_{n=f_l}^{f_h} n * |FFT(n)|}{\sum_{n=f_l}^{f_h} |FFT(n)|}$$

and

$$\text{median frequency} = \sum_{n=f_l}^{f_h} |FFT(n)| > 0.5 * \sum |FFT(n)|$$

where n is a frequency index, $|FFT(n)|$ represents the magnitude of the Fast Fourier Transform [19] of the EMG burst, and f_l and f_h are the indices that relate to the lowest and highest frequencies, respectively.

The upper-right plot displays the power spectrum of the current burst along with the mean and median power frequencies and a Gaussian fit to the power spectrum. The power spectrum is read from the FFT coefficient file stored by the real-time analysis system. The mean and median power frequencies are found using the same equations used to find the mean and median magnitude frequencies given above, but with a focus on power instead of magnitude. The Gaussian fit is found by utilizing a `gaussfit.m` MATLAB script written by Przemyslaw Baranski [26]– see the Appendix for the script and the copyright/licensing information. The Baranski script takes in the desired data set and uses an iterative LMS method to determine the average and standard deviation of the symmetric Gaussian curve that best fits that data set.

The middle-left plot depicts the trend of the total spectral magnitude for the entire session. The total spectral magnitude is the sum of the square roots of the FFT coefficients of each individual burst.

The middle-right plot depicts the trend of all of the mean and median frequencies found for the upper-left and upper-right plots for the entire session.

The lower-left plot depicts the skewness of the Gaussian fit found in the upper-right plot. The skewness is determined for each muscle burst using the expression

$$skewness(i) = \frac{\sum_{n=1}^{N}(Y_i(n)-\bar{Y})^3}{s^3}/N$$

where i is a burst index, $Y_i(n)$ is a spectral value, $\bar{Y}$ is the average spectral value of the fitted Gaussian curve, s is the standard deviation of the fitted Gaussian curve, n is a frequency-domain index, and N is the number of frequency-domain coefficients. The above formula for skewness is referred to as the Fisher-Pearson coefficient of skewness [25]. Some software programs adjust (multiply) this coefficient by $\frac{\sqrt{N(N-1)}}{N-1}$ to correct for smaller sample sizes. This adjustment was not used because the sample sizes were sufficiently large enough ($N > 250$) that the correction would not induce a noticeable effect on the skewness coefficient.

The lower-right plot depicts the kurtosis of the Gaussian fit found in the upper-right plot. Kurtosis is described as the peaked-ness of a curve. The kurtosis is found for the power spectrum of each muscle burst using expression

$$kurtosis(i) = \frac{\sum_{n=1}^{N}(Y_i(n)-\bar{Y})^4}{s^4}/N,$$

where the variables in the expression are the same variables used in the skewness calculation described above.

CHAPTER 5: ANALYSIS RESULTS

A. Summary of Experiments

The constant force experiments consisted of a subject gripping the hand-forearm ergometer in intervals of two seconds up, one second down, and one second for rest. The subjects contracted at a frequency of 20 contractions/minute at a fixed power output of 85% of their maximum power, which was found on a previous date. The subject was paced by an audio file. This process was repeated either with constant or increasing force until the subject could no longer grip the two bars of the ergometer together. The increasing force experiment consisted of a subject pedaling on a cycle ergometer. The subjects pedaled on the ergometer while the power output increased by 60 Watts/min starting from 25 Watts. This continued until the subjects reached exhaustion. While the cycle ergometer data (described first below) were clearly acquired with a setup that is different than the hand-forearm ergometer, these data were included here as a means to verify the correct operation of the burst identification and parameter calculation code.

B. Observations from Data

The main goal of this work was to build a real-time hand-forearm fatigue analysis system based off of the current post-processing model. Logically, the post-processing system can be used as a baseline (i.e., a comparison standard) to gauge the accuracy of the real-time system by way of several test data sets. With this in mind, the real-time system was applied to four experimental data sets and evaluated by tracking (a) the number of bursts detected during each experiment and (b) the accuracy of burst identification over three experimental time spans chosen for each experiment.

Test 1 – Increasing Force Experiment

For this first test, the subject pedaled on a cycle ergometer at a starting power output of 25 Watts. The researcher then increased the power output by 60 Watts at each one-minute interval until the subject reached exhaustion. As noted in Figure 5.1, the sampling frequency was set to 1 kHz, the data set was replayed through the system, and the buffer size was set to 500 data points. The start and stop points are unequal due to the subject engaging their muscles as the researcher ended the completed experiment. Figure 5.2 illustrates that (a) the RMS value peaks as the subject fatigues, (b) the mean and median power frequencies for the broadband spectrum dip, and (c) the mean and median power frequencies for the low band (5-30 Hz) stay constant throughout the experiment. The RMS value drops sharply toward the end of the experiment due to subject movement while the system was still running. Note that since the experiment was replayed using data files of an unsupported format (e.g., data gathered from a previous version of the software, where two different files were used to store force/displacement and EMG data), the force and displacement data shown are incorrect. Also note that the mean and median frequencies on the LabVIEW screen represent power rather than magnitude. Figure 5.3 illustrates that the mean and median power frequencies in the high band (95+ Hz) hold constant during the experiment. The dip at the end is due to post-experiment movement. The skewness and kurtosis parameters stay relatively constant.

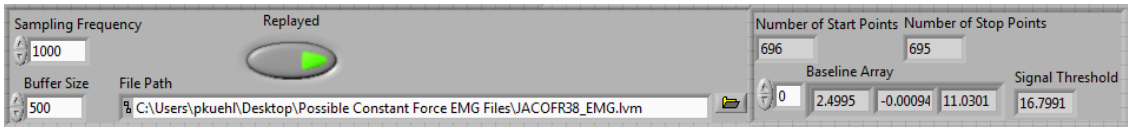

Sampling Frequency: 1000
Replayed
Buffer Size: 500
File Path: C:\Users\pkuehl\Desktop\Possible Constant Force EMG Files\JACOFR38_EMG.lvm
Number of Start Points: 696
Number of Stop Points: 695
Baseline Array: 0 | 2.4995 | -0.00094 | 11.0301
Signal Threshold: 16.7991

Figure 5.1 Input variables for Test 1.

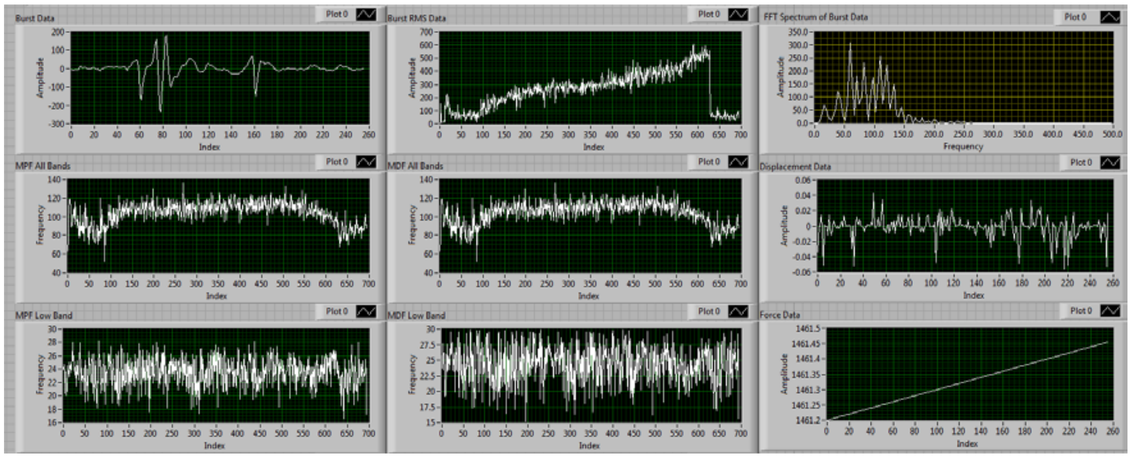


Figure 5.2 Top half of the output panels for Test 1.

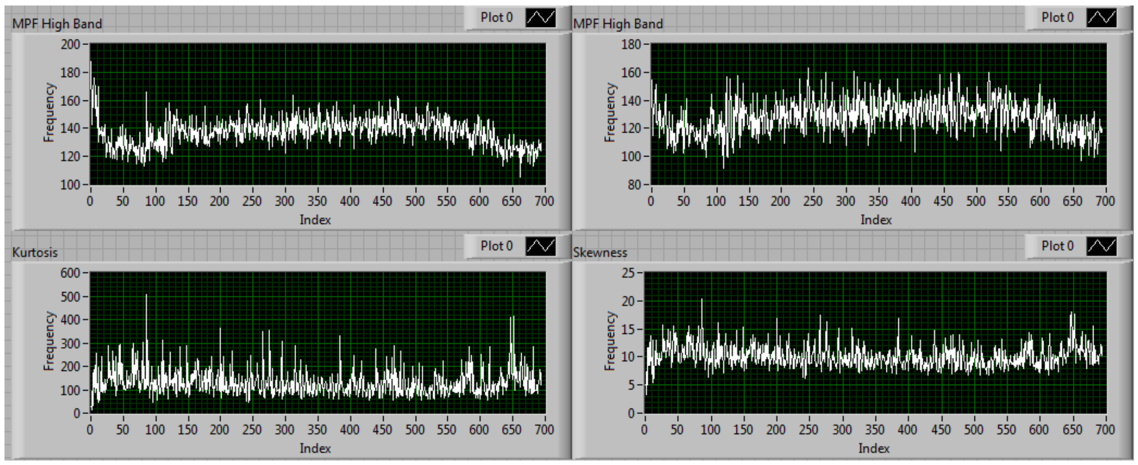
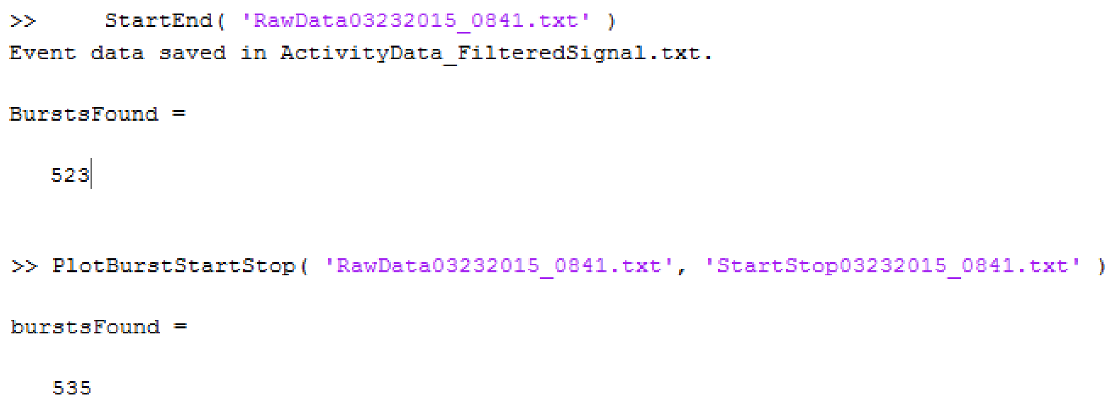


Figure 5.3 Bottom half of the output panels for Test 1.

As noted in Figure 5.4, the real-time system found more bursts during the experiment than the post-processing system. Even so, when only the total number of bursts is considered, the relative error of the real-time system when compared to the post-processing model would be 2.2%.



```
>>     StartEnd( 'RawData03232015_0841.txt' )
Event data saved in ActivityData_FilteredSignal.txt.

BurstsFound =

   523


>> PlotBurstStartStop( 'RawData03232015_0841.txt', 'StartStop03232015_0841.txt' )

burstsFound =

   535
```

Figure 5.4 Number of bursts found using the post-processing system (top) versus the real-time system (bottom).

Figure 5.5 displays a comparison between the start and stop times for Test 1 over the whole experiment; the post-processing results are on the top and the real-time results are on the bottom. Generally, both systems found a sufficient amount of muscle bursts and there are no noticeable sections of experiment data where either system did not find muscle bursts. Figure 5.6 and Figure 5.7 help to gauge the burst-to-burst accuracy for the post-processing system versus the real-time system by comparing start/stop points for the individual bursts for the beginning of Test 1. While the peaks can be easily seen, the threshold used in the post-processing approach was unable to pick them out. The real-time system exhibited a similar result. For the first two blocks of ten seconds, the real-time system identifies more muscle bursts than the post-processing system. However, in the last ten-second block, the post-processing system identifies more muscle bursts. Figure 5.8 and Figure 5.9 address the accuracy of the post-processing system versus the real-time system for the middle of Test 1. During this time window, the post-processing system easily identified all of the muscle bursts, as did the real-time system. Figure 5.10 and Figure 5.11 address the accuracy of the post-processing system versus the real-time system for the end of Test 1. During this time window, the post-processing system easily identified all of the muscle bursts, as did the real-time system.

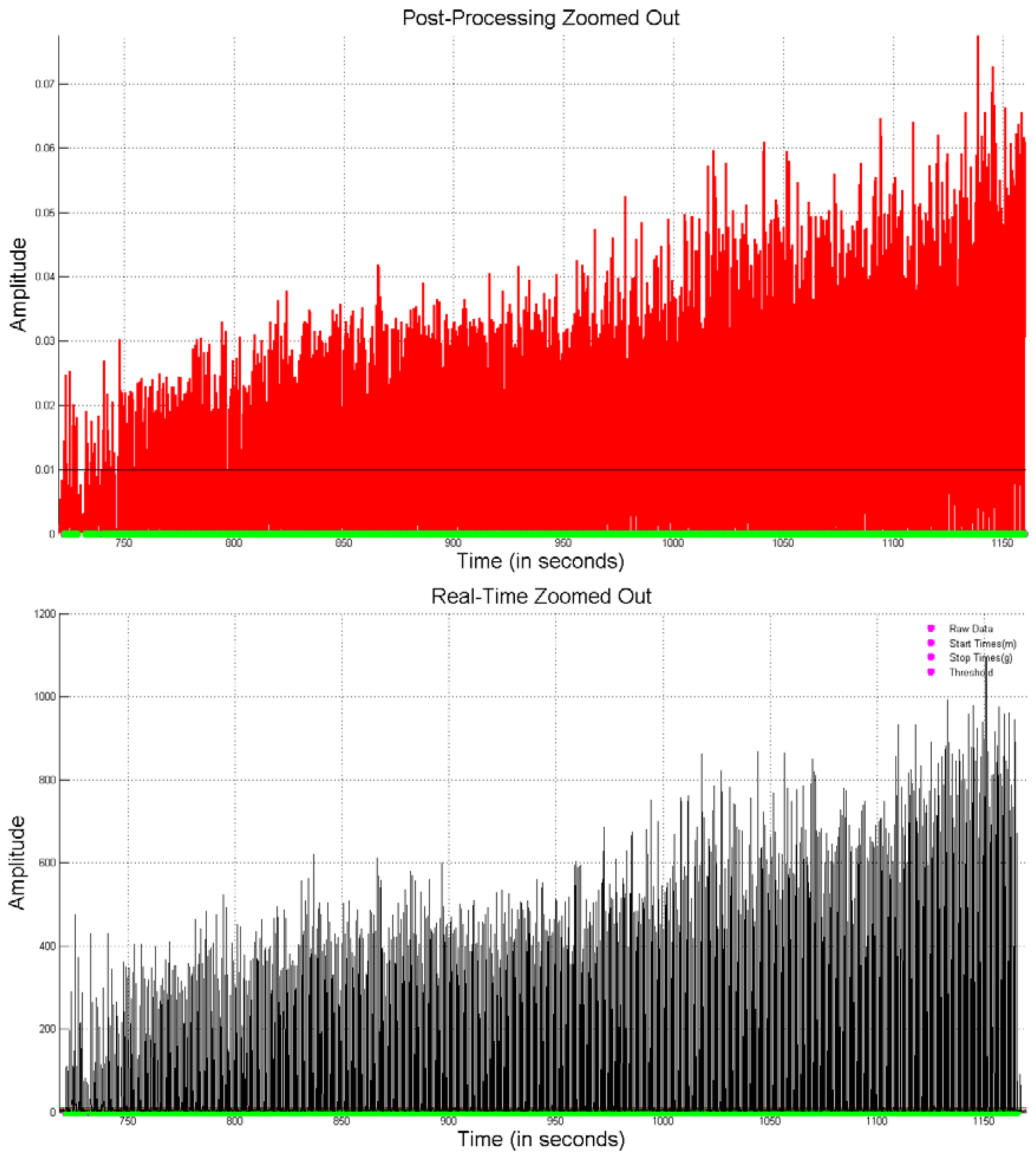


Figure 5.5 Full experiment with start/stop times identified post-experiment (top) and in real-time (bottom).

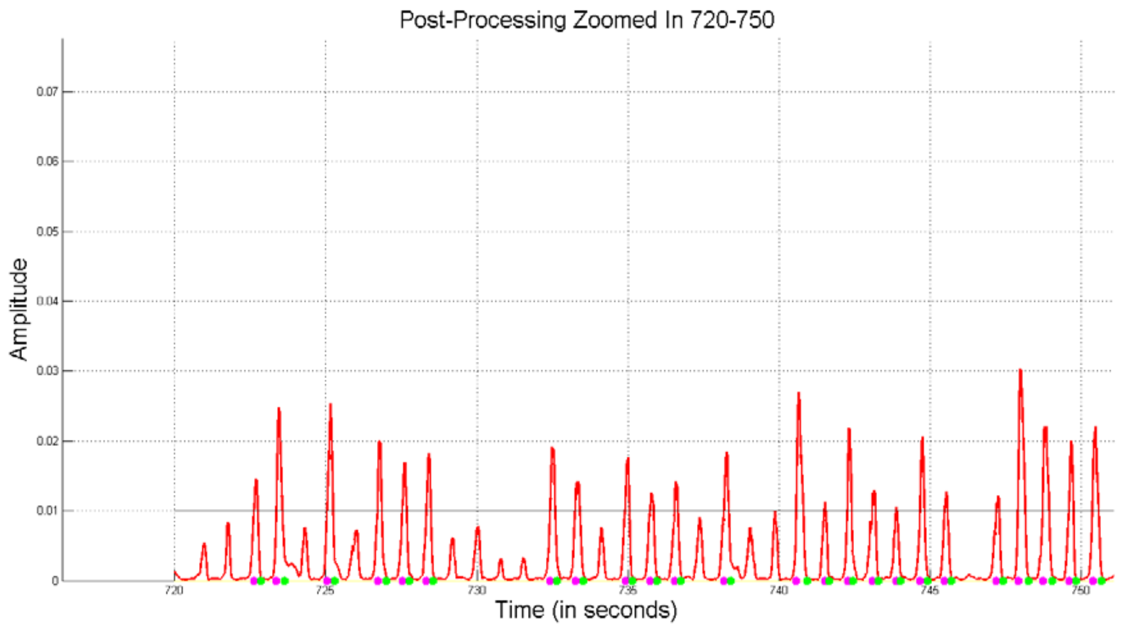


Figure 5.6 Post-processing of Test 1 data over the time interval of [720, 750] seconds.

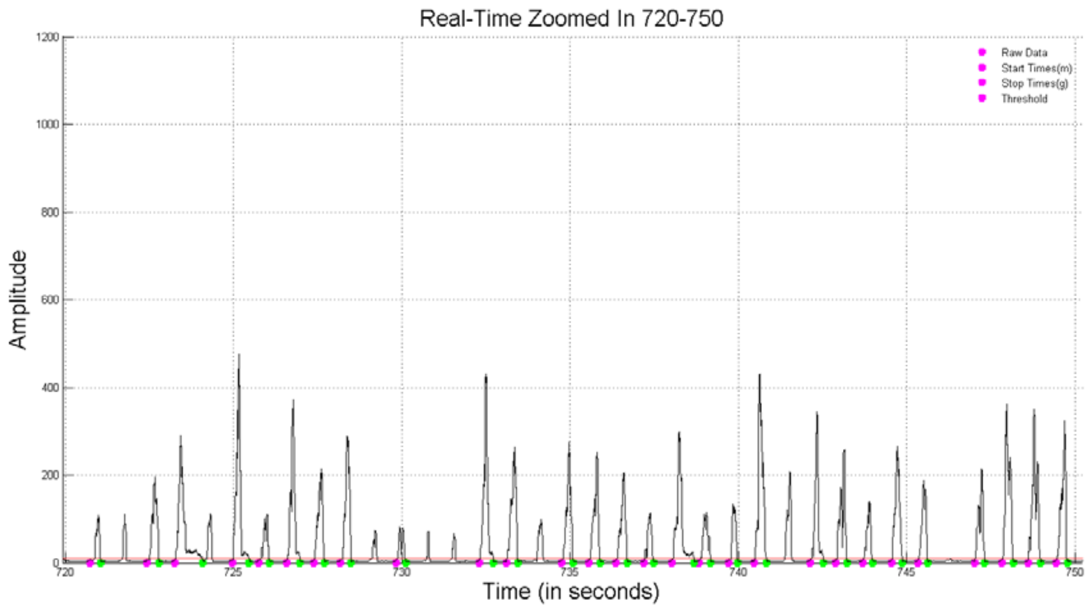


Figure 5.7 Real-time processing of Test 1 data over the time interval of [720, 750] seconds.

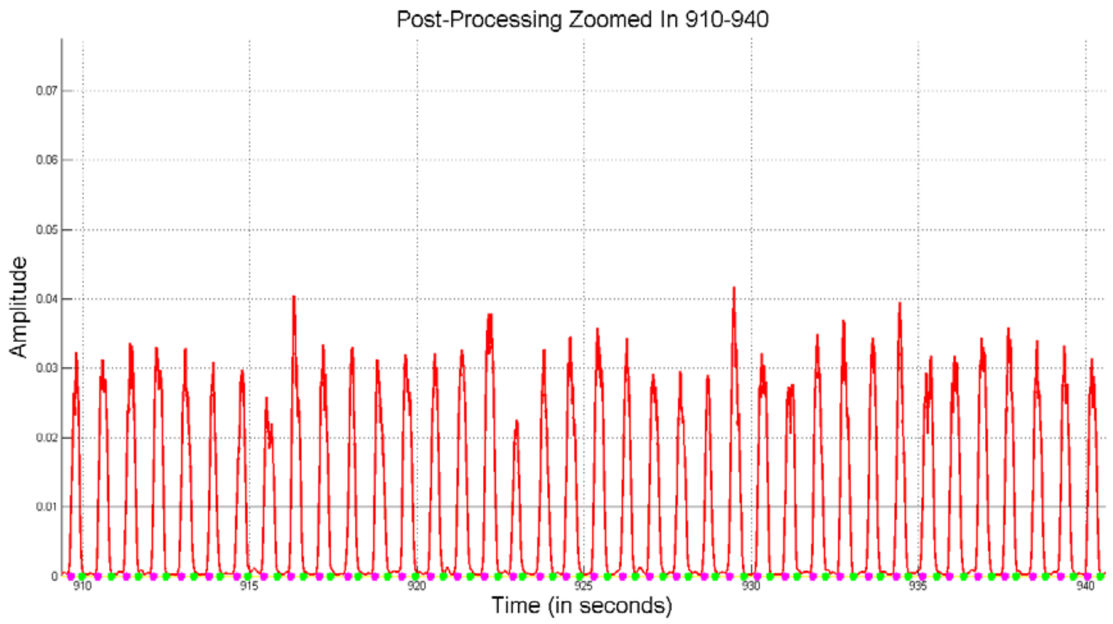


Figure 5.8 Post-processing of Test 1 data over the time interval of [910, 940] seconds.

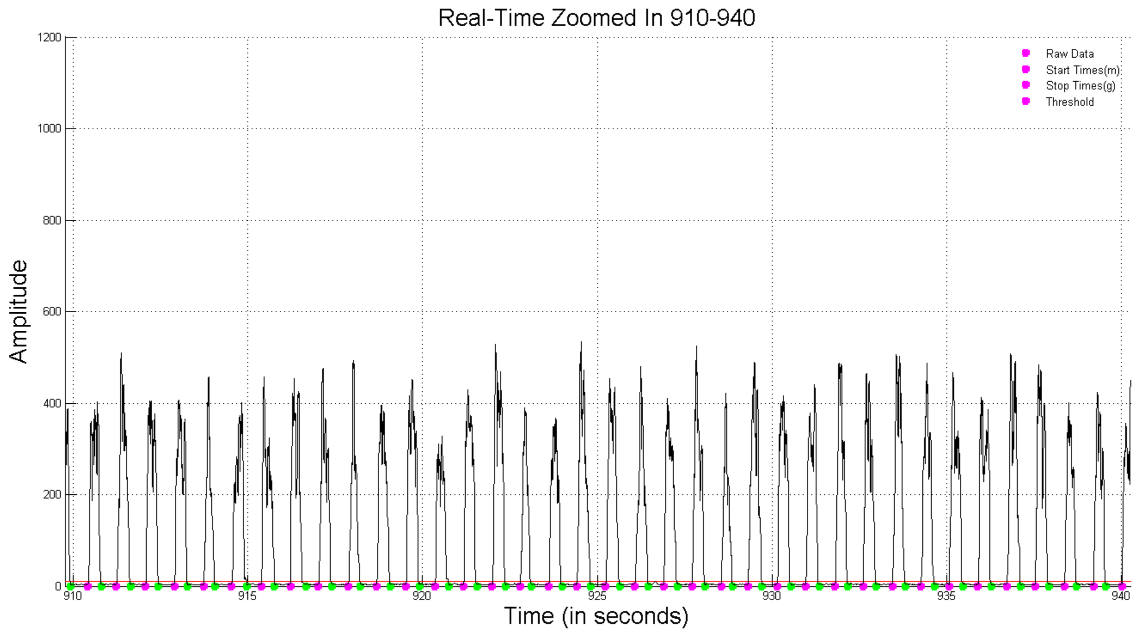


Figure 5.9 Real-time processing of Test 1 data over the time interval of [910, 940] seconds.

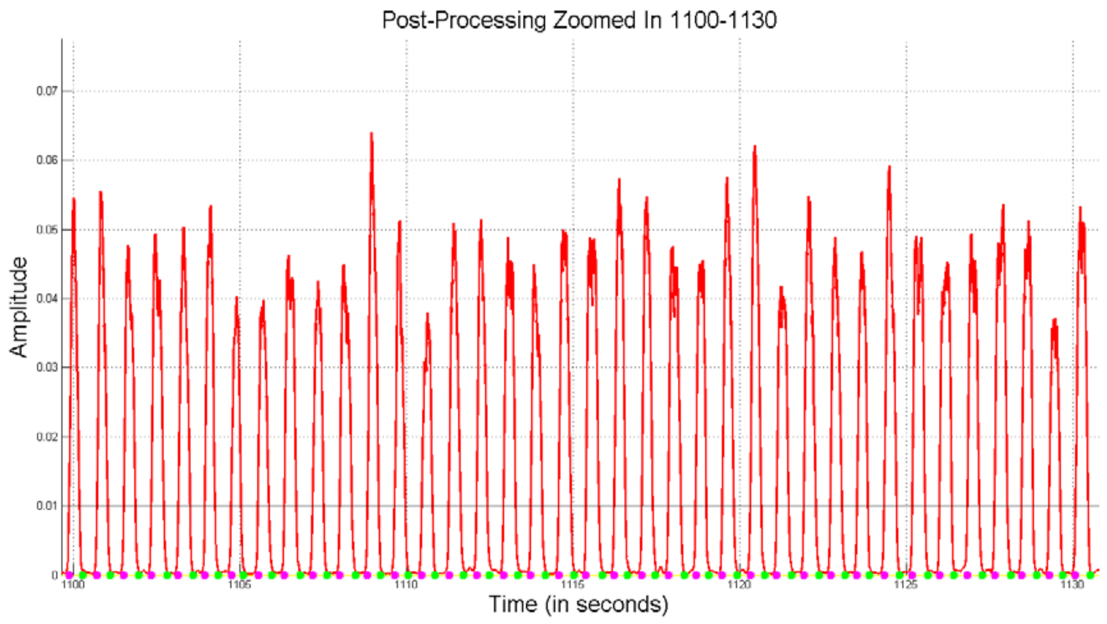


Figure 5.10 Post-processing of Test 1 data over the time interval of [1100, 1130] seconds.

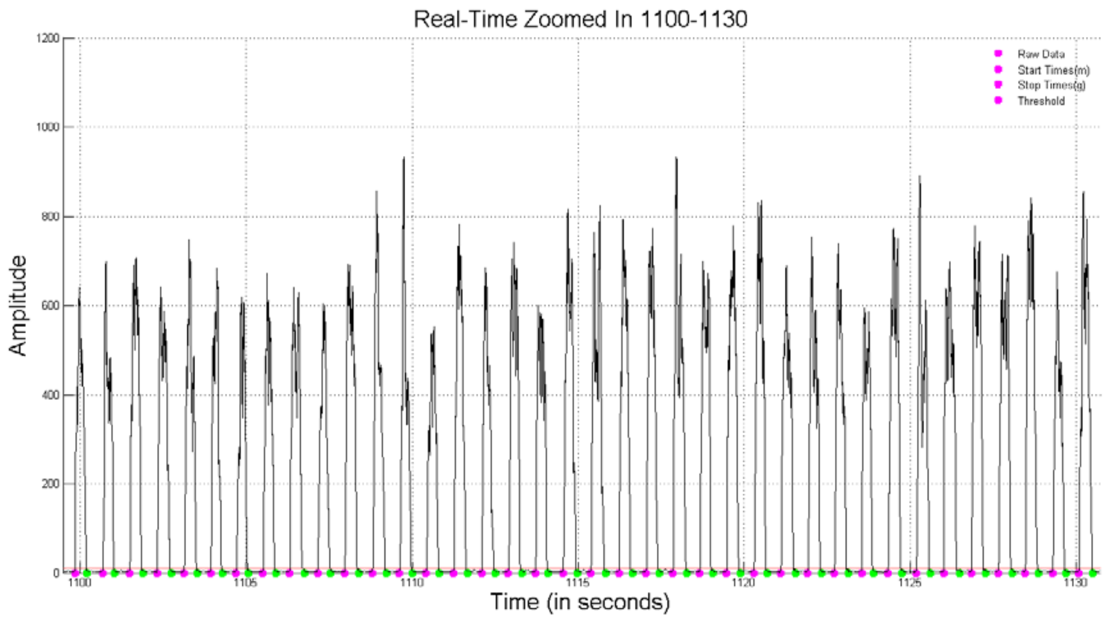


Figure 5.11 Real-time processing of Test 1 data over the time interval of [1100, 1130] seconds.

These time windows illustrate a scenario where the real-time processing system more accurately identified muscle bursts than the post-processing system. This improved burst identification is due to the fact that the real-time system uses a moving threshold, while the post-processing system employs a static threshold. This allows the real-time system to quickly adapt to the changing level of the EMG.

Figure 5.12 illustrates the Test 1 time-domain fatigue parameters calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the RMS value increases with fatigue. The integrated EMG and average rectified value also increase. The zero crossing rate, though, is fairly constant during Test 1. As a reminder, the LabVIEW screen contained all of the bursts before (i.e., test bursts), during, and after the experiment, while the burst movie contains just those during the experiment. Figure 5.13 illustrates the Test 1 frequency-domain fatigue parameters calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the MPF and MDF decrease near the end of Test 1, whereas the total spectral magnitude increases. The skewness and kurtosis values are fairly constant during Test 1.

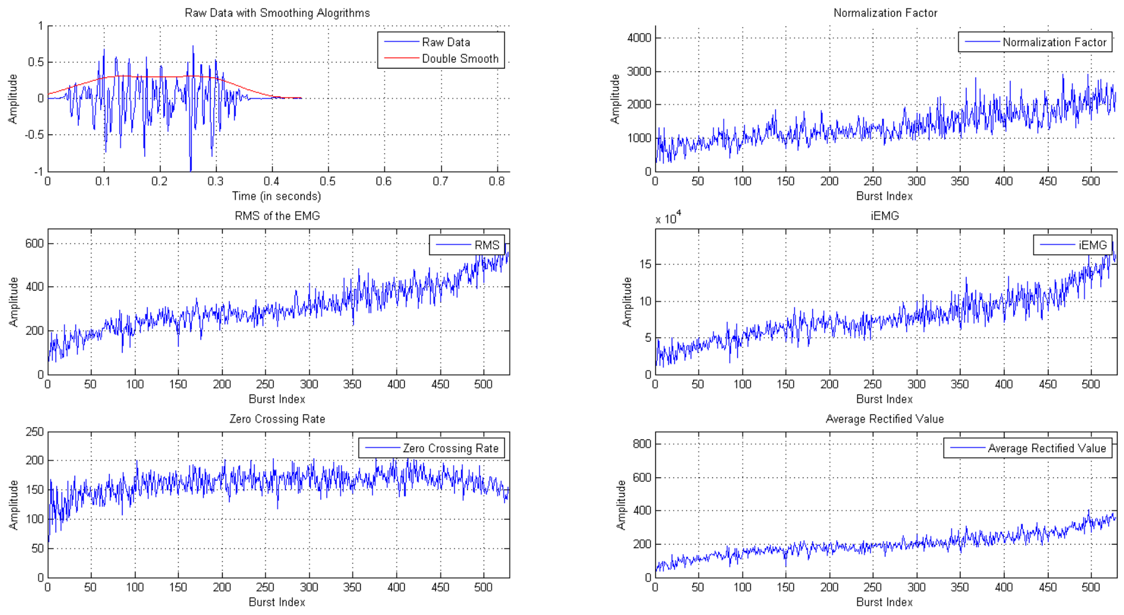


Figure 5.12 Time-domain movie frame for Test 1.

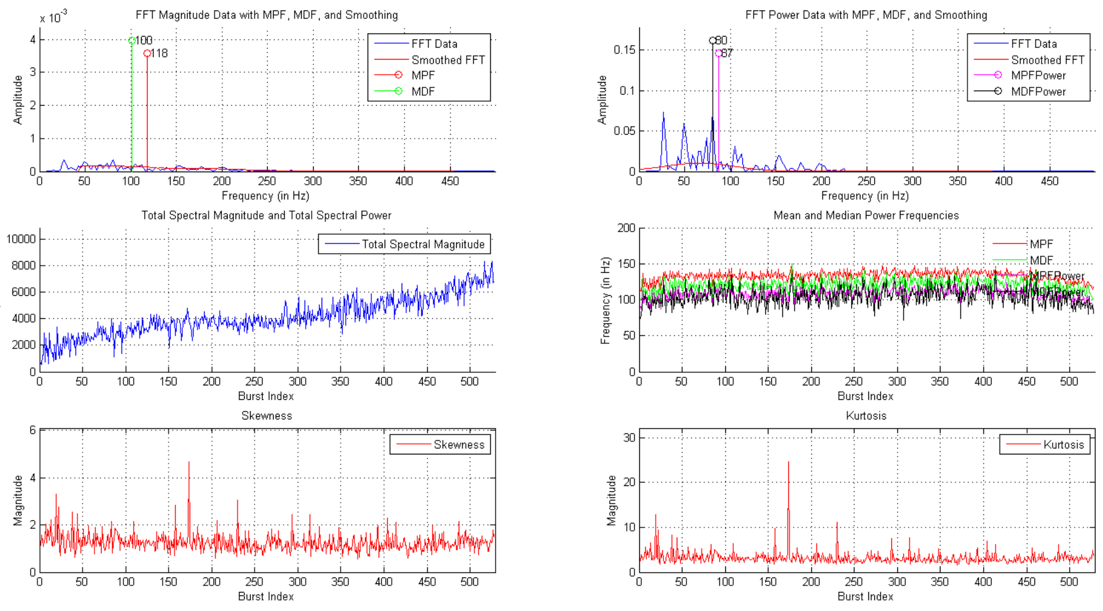
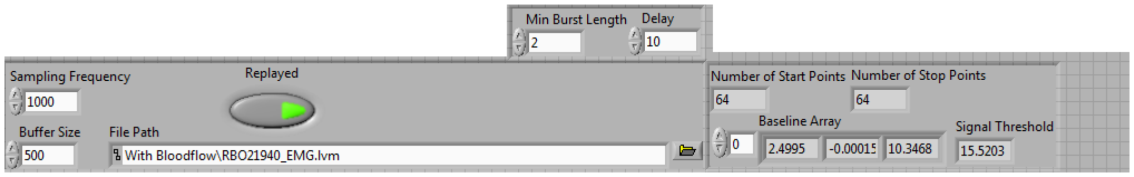


Figure 5.13 Frequency-domain movie frame for Test 1.

Test 2 – Constant Force Experiment A

The next test, Test 2, is the first of three tests that are constant force experiments. These tests were performed an on hand-forearm ergometer. The subjects were tested for maximum power output at a previous date before the fatigue experiment. The subjects contracted at a frequency of 20 contractions/min at a fixed power output of 85% of their maximum power output. This power output was maintained until the subject reached exhaustion.

As noted in Figure 5.14, the sampling frequency was set to 1 kHz, the data set was replayed through the system, and the buffer size was set to 500 data points. Figure 5.15 illustrates that (a) the RMS value increases as the subject fatigues, (b) the mean and median power frequencies for the broadband spectrum decrease, and (c) the mean and median power frequencies of the low band (5-30 Hz) stay constant throughout the experiment. As before, since the experiment was replayed using data files of an unsupported format, the force and displacement data shown are incorrect. Figure 5.16 illustrates that the mean and median power frequencies in the high band (95+ Hz) decrease during the experiment. The skewness and kurtosis parameters increase slightly as the subject approaches fatigue.



Min Burst Length 2
Delay 10
Sampling Frequency 1000
Replayed
Buffer Size 500
File Path With Bloodflow\RBO21940_EMG.lvm
Number of Start Points 64
Number of Stop Points 64
Baseline Array 0 2.4995 -0.00015 10.3468
Signal Threshold 15.5203

Figure 5.14 Input variables for Test 2.

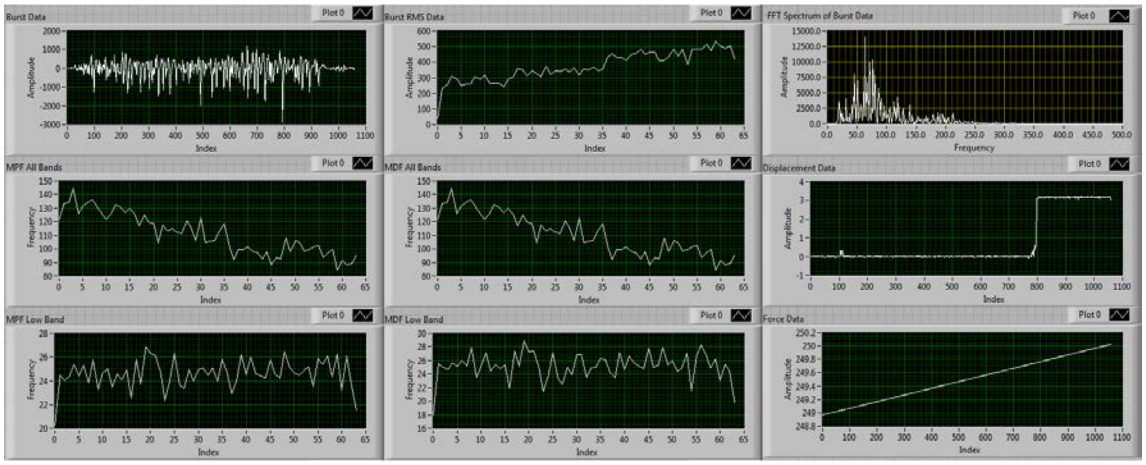


Figure 5.15 Top half of the output panels for Test 2.

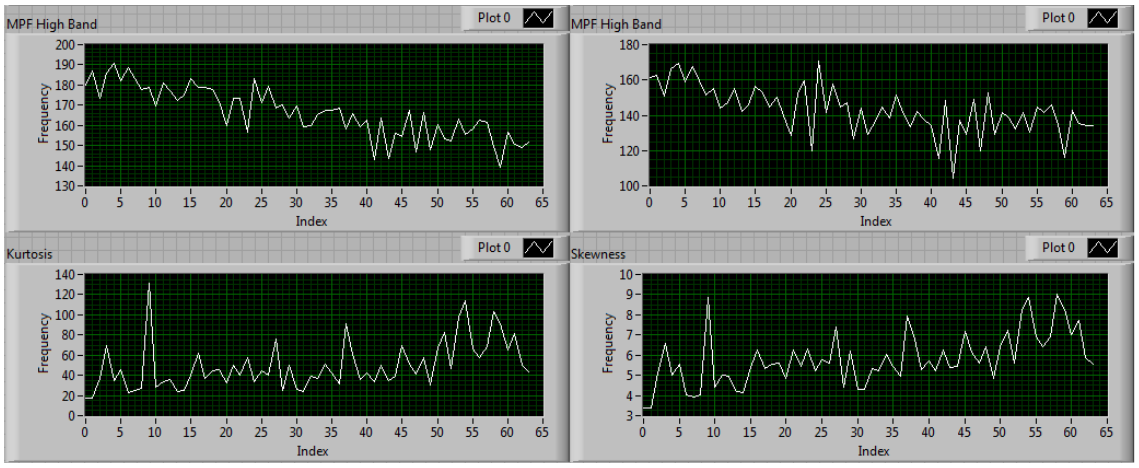
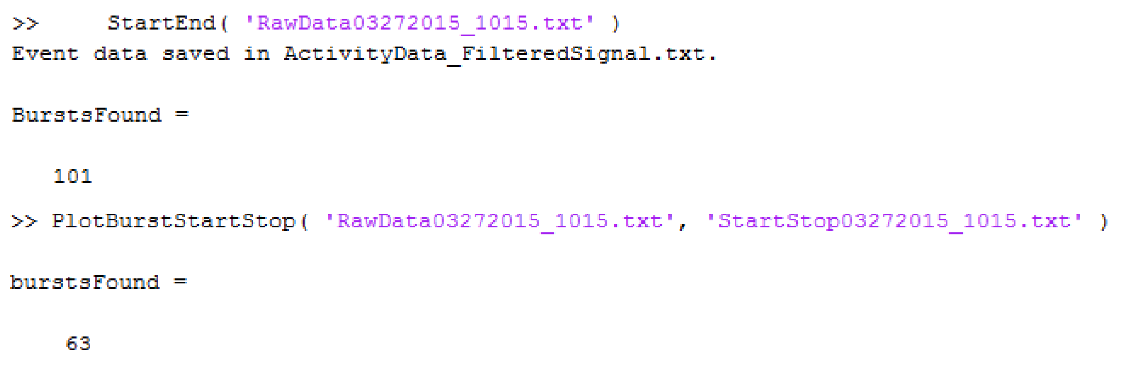


Figure 5.16 Bottom half of the output panels for Test 2.

As noted in Figure 5.17, the real-time system found fewer bursts during the experiment than the post-processing system, yielding a relative error of -37.6%, which is not encouraging.



```
>>     StartEnd( 'RawData03272015_1015.txt' )
Event data saved in ActivityData_FilteredSignal.txt.

BurstsFound =

   101

>> PlotBurstStartStop( 'RawData03272015_1015.txt', 'StartStop03272015_1015.txt' )

burstsFound =

    63
```

Figure 5.17 Number of bursts found using the post-processing system (top) versus the real-time system (bottom).

Figure 5.18 displays a comparison between the start and stop times for Test 2 over the whole experiment; the post-processing results are on the top and the real-time results are on the bottom. Generally, both systems found a sufficient amount of muscle bursts and there are no noticeable sections of experiment data where either system did not find muscle bursts. Figure 5.19 and Figure 5.20 help to gauge the burst-to-burst accuracy of the post-processing system versus the real-time system by comparing start/stop points for the individual bursts for the beginning of Test 2. The post-processing system identifies multiple bursts in some cases when only one burst exists. This is one cause of the discrepancy in the number of bursts found in Figure 5.19. Unlike the post-processing system, the real-time processing system does not artificially increase the number of muscle bursts detected. Figure 5.21 and Figure 5.22 address the accuracy of the post-processing system versus the real-time system for the middle of Test 2. During this time window, the post-processing system is still artificially increasing the number of muscle bursts by misidentifying the trailing edge of individual bursts as new bursts. During this time window, the real-time processing system was able to properly identify individual muscle bursts. Figure 5.23 and Figure 5.24 address the accuracy of the post-processing system versus the real-time system for the end of Test 2. During this time window, the post-processing system continues to over-count the number of muscle bursts, whereas the real-time processing system properly identifies the muscle bursts.

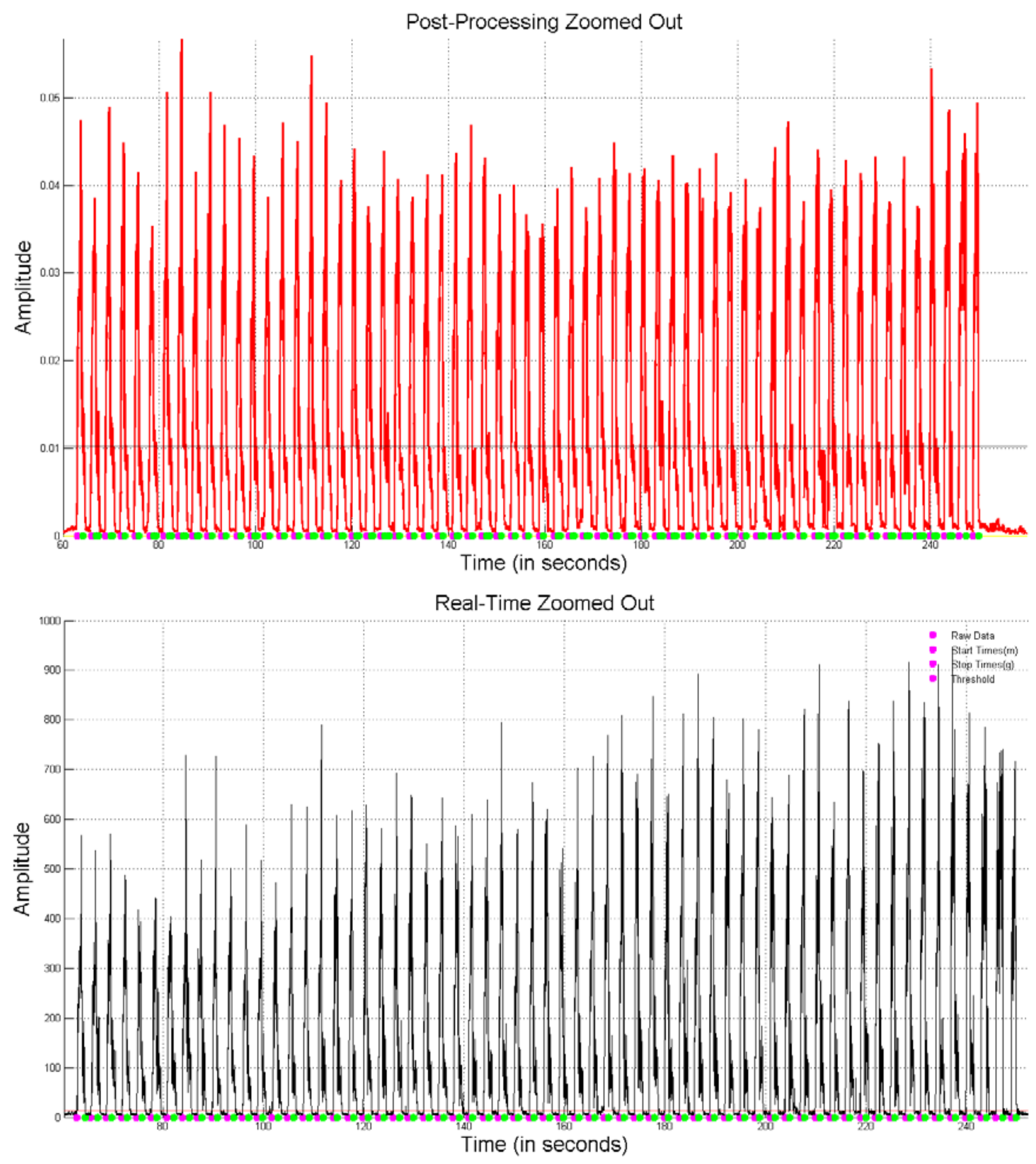


Figure 5.18 Full experiment with start/stop times identified post-experiment (top) and in real-time (bottom).

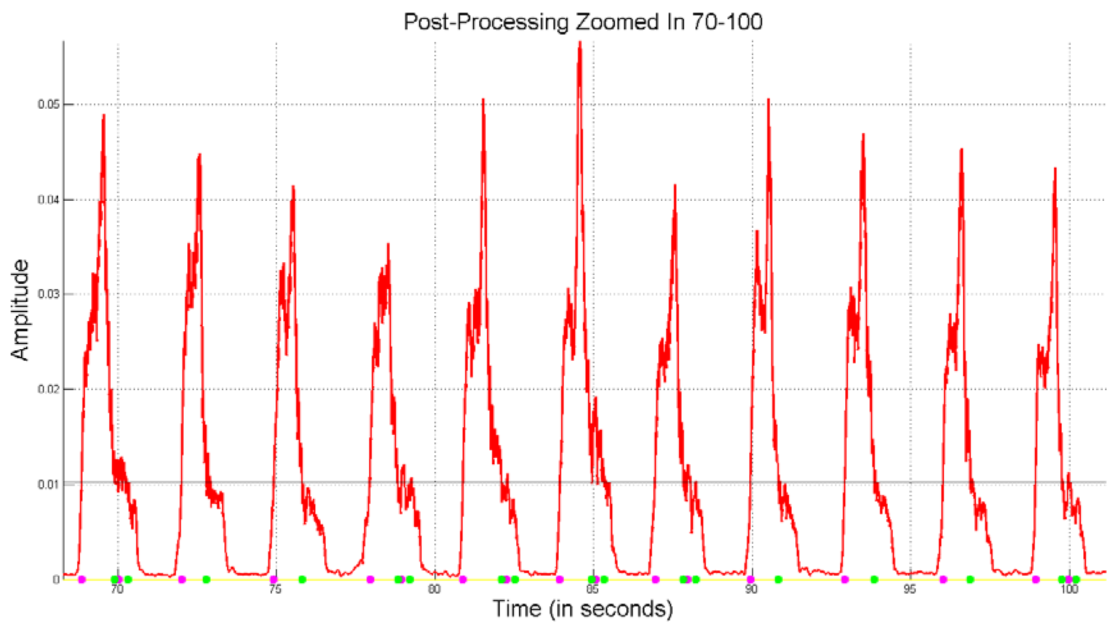


Figure 5.19 Post-processing of Test 2 data over the time interval of [70, 100] seconds.

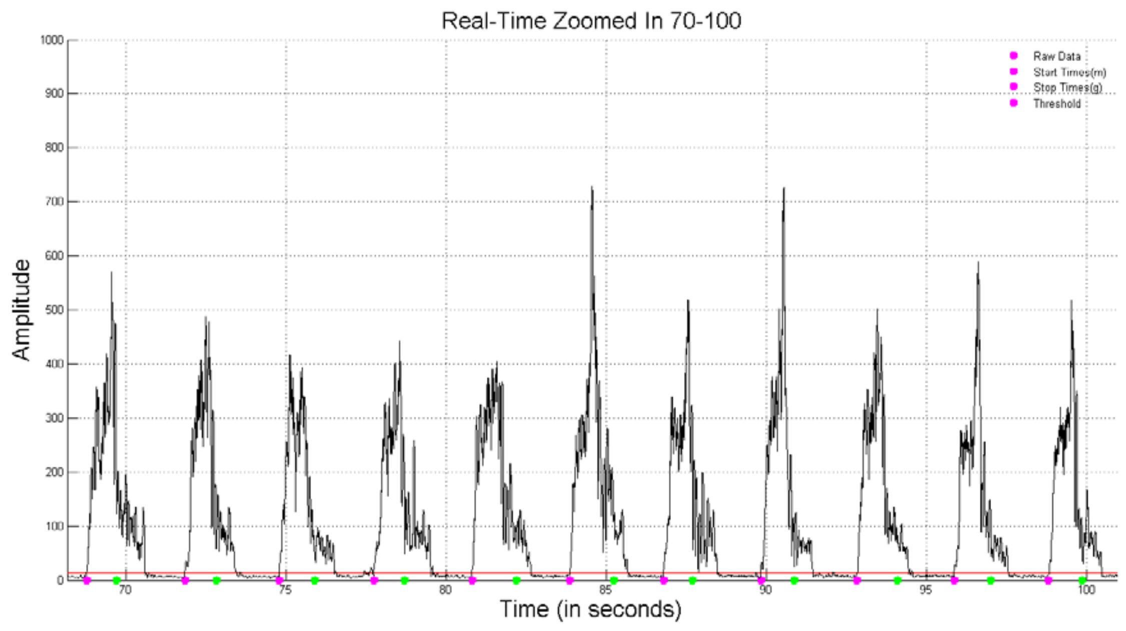


Figure 5.20 Real-time processing of Test 2 data over the time interval of [70, 100] seconds.

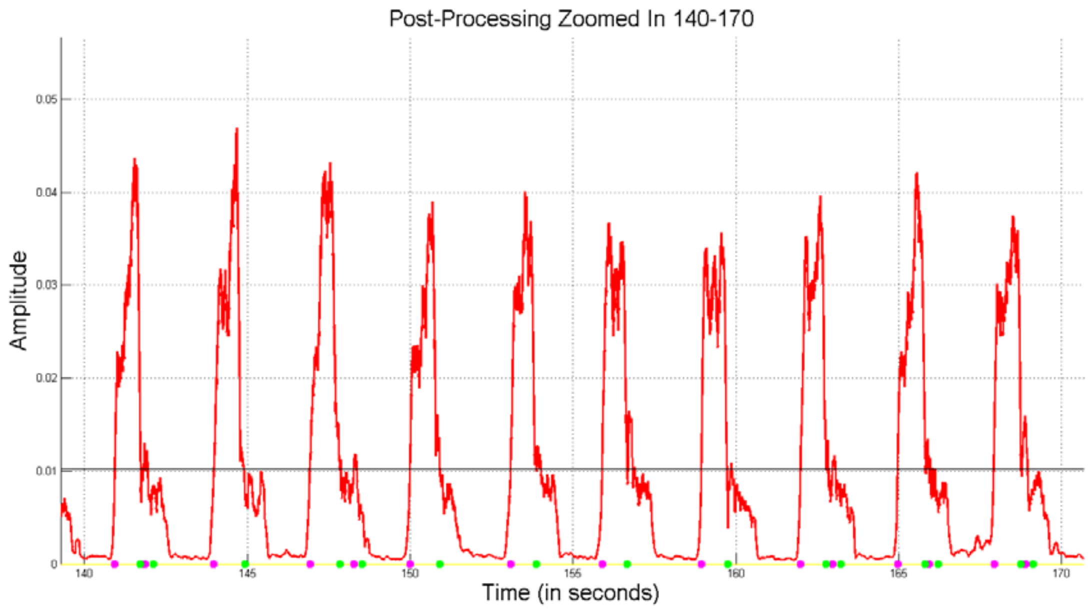


Figure 5.21 Post-processing of Test 2 data over the time interval of [140, 170] seconds.

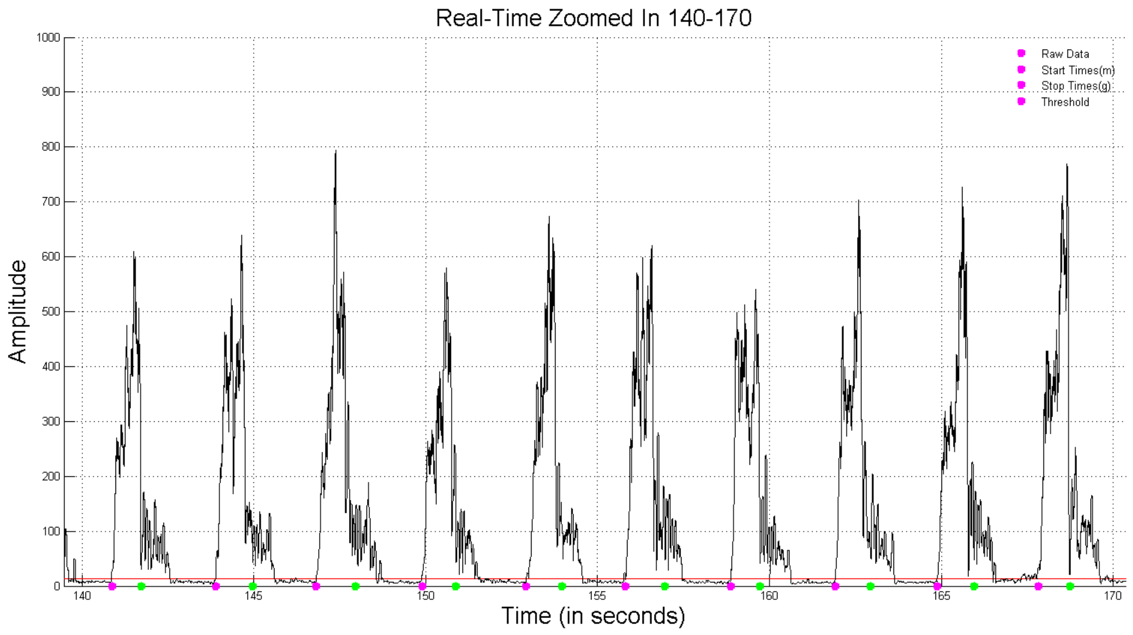


Figure 5.22 Real-time processing of Test 2 data over the time interval of [140, 170] seconds.

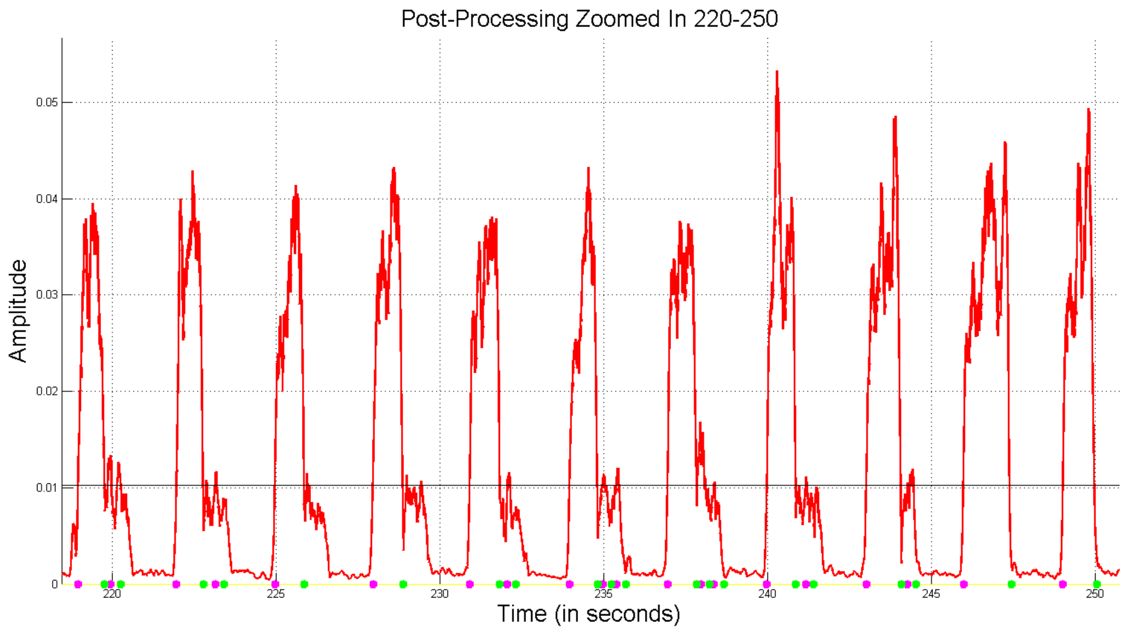


Figure 5.23 Post-processing of Test 2 data over the time interval of [220, 250] seconds.

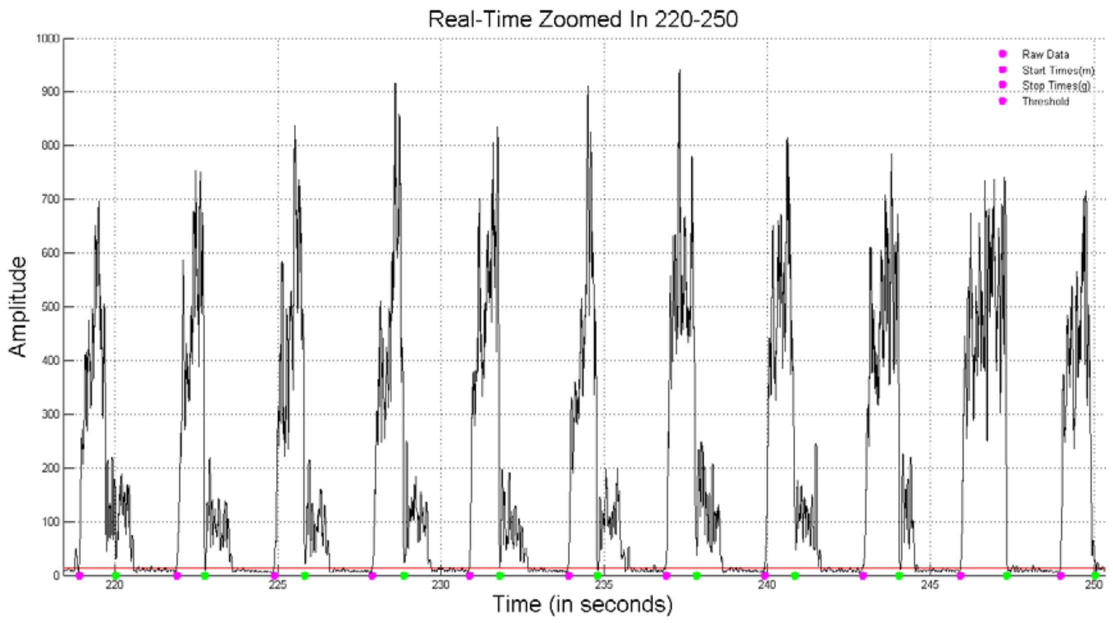


Figure 5.24 Real-time processing of Test 2 data over the time interval of [220, 250] seconds.

These time windows illustrate a scenario where the real-time processing system more accurately identified muscle bursts than the post-processing system. In this scenario, after the user releases their hold on the ergometer handles, the musculature continues to generate a trailing EMG signature. This improved burst identification is due to the fact that the real-time system uses a moving threshold, while the post-processing system employs a static threshold. This allows the real-time system to quickly adapt to the changing level of the EMG.

Figure 5.25 illustrates the Test 2 time-domain fatigue parameters calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the RMS value increases with fatigue. The integrated EMG and average rectified value also increase, whereas the zero crossing rate noticeably decreases. As a reminder, the LabVIEW screen contained all of the bursts before, during, and after the experiment, while the burst movie contains just those during the experiment. Figure 5.26 illustrates the Test 2 frequency-domain fatigue parameters calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the MPF and MDF (both magnitude and power) decreases near the end of Test 2. The total spectral magnitude, skewness, and kurtosis increase as the subject approaches fatigue. It is important to remember that the skewness and kurtosis are calculated using probability density function values acquired from a Gaussian fit to the power FFT coefficients, not calculated directly from the raw power FFT coefficients, which is how the real-time system calculates skewness and kurtosis.

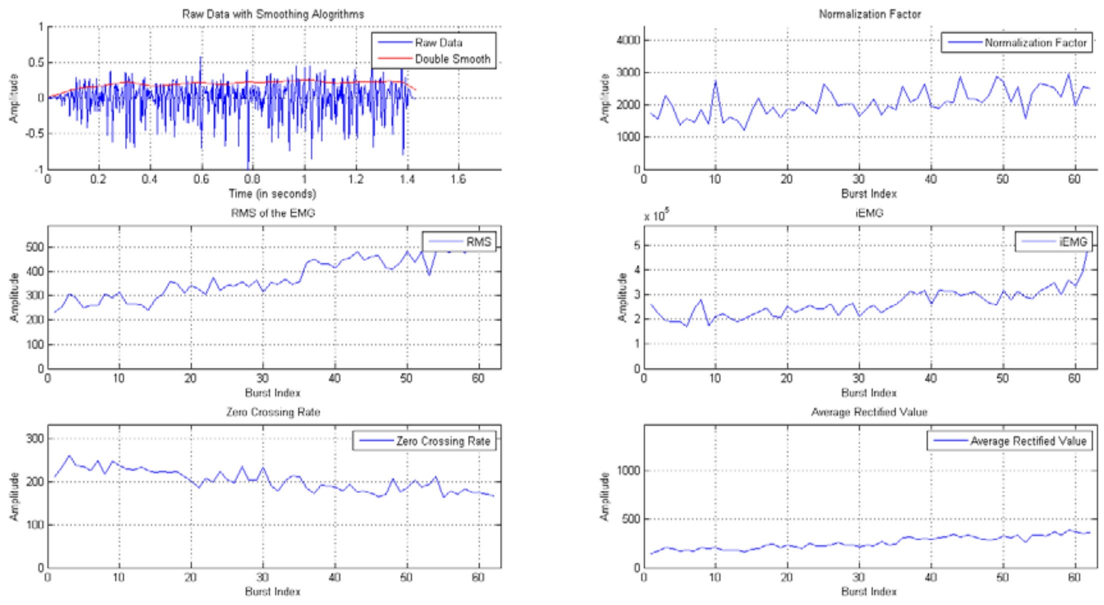


Figure 5.25 Time-domain movie frame for Test 2.

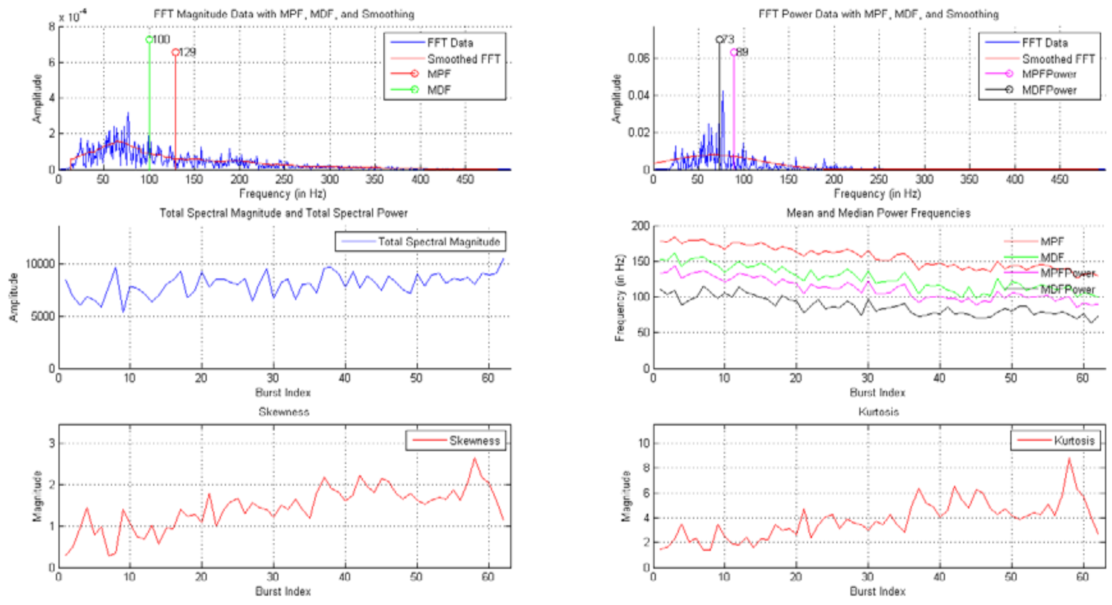


Figure 5.26 Frequency-domain movie frame for Test 2.

Test 3 – Constant Force Experiment B

As noted in Figure 5.27, the sampling frequency was set at 1 kHz, the data set was replayed through the system, and the buffer size was set to 500 data points. Figure 5.28 illustrates that (a) the RMS value increases as the subject fatigues, (b) the mean and median power frequencies for the broadband spectrum decrease, and (c) the mean and median power frequencies for the low band (5-30 Hz) stay constant throughout the experiment. As before, since the experiment was replayed using data files of an unsupported format, the force and displacement data shown are incorrect. Figure 5.29 illustrates that the mean and median power frequencies in the high band (95+ Hz) decrease during the experiment. The skewness and kurtosis parameters hold constant as the subject approaches fatigue.

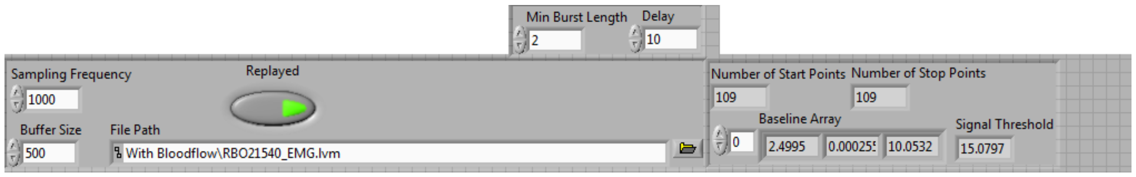

Min Burst Length: 2
Delay: 10
Sampling Frequency: 1000
Replayed
Buffer Size: 500
File Path: With Bloodflow\RBO21540_EMG.lvm
Number of Start Points: 109
Number of Stop Points: 109
Baseline Array: 0 | 2.4995 | 0.00025! | 10.0532
Signal Threshold: 15.0797

Figure 5.27 Input variables for Test 3.

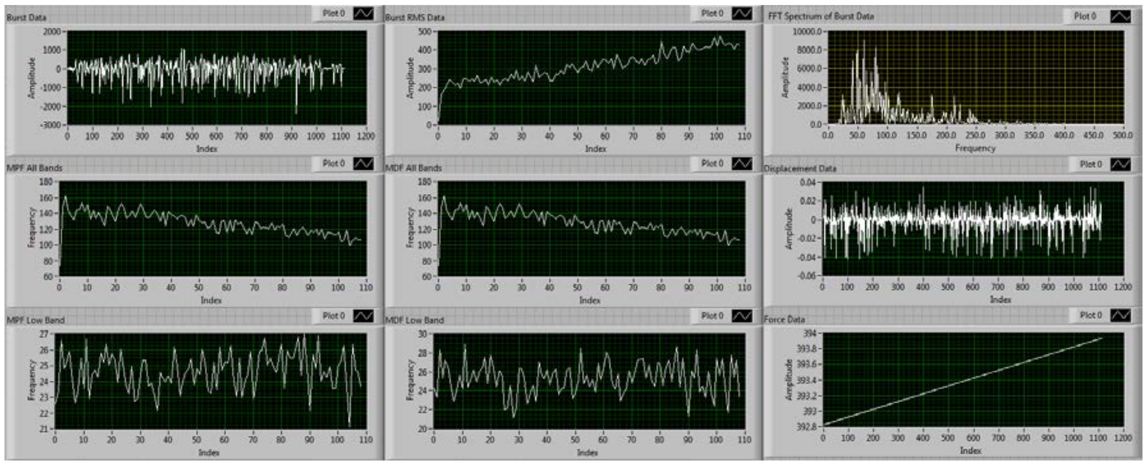


Figure 5.28 Top half of the output panels for Test 3.

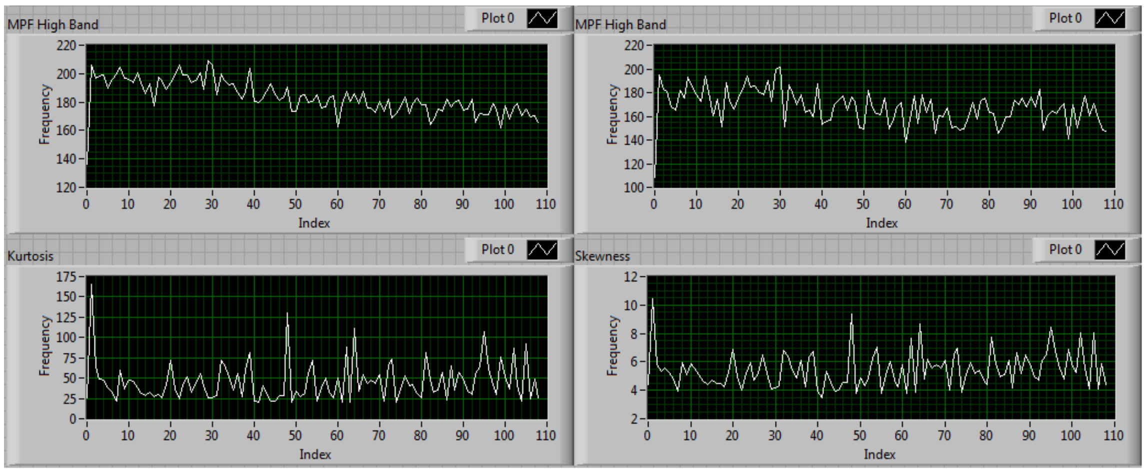
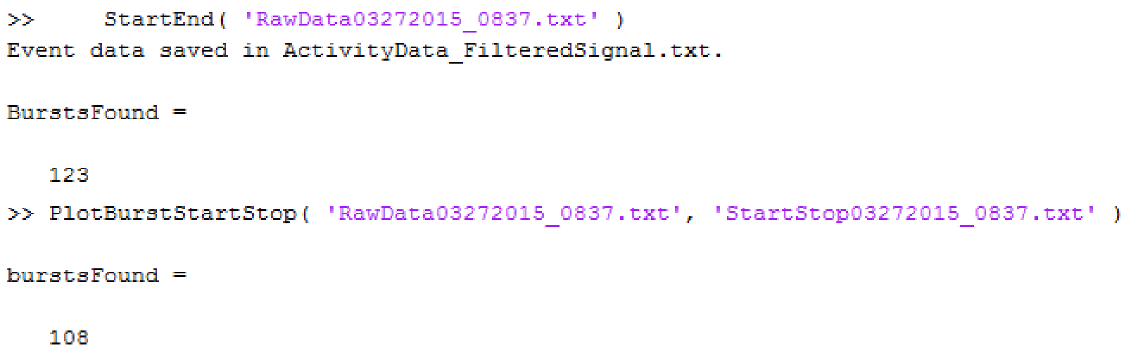


Figure 5.29 Bottom half of the output panels for Test 3.

As noted in Figure 5.30, the real-time system found fewer bursts during the experiment than the post-processing system, yielding a relative error of -12.2%, which is not encouraging.



```
>>     StartEnd( 'RawData03272015_0837.txt' )
Event data saved in ActivityData_FilteredSignal.txt.

BurstsFound =

   123
>> PlotBurstStartStop( 'RawData03272015_0837.txt', 'StartStop03272015_0837.txt' )

burstsFound =

   108
```

Figure 5.30 Number of bursts found using the post-processing system (top) versus the real-time system (bottom).

Figure 5.31 displays a comparison between the start and stop times for Test 3 over the whole experiment; the post-processing results are on the top and the real-time results are on the bottom. Generally, both systems found a sufficient amount of muscle bursts and there are no noticeable sections of experiment data where either system did not find muscle bursts. Figure 5.32 and Figure 5.33 help to gauge the burst-to-burst accuracy of the post-processing system versus the real-time system by comparing start/stop points for the individual bursts for the beginning of Test 3. The post-processing system double-counts a burst only once in this window. This behavior may be one cause of the discrepancy in the number of bursts found in Figure 5.32. Unlike the post-processing system, the real-time processing system does not artificially increase the number of muscle bursts detected. Figure 5.34 and Figure 5.35 address the accuracy of the post-processing system versus the real-time system for the middle of Test 3. During this time window, the post-processing system is still artificially increasing the number of muscle bursts. The real-time processing system properly identified these muscle bursts. Figure 5.36 and Figure 5.37 address the accuracy of the post-processing system for the end of Test 3. During this time window, the post-processing system continues to over-count the number of muscle bursts, whereas the real-time processing system properly identifies the muscle bursts.

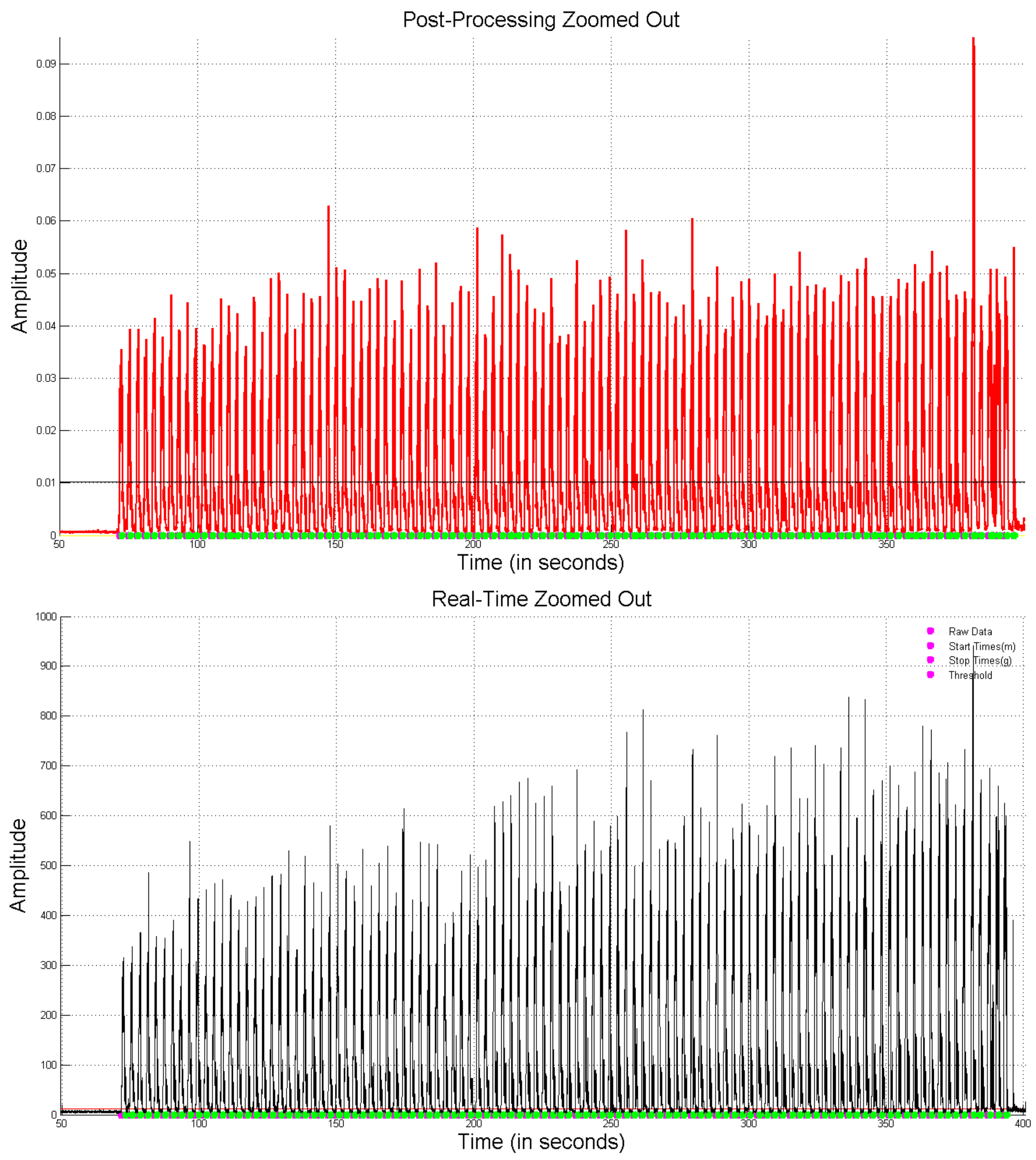


Figure 5.31 Full experiment with start/stop times identified post-experiment (top) and in real time (bottom).

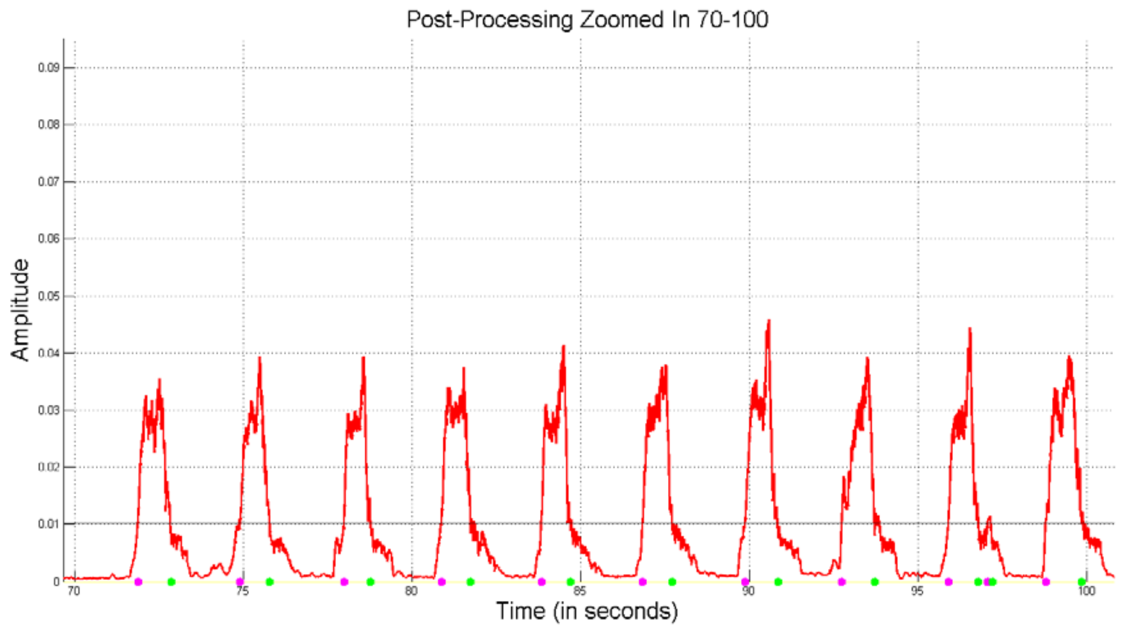


Figure 5.32 Post-processing of Test 3 data over the time interval of [70, 100] seconds.

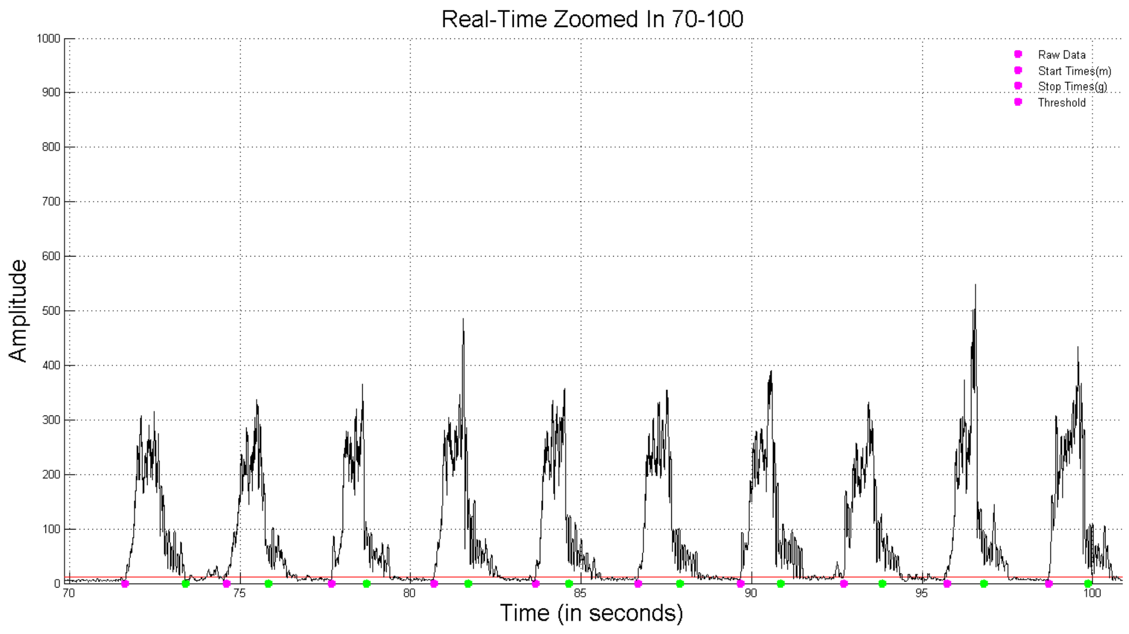


Figure 5.33 Real-time processing of Test 3 data over the time interval of [70, 100] seconds.

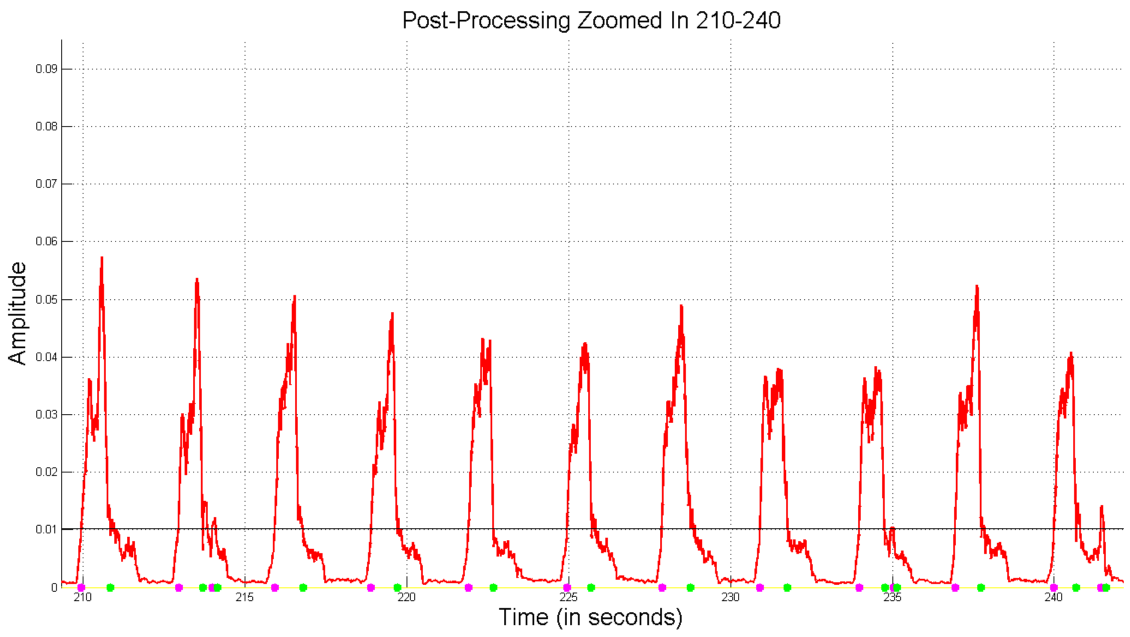


Figure 5.34 Post-processing of Test 3 data over the time interval of [210, 240] seconds.

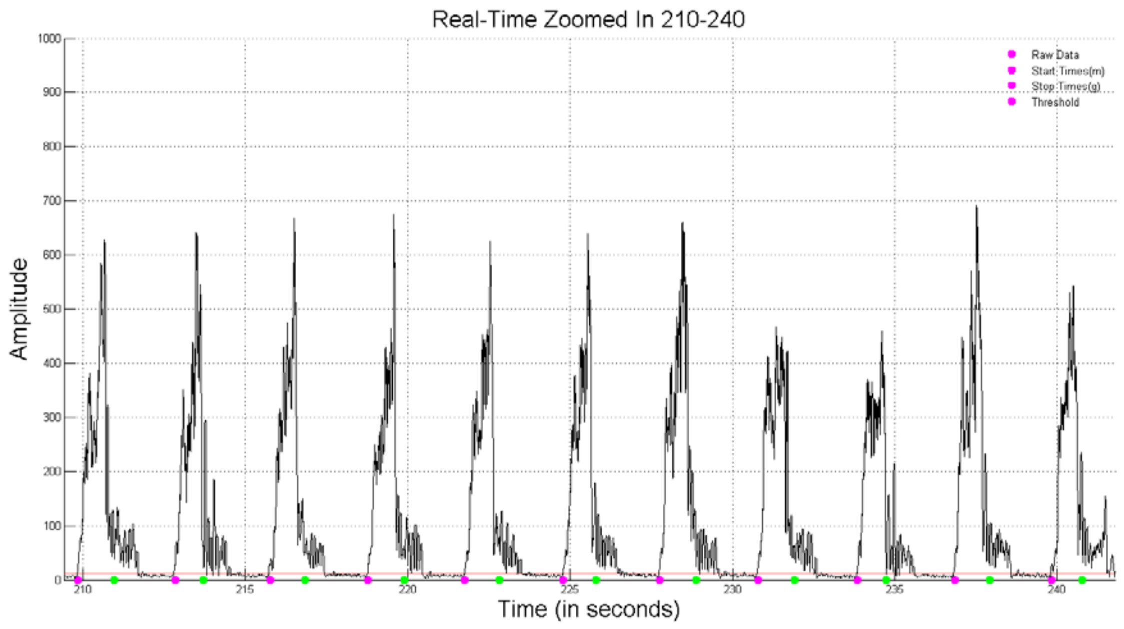


Figure 5.35 Real-time processing of Test 3 data over the time interval of [210, 240] seconds.

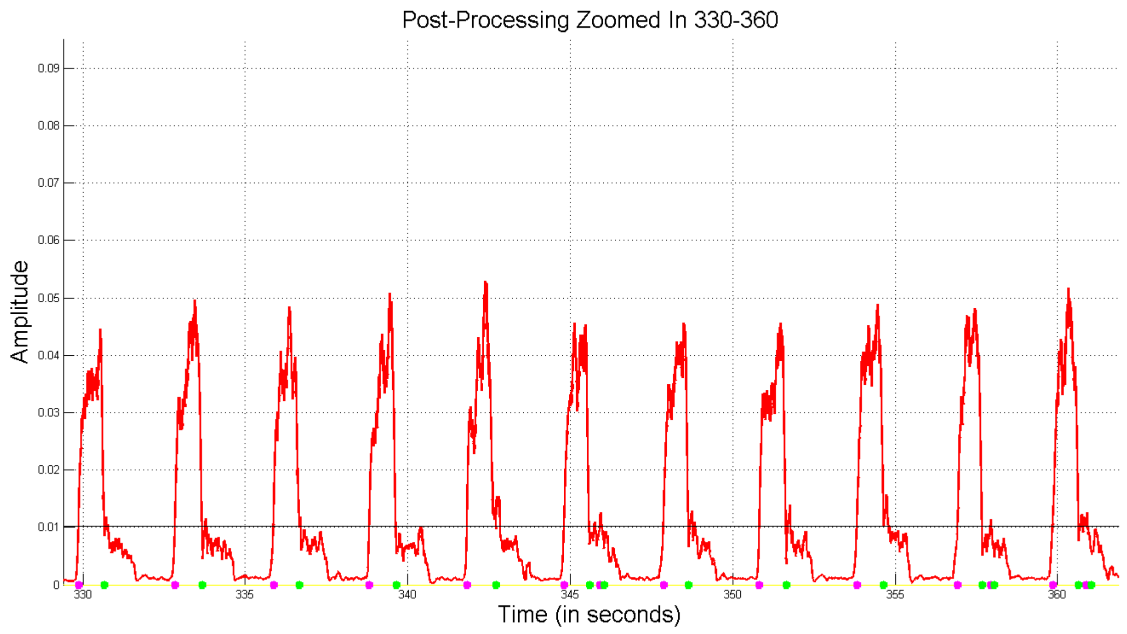


Figure 5.36 Post-processing of Test 3 data over the time interval of [330, 360] seconds.

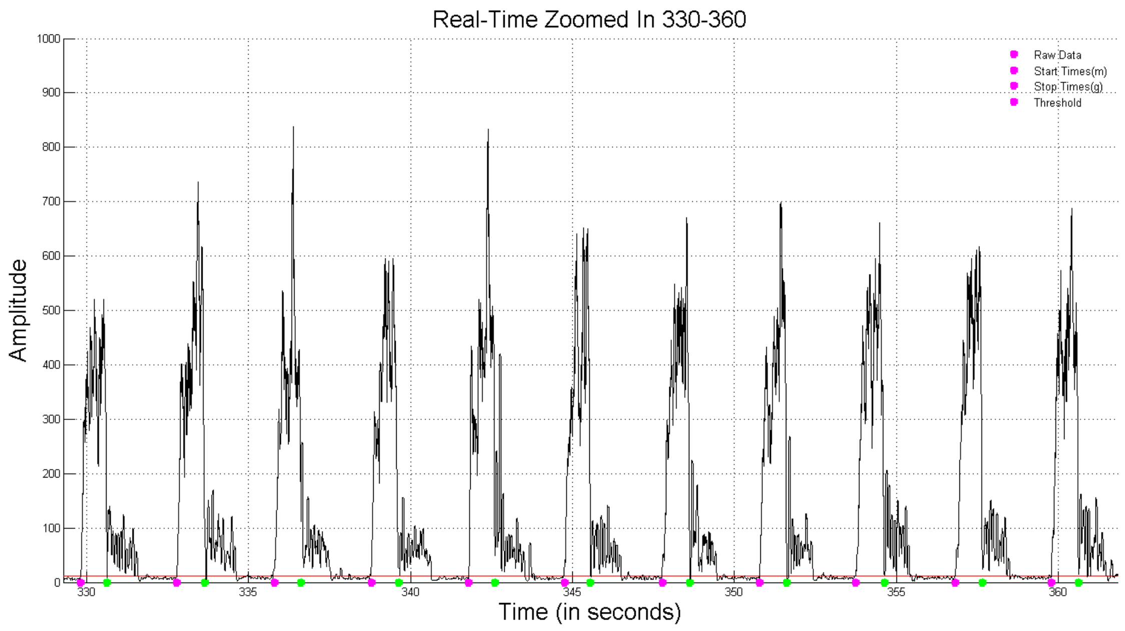


Figure 5.37 Real-time processing of Test 3 data over the time interval of [330, 360] seconds.

These time windows illustrate a scenario where the real-time processing system more accurately identified muscle bursts than the post-processing system. As with Test 2, this improved burst identification is due to the fact that the real-time system uses a moving threshold, while the post-processing system employs a static threshold. This allows the real-time system to quickly adapt to the changing level of the EMG.

Figure 5.38 illustrates the Test 3 time-domain fatigue parameters that were calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the RMS value increases with fatigue. The integrated EMG and average rectified value also increase, whereas the zero crossing rate noticeably decreases. As a reminder, the LabVIEW screen contained all of the bursts before, during, and after the experiment, while the burst movie contains just those during the experiment. Figure 5.39 illustrates the Test 3 frequency-domain fatigue parameters calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the MPF and MDF (both magnitude and power) decrease near the end of Test 3. The total spectral magnitude noticeably increases, while the skewness and kurtosis have a very small increase.

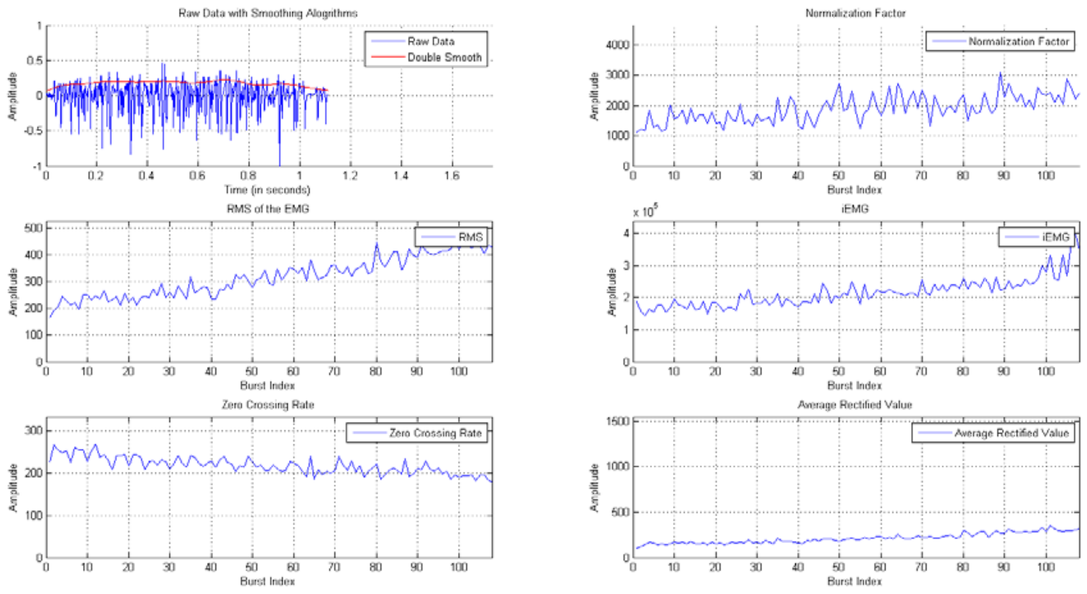


Figure 5.38 Time-domain movie frame for Test 3.

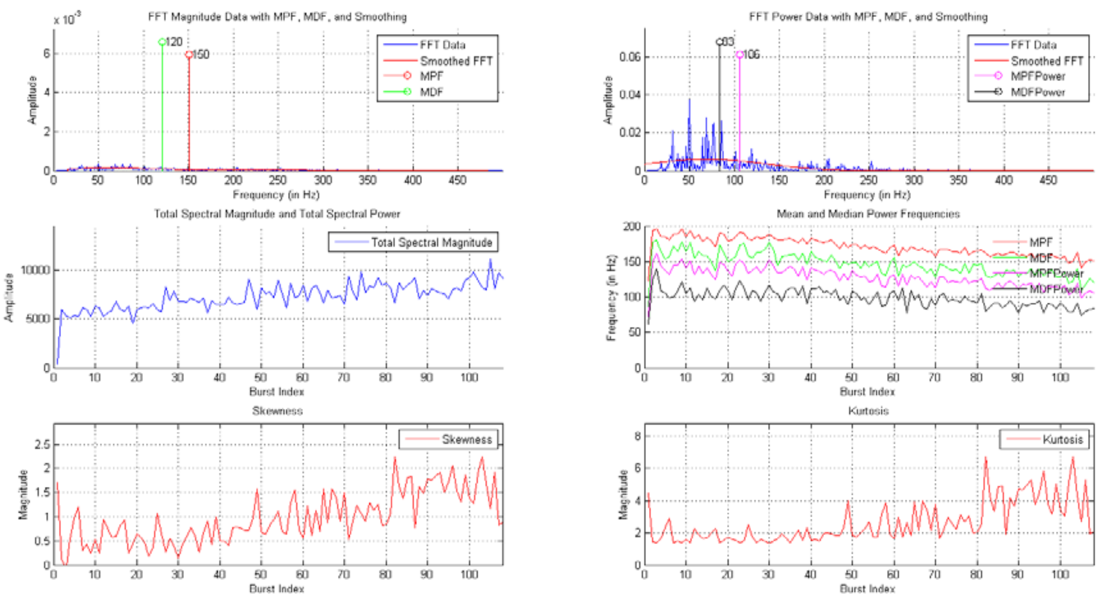
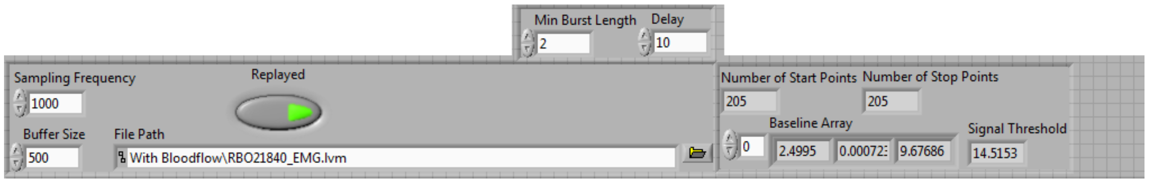


Figure 5.39 Frequency-domain movie frame for Test 3.

Test 4 – Constant Force Experiment C

As noted in Figure 5.40, the sampling frequency was set at 1 kHz, the data set was replayed through the system, and the buffer size was set to 500 data points. Figure 5.41 illustrates that (a) the RMS increases as the subject fatigues, (b) the mean and median power frequencies for the broadband spectrum decrease, and (c) the mean and median power frequencies for the low bands (5-30 Hz) stay constant throughout the experiment. About halfway through the experiment, the RMS, MDF, and MPF all remain constant. As before, since the experiment was replayed using data files of an unsupported format, the force and displacement data shown are incorrect. Figure 5.42 illustrates that the mean and median power frequencies in the high band (95+ Hz) decrease during the experiment. The skewness and kurtosis parameters hold constant as the subject approaches fatigue.



Min Burst Length 2
Delay 10
Sampling Frequency 1000
Replayed
Buffer Size 500
File Path With Bloodflow\RBO21840_EMG.lvm
Number of Start Points 205
Number of Stop Points 205
Baseline Array 0 2.4995 0.00072: 9.67686
Signal Threshold 14.5153

Figure 5.40 Input variables for Test 4.

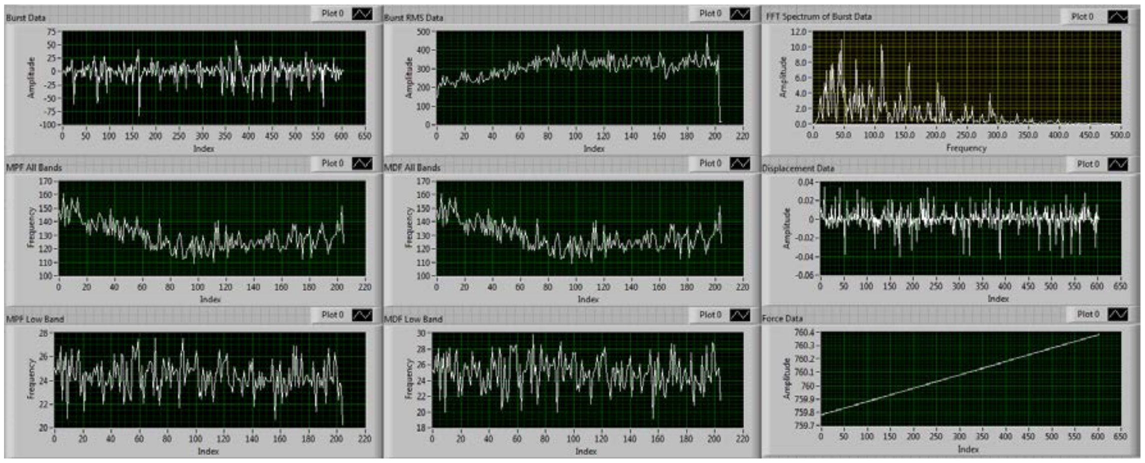


Figure 5.41 Top half of the output panels for Test 4.

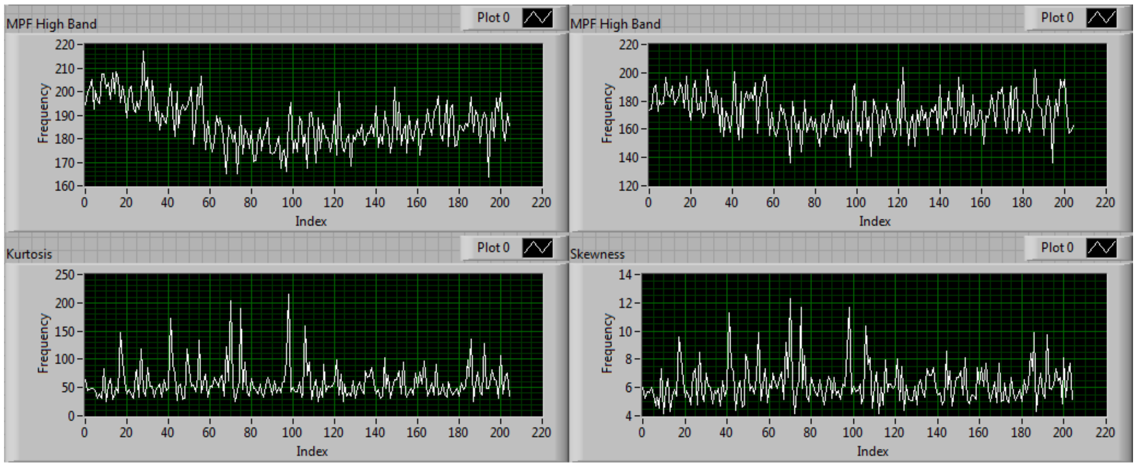
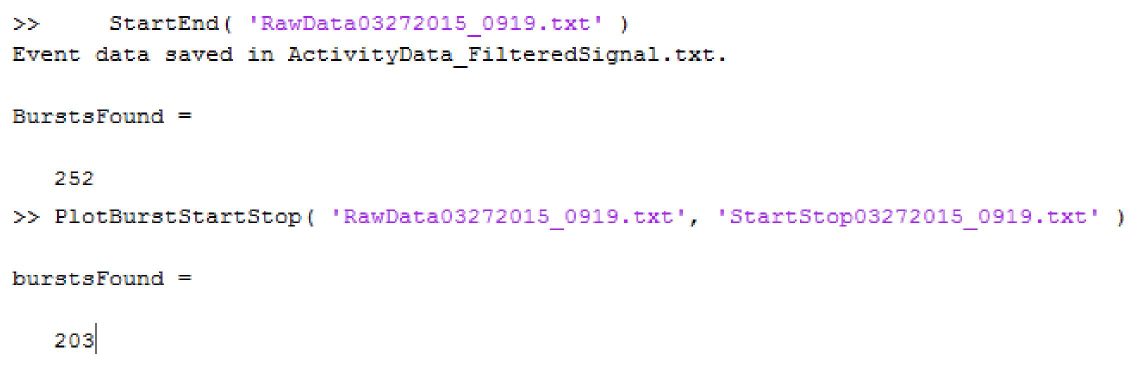


Figure 5.42 Bottom half of the output panels for Test 4.

As noted in Figure 5.43, we see that the real-time system found less bursts during the experiment than the post-processing system, yielding a relative error of -19.4%, which is not encouraging.


```
>>     StartEnd( 'RawData03272015_0919.txt' )
Event data saved in ActivityData_FilteredSignal.txt.

BurstsFound =

   252
>> PlotBurstStartStop( 'RawData03272015_0919.txt', 'StartStop03272015_0919.txt' )

burstsFound =

   203
```

Figure 5.43 Number of bursts found using the post-processing system (top) versus the real-time system (bottom).

Figure 5.44 displays a comparison between the start and stop times for Test 4 over the whole experiment; the post-processing results are on the top and the real-time results are on the bottom. Generally, both systems found a sufficient amount of muscle bursts and there are no noticeable sections of experiment data where either system did not find muscle bursts. Figure 5.45 and Figure 5.46 help to gauge the burst-to-burst accuracy of the post-processing system versus the real-time system by comparing start/stop points for the individual bursts for the beginning of Test 4. The post-processing system double counts bursts twice in this window. This behavior may be one cause of the discrepancy in the number of bursts found in Figure 5.45. Unlike the post-processing system, the real-time processing system does not artificially increase the number of muscle bursts detected. Figure 5.47 and Figure 5.48 address the accuracy of the post-processing system versus the real-time system for the middle of Test 4. During this time window, the post-processing system miscounts a burst only once. The real-time processing system properly identified these muscle bursts. Figure 5.49 and Figure 5.50 address the accuracy of the post-processing system for the end of Test 4. During this time window, the post-processing system continues to over-count the number of muscle bursts, whereas the real-time processing system properly identifies the muscle bursts.

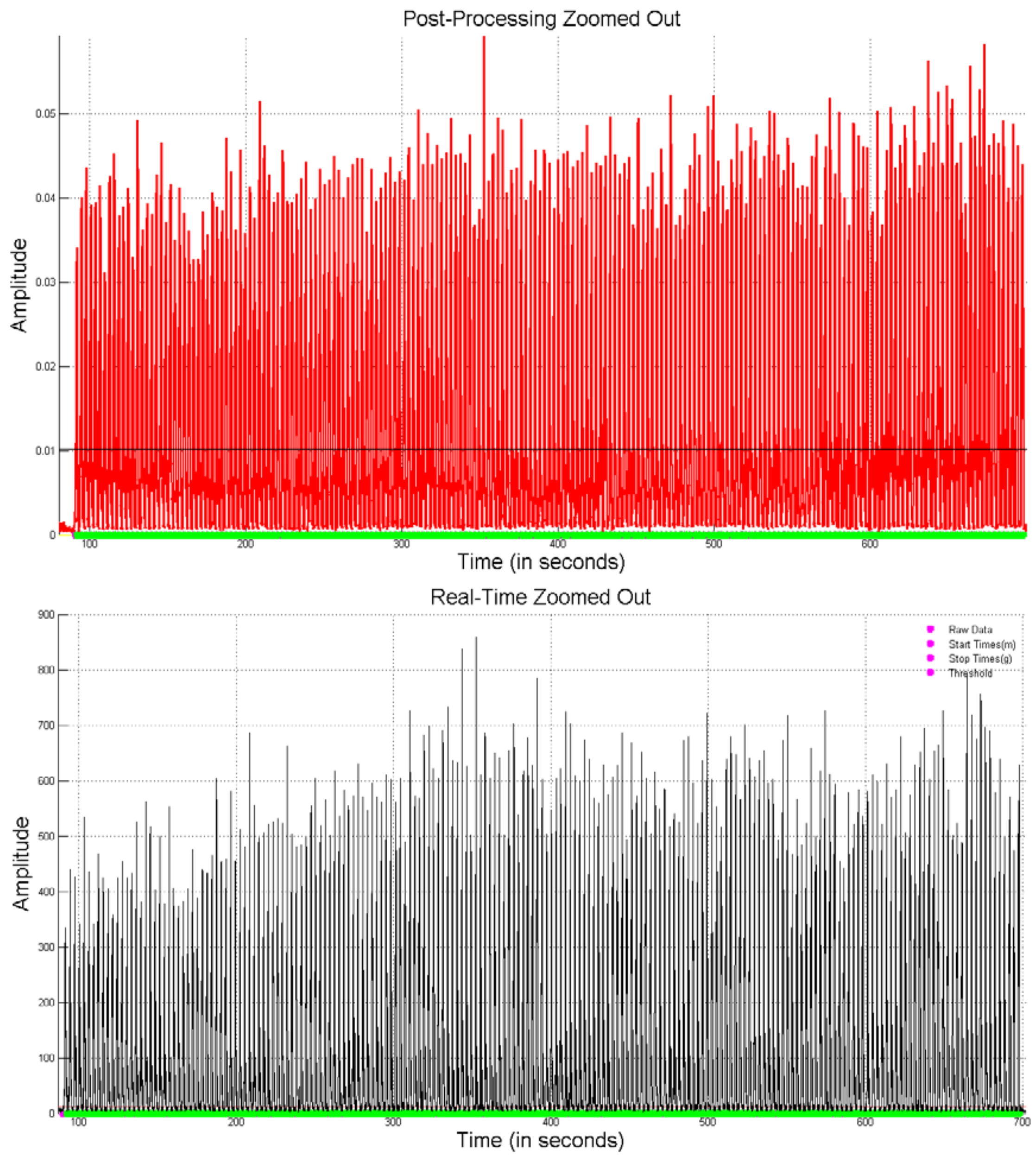


Figure 5.44 Full experiment with start/stop times identified post-experiment (top) and in real time (bottom).

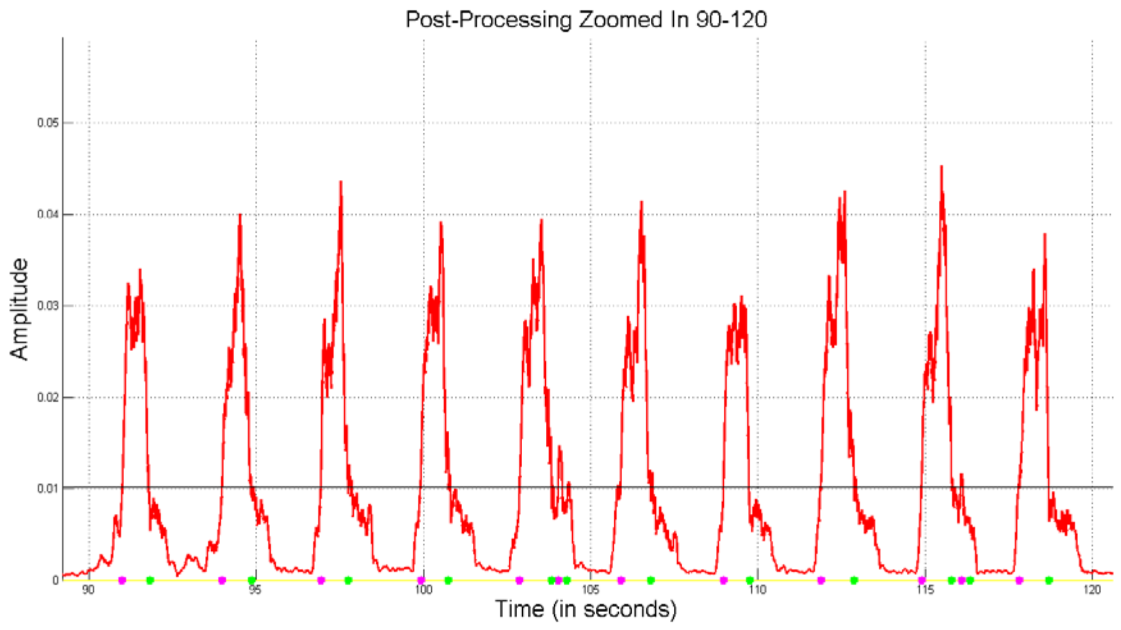


Figure 5.45 Post-processing of Test 4 data over the time interval of [90, 120] seconds.

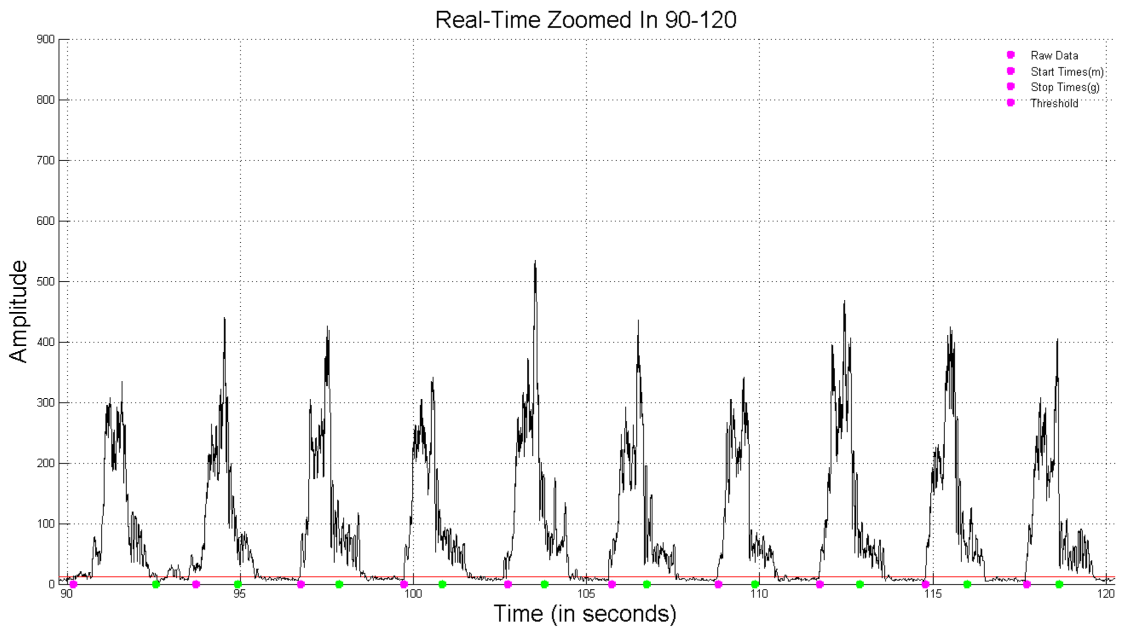


Figure 5.46 Real-time processing of Test 4 data over the time interval of [90, 120] seconds.

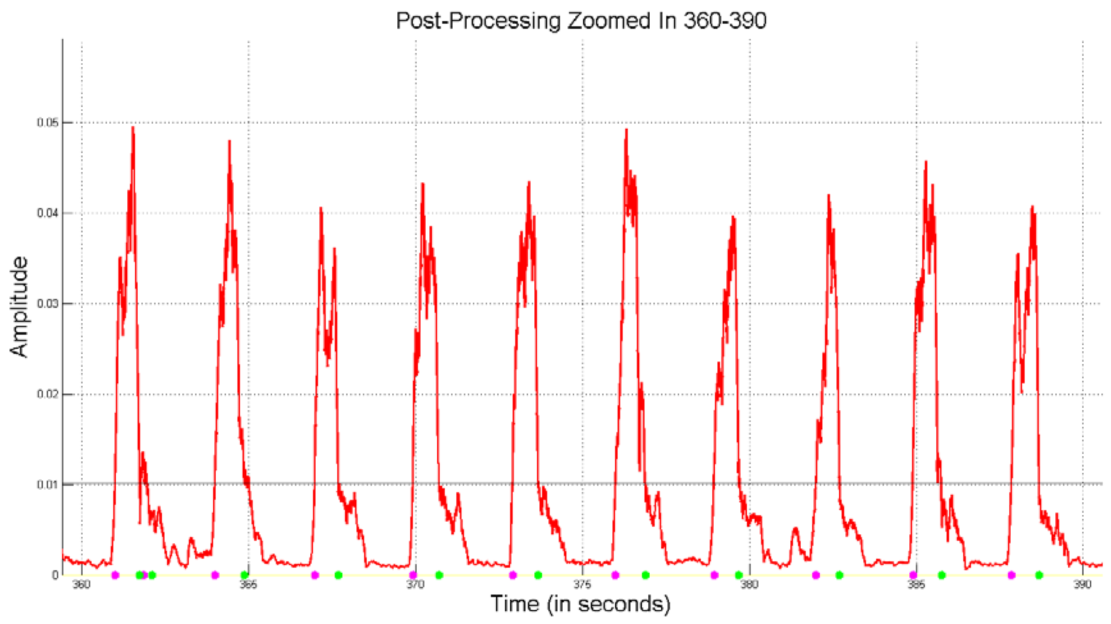


Figure 5.47 Post-processing of Test 4 data over the time interval of [360, 390] seconds.

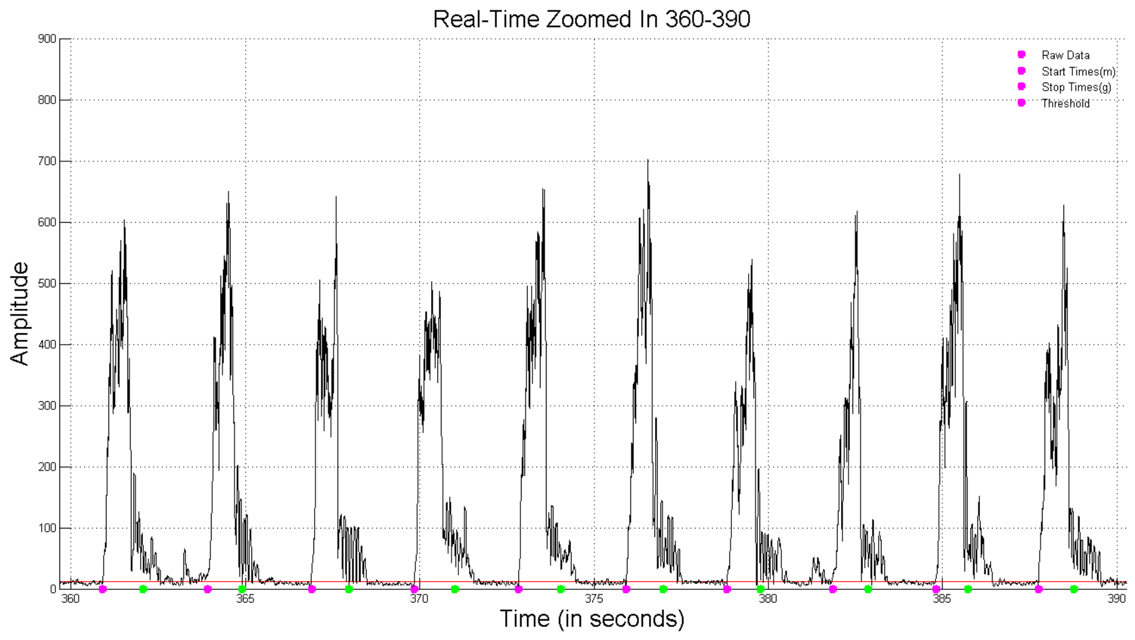


Figure 5.48 Real-time processing of Test 4 data over the time interval of [360, 390] seconds.

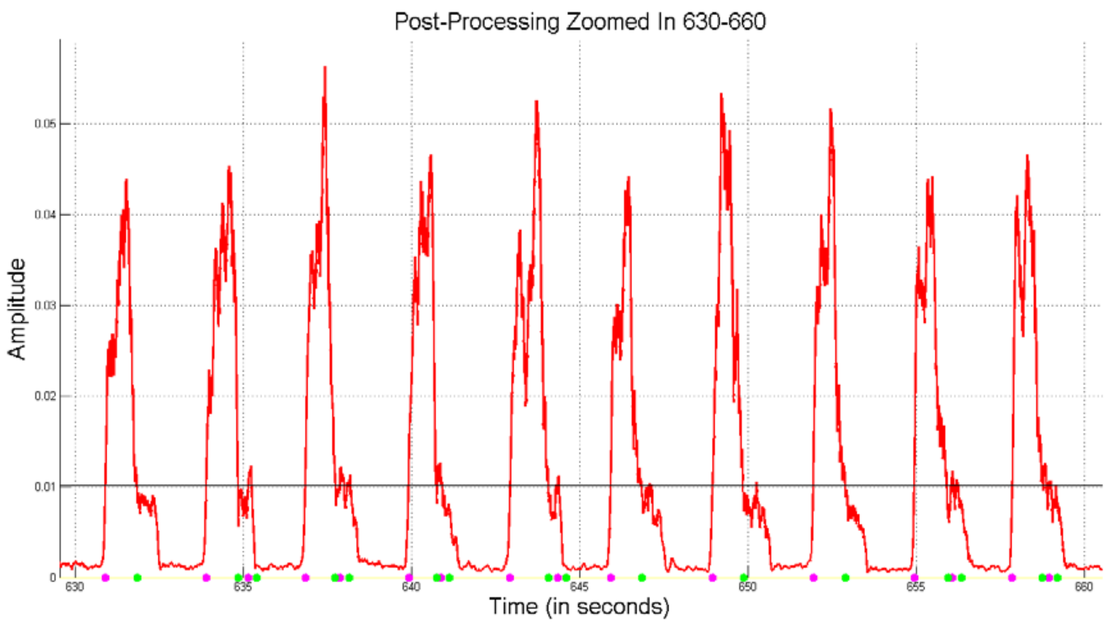


Figure 5.49 Post-processing of Test 4 data over the time interval of [630, 660] seconds.

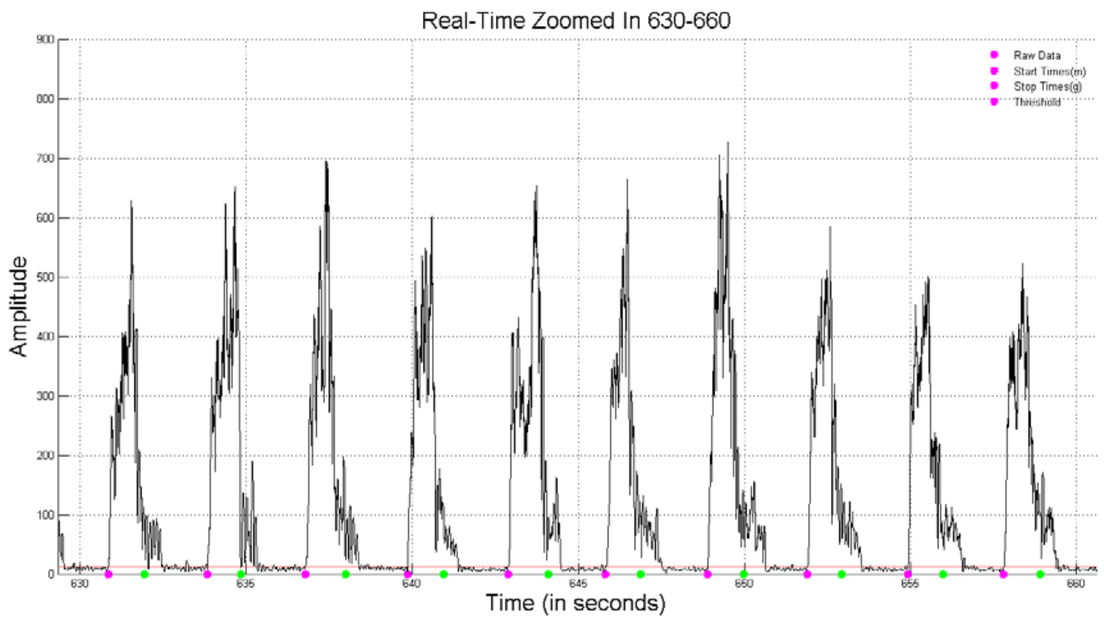


Figure 5.50 Real-time processing of Test 4 data over the time interval of [630, 660] seconds.

These time windows illustrate another scenario where the real-time processing system more accurately identified muscle bursts than the post-processing system. As with Tests 2 and 3, this improved burst identification is due to the fact that the real-time system uses a moving threshold, while the post-processing system employs a static threshold. This allows the real-time system to quickly adapt to the changing level of the EMG.

Figure 5.51 illustrates the Test 4 time-domain fatigue parameters that were calculated in the MATLAB burst movie script. Consistent with the results on the LabVIEW screen, the RMS value and integrated EMG increase with fatigue, whereas the zero crossing rate noticeably decreases. As seen in the LabVIEW screen, the parameters all hold constant during the last half of the experiment. Figure 5.52 illustrates the Test 4 frequency-domain fatigue parameters calculated in the MATLAB burst movie script. The MPF and MDF (both magnitude and power) decrease at the beginning of Test 4, which is consistent with the results on the LabVIEW screen. The total spectral magnitude noticeably increases, while the skewness and kurtosis have a very small increase.

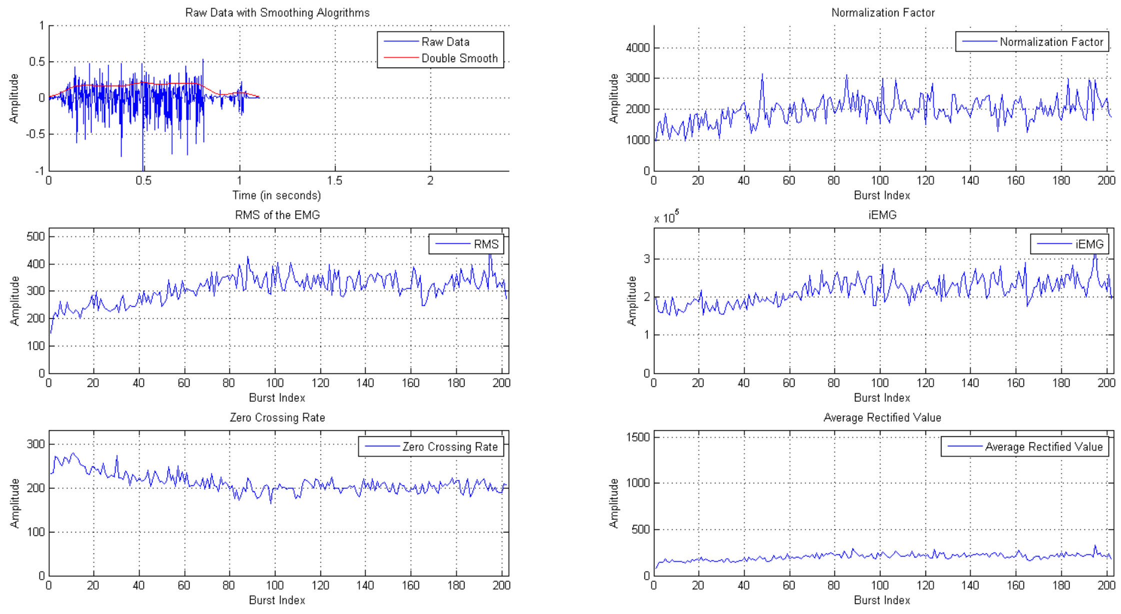


Figure 5.51 Time-domain movie frame for Test 4.

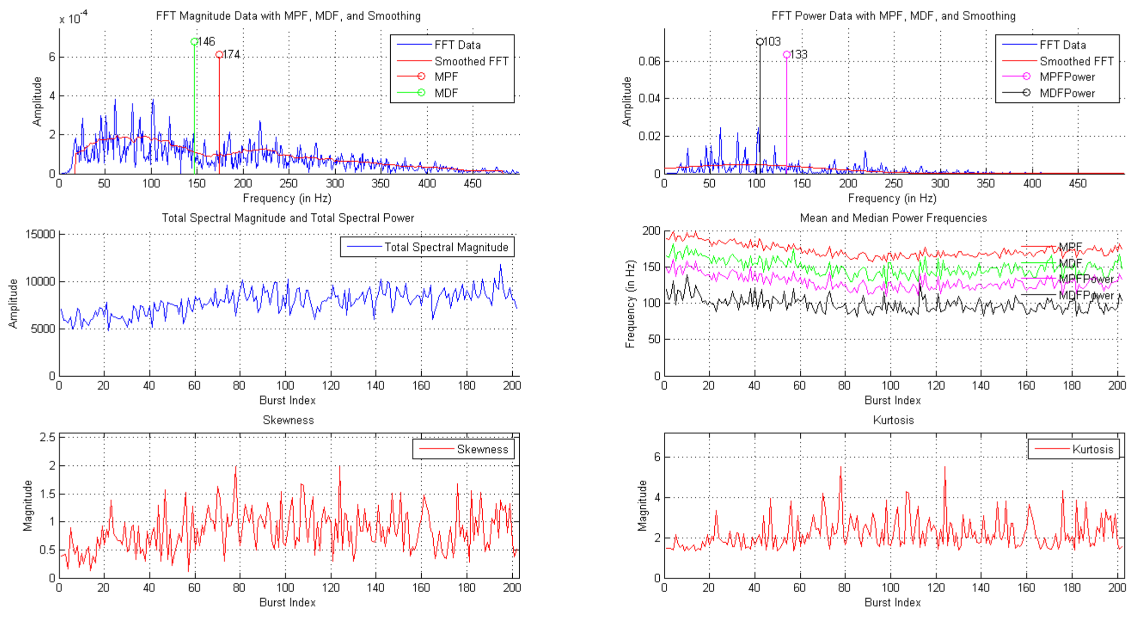


Figure 5.52 Frequency-domain movie frame for Test 4.

Results Summary

The secondary goal of this work was to evaluate the effectiveness of traditional and non-traditional parameters to identify the onset of fatigue in repetitive (as opposed to sustained) muscle contractions. From the LabVIEW results in all four tests, the RMS values increased, and the MPF and MDF of each broadband spectrum decreased as the subject approached fatigue. The MPF and MDF for the low and high bands, the skewness, and the kurtosis did not exhibit significant trends during the increasing force experiment. However, the MPF and MDF parameters for the high frequency band and, to some extent, skewness and kurtosis, showed substantive fatigue trends in the constant force tests. From the burst movies, the iEMG, average rectified value, and total spectral magnitude increased with subject fatigue in all tests, while the zero crossing rate, skewness, and kurtosis exhibited slight upward trends in the constant force tests.

CHAPTER 6: CONCLUSION AND FUTURE WORK

This work focused on detection of fatigue onset in forearm muscles. The main goal of this work was to develop a system to detect and process muscle bursts in real-time. The resulting LabVIEW VI accomplished this goal and was supplemented with a MATLAB script that aided our investigation into new fatigue trends. While the LabVIEW system has the ability to support live experiments and replayed experiments, the replay functionality is limited to only the EMG data. With its current abilities, however, the real-time system out-performed the previous post-processing system in terms of burst count and local accuracy at the beginning, middle, and end of experiments. The tests also noted that some fatigue parameters, such as RMS value, iEMG, MDF, and MPF, are better suited for monitoring fatigue during repetitive movements in the forearm.

This work focused on merging a LabVIEW-based data acquisition system with a collection of post-processing tools to create an integrated real-time system acquisition and analysis system. The hand-forearm ergometer system now automatically gathers the data from the subject, calculates and displays fatigue parameters, and saves the raw and calculated data for further use. Having accomplished the goal of automating the processing of these data, the logical next step would be to actively predict fatigue and task failure. Doing so would create a fully automated system that would have the ability to warn a researcher or a subject of the subject's impending task failure due to muscle fatigue. To accomplish this next task, the start/stop point accuracy must be perfected. While a substantial upgrade was made with this work, the muscle burst detection will always be a top upgrade priority.

Another future work target would be analyzing the fast- and slow-twitch muscles in the forearm and their effect on the resulting muscle bursts' magnitude and power frequency spectra. Observing both types of muscles' effects on the overall skewness and kurtosis may lead to better picture of how forearm muscles respond to fatigue. Furthermore, separating the slow-twitch muscles' frequency spectrum from the fast-twitch muscles' frequency spectrum would also have the potential to greatly increase our understanding of how forearm muscles react to fatigue.

CHAPTER 7: REFERENCES

[1] Thomas Barstow, Carl Ade, Steve Warren, Chris Lewis, "Standardized "Pre-flight" Exercise Tests to Predict Performance During Extravehicular Activities in a Lunar Environment ", ed. Step-2 proposal, Research and Technology Development to Support Crew Health and Performance in Space Exploration Missions, NASA Human Research Program, Exploration Systems Mission Directorate, Johnson Space Center, Houston, TX, 7/1/2010–6/30/2013: Kansas State University.

[2] Thomas W Nolan, "System changes to improve patient safety," *BMJ: British Medical Journal,* vol. 320, p. 771, 2000.

[3] Maureen J MacDonald, Heather L Naylor, Michael E Tschakovsky, and Richard L Hughson, "Peripheral circulatory factors limit rate of increase in muscle O2 uptake at onset of heavy exercise," *Journal of Applied Physiology,* vol. 90, pp. 83-89, 2001.

[4] Darren P Casey, Michael J Joyner, Paul L Claus, and Timothy B Curry, "Hyperbaric hyperoxia reduces exercising forearm blood flow in humans," *American Journal of Physiology-Heart and Circulatory Physiology,* vol. 300, pp. H1892-H1897, 2011.

[5] Vladimir Shushakov, Christian Stubbe, Antje Peuckert, Volker Endeward, and Norbert Maassen, "The relationships between plasma potassium, muscle excitability and fatigue during voluntary exercise in humans," *Experimental Physiology,* vol. 92, pp. 705-715, 2007.

[6] Y. Ohira, A. Ito, and S. Ikawa, "Hemoconcentration during isotonic handgrip exercise," *Journal of Applied Physiology,* vol. 42, pp. 744-745, May 1, 1977 1977.

[7] Hisao Oka, "Estimation of muscle fatigue by using EMG and muscle stiffness," in *Engineering in Medicine and Biology Society, 1996. Bridging Disciplines for Biomedicine. Proceedings of the 18th Annual International Conference of the IEEE,* 1996, pp. 1449-1450.

[8] Daniel J. Green, William Bilsborough, Louise H. Naylor, Chris Reed, Jeremy Wright, Gerry O'Driscoll, *et al.*, "Comparison of forearm blood flow responses to incremental handgrip and cycle ergometer exercise: relative contribution of nitric oxide," *The Journal of Physiology,* vol. 562, pp. 617-628, 2005.

[9] M. Marina, J. Porta, L. Vallejo, and R. Angulo, "Monitoring hand flexor fatigue in a 24-h motorcycle endurance race," *Journal of Electromyography and Kinesiology,* vol. 21, pp. 255-261, 4// 2011.

[10] Dana M. Gude, "Automated Hand-Forearm Ergometer Data Acquisition and Analysis System," Electrical Engineering, Electrical & Computer Engineering, Kansas State University, 2011.

[11] Chen Jia Phillip Kuehl, Dana Gude, Ryan Broxterman, Thomas Barstow, & Steve Warren, "Real-Time Processing of Electromyograms in an Automated Jand-Forearm Ergometer Data Collection and Analysis System," presented at the NASA Human Research Program Investigators' Workshop, Galveston, TX, 2014.

[12] Chen Jia Phillip Kuehl, Dana Gude, Ryan Broxterman, Thomas Barstow, & Steve Warren, "Real-Time Processing of Electromyograms ni an Automated Hand-Forearm Ergometer Data Collection and Analysis System," presented at the 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, Sheraton Chicago Hotel and Towers, Chicago, IL, USA, 2014.

[13] G. M. Hagg, "Comparison of Different Estimators of Electromyographic Spectral Shifts during Work when Applied on Short Test Contractions," *Medical & Biological Engineering & Computing,* vol. 29, pp. 511-516, 1991.

[14] Daniel R. Rogers & Dawn T. MacIsaac, "EMG-based Muscle Fatigue Assessment During Dynamic Contractions Using Principal Component Analysis," *Journal of Electromyography and Kinesiology,* vol. 21, 2011.

[15] M. Amann, M.W. Eldridge, A.T. Lovering, M.K. Stickland, D.F. Pegelow, & J.A. Dempsey, "Arterial Oxygenation Influences Central Motor Output and Exercise Performance via Effects on Peripheral Locomotor Muscle Fatigue in Humans," *The Journal of Physiology,* vol. 575, pp. 937-952, 2006.

[16] C.J. De Luca, "The Use of Surface Electromyography in Biomechanics," *Journal of Applied Biomechanics,* vol. 13, pp. 135-163, 1997.

[17] M. Gonzalez-Izal, A. Malanda, E. Gorostiaga, & M. Izquierdo, "Electromyographic Models to Assess Muscle Fatigue," *Journal of Electromyography and Kinesiology,* vol. 22, pp. 201-512, 2012.

[18] A.F. Mannion P. Dolan, & M.A. Adams, "Fatigue of the Erector Spinae Muscles. A Quantitative Assessment Using "Frequency Banding" of the Surface Electromyography Signal," *Spine,* vol. 20, pp. 149-159, 1995.

[19] P. Duhamel and M. Vetterli, "Fast Fourier transforms: A tutorial review and a state of the art.," *Signal Processing*, vol. 19, pp. 259–299, 1990.

[20] Madeleine M. Lowery & Mark J. O'Malley, "Analysis and Simulation of Changes in EMG Amplitude During High-Level Fatiguing Contractions," *IEEE Transactions on Biomedical Engineering,* vol. 50, pp. 1052-1062, 2003.

[21] Philip A. Parker Dawn T. MacIsaac, Kevin B. Englehart, & Daniel R. Rogers, "Fatigue Estimation With a Multivariable Myoelectric Mapping Function," *IEEE Transactions on Biomedical Engineering,* vol. 53, pp. 694-700, 2006.

[22] Vladimir Medved Mario Cifrek, Stanko Tonkovic, & Sasa Ostojic, "Surface EMG Based Muscle Fatigue Evaluation in Biometrics," *Clinical Biomechanics,* vol. 24, pp. 327-340, 2009.

[23] Gerardo Bilotto Holger Broman, & Carlo J. De Luca, "Myoelectric Signal Conduction Velocity and Spectral Parameters: Influence of Force and Time," *Journal of Applied Physiology,* vol. 58, pp. 1428-1437, 1985.

[24] L. Arendt-Nielsen & K.R. Mills, "The Relationship Between Mean Power Frequency of the EMG Spectrum and Muscle Fibre Conduction Velocity," *Electroencephalography and Clinical Neurophysiology,* vol. 60, 28 August 1984.

[25] NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm, March 20, 2015.

[26] Przemyslaw Baranski, http://www.mathworks.com/matlabcentral/fileexchange/35122-gaussian-fit,, Feb. 4th, 2012. (2012, Feb. 4th). *Gaussian Fit*.

APPENDIX A: MATLAB SCRIPTS

Burst Movie

Main Code

```
function movieFile = BurstMovie(fftDataFile, signalFile,
startStopDataFile)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Author: Phillip Kuehl               %
% Kansas State University             %
% Date: April 16, 2015                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%                       About this Function Call
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%
% This function is used to create a time-domain and frequency-domain
movie
% file which shows each individual burst in an experiment.
%
% This function requires the input files to be in the following format:
%
% fftDataFile - This file is the frequency power spectra of each
individual
% burst in columns of frequency and amplitude as 2-column pairs. The
% columns for each burst are detailed as such:
    % Column 1 : Frequency
    % Column 2 : Amplitudes of the power frequency spectrum of the
current
    %            burst
%
% signalFile - This file is the original EMG data set in which you
% are wanting to anaylise. The columns of raw data are detailed as
such:
    % Column 1 : Time
    % Column 2 : Force
    % Column 3 : Displacement
    % Column 4 : raw EMG
%
% startStopDataFile - This file contains the start and stop points in
the
% raw EMG data where the individual bursts are located as well as the
% parameter information for each burst. Each row represents a burst and
% each column represents a different parameter. The columns are
detailed as
% such:
    % Column 1 : Start Point
    % Column 2 : Stop Point
    % Column 3 : Mean Power Frequency of All Bands (0-500 Hz)
    % Column 4 : Median Power Frequency of All Bands (0-500 Hz)
    % Column 5 : Mean Power Frequency of Low Bands (5-30 Hz)
    % Column 6 : Median Power Frequency of Low Bands (5-30 Hz)
    % Column 7 : Mean Power Frequency of High Bands (95-500 Hz)
    % Column 8 : Median Power Frequency of High Bands (95-500 Hz)
    % Column 9 : Work
    % Column 10: Power
    % Column 11: Kurtosis
    % Column 12: Skewness
%
% This function will display a series of two figures with the
corresponding
% information on them. Please do not disturb the program while it is
% running. It may take quite a bit of time to save each image into each
% movie file. If you want to plot burst during a certain time frame,
% uncomment the section entitled "Change Start and Stop Time of
Experiment"
% and choose the times for variables start and stop in the following
% format:
%       start = (seconds * sampling frequency)
%       stop = (seconds * sampling frequency)
%%
%Initialize Data Points

%Load in the FFT data file and signal data file
[~,fftData] = hdrload(fftDataFile);
[~,signalData] = hdrload(signalFile);
[~,startStopData] = hdrload(startStopDataFile);

%If time values are present, remove times values
if (size(signalData,2) == 4)
    samplingFrequency = 1/...
                          (signalData(3,1)-signalData(2,1));
    signalData = signalData(:,2:4);
else
    samplingFrequency = 1000;
end

%%
%Change Start and Stop Time of Experiment
% %Manually figure out where the experiment starts and stops
% start = 400*samplingFrequency;
% stop = 1345*samplingFrequency;
% startInbetween = startStopData((startStopData(:,1)>start),1);
% lenStartDelete = max(size(startStopData)) -
max(size(startInbetween));
% startInbetween = startInbetween(startInbetween(:,1)<stop);
% lenStopDelete = max(size(startStopData)) - lenStartDelete -
max(size(startInbetween));
% stopInbetween=startStopData((startStopData(:,1)>start),2);
% stopInbetween=stopInbetween(startInbetween(:,1)<stop);
%
% startStopData = [startInbetween stopInbetween];
% [fftRows,fftCols] =size(fftData);
% fftData = fftData(:,(2*lenStartDelete-1):(2*(fftCols/2-lenStopDelete-
1)));

%%

%Get dimensions of the data
[numRows, numColumns] = size(fftData);

%Initialize variables
lengthColumn = 0;
count = 1;
currentColumn = 2;
frameRate = 1;
fig1 = figure(1);
fig2 = figure(2);
upperBound = 500;
lowerBound = 5;
[numBursts,~] = size(startStopData);
maxFFT = 0;
maxFFTPower= 0;
totalSpectralMagnitude(numBursts) = 0;
totalSpectralPower(numBursts)=0;
MDF(numBursts) = 0;
MPF(numBursts) = 0;
MDFPower(numBursts) = 0;
MPFPower(numBursts) = 0;
normalizationBurstFactor(numBursts) = 0;
emgRMS(numBursts) = 0;
iEMG(numBursts) = 0;
zeroCrossings = 0;
zeroCrossingRate(numBursts) = 0;
averageRectifiedValue(numBursts) = 0;

%Initialize .avi file for FFT video
file = fftDataFile;
[pathstr, oldfile, ext] = fileparts(file);
fileName = [oldfile '_burstMovie.avi'  ];
videoObjectFFT = VideoWriter(fileName);
videoObjectFFT.FrameRate = frameRate;

%Initialize .avi file for Raw Data video
file = signalFile;
[pathstr, oldfile, ext] = fileparts(file);
fileName = [oldfile '_burstMovie.avi'  ];
videoObjectBurst = VideoWriter(fileName);
videoObjectBurst.FrameRate = frameRate;

%Get all axes values
xValueBurst = max(startStopData(:,2)-startStopData(:,1));
yValueBurst = max(max(signalData(:,3),abs(min(signalData(:,3)))));
xValueFFT = max(max(fftData(:,1:2:end)));
%yMinValueSpectral = min(sum(fftData(5:end,2:2:end)));
```

```
yMaxValueSpectral = max(max(sum(fftData(5:end,2:2:end).^(1/2))));
% yMinValueSpectralPower= min(sum(fftData(5:end,2:2:end).^2));
% yMaxValueSpectralPower= max(max(sum(fftData(5:end,2:2:end).^2)));

kurtosisFFT(numBursts) = 0;
skewnessFFT(numBursts) = 0;

%Get FFT axis value
while (currentColumn < (numColumns + 1))

    %Adjust the size of the data to be plotted (e.g. remove unnecessary
    %zeros)
    while(count < numRows)

        %Check for the end of the data stream in the current column
        if(fftData(count,currentColumn) == 0 &&
fftData(count,currentColumn - 1) == 0 && count > 1)
            if(lengthColumn == 0)
                lengthColumn = count;
            end
        end

        %Iterate count
        count = count + 1;
    end

    if(lengthColumn == 0)
       lengthColumn = count;
    end

    %Get current start and stop times of the burst and the burst data
from
    %the data files
    startTime = startStopData(currentColumn/2,1);
    stopTime = startStopData(currentColumn/2,2);
    currentBurstData = signalData(startTime:stopTime,3);

    %Get frequency indices and fft data from data file
    frequencyIndex = fftData(1:lengthColumn, currentColumn - 1);
    currentFFTData = fftData(1:lengthColumn, currentColumn);

    %Remove DC values, find total spectral power, and normalize FFT
Data
    currentFFTData(1:4) = [0 0 0 0];
    normalizedFFTData = currentFFTData.^(1/2) / (sum(currentFFTData));
    normalizedFFTPowerData = currentFFTData / (sum(currentFFTData));

    %Hold maximum value of normalized FFT Data
    maxFFT = max(max(normalizedFFTData,maxFFT));
    maxFFTPower = max(max(normalizedFFTPowerData,maxFFTPower));

    %Calculate Skewness and Kurtosis
    [gaussianArray] = GenerateGaussian( frequencyIndex,
normalizedFFTPowerData );
    skewnessFFT(currentColumn/2) = skewness(gaussianArray);
    kurtosisFFT(currentColumn/2) = kurtosis(gaussianArray);

    %Calculate EMG RMS and iEMG
    emgRMS(currentColumn/2) = (1/length(currentBurstData) *
sum(currentBurstData.^2)).^(1/2);
    iEMG(currentColumn/2) = sum(abs(currentBurstData));

    %Increment currentColumn
    currentColumn = currentColumn + 2;

    %Reset counter variables
    count = 1;
    lengthColumn = 0;
end

yMaxRMS = max(emgRMS);
yMaxIEMG = max(iEMG);
yMaxSkewness = max(skewnessFFT);
yMaxKurtosis = max(kurtosisFFT);
currentColumn = 2;
yValueFFT = maxFFT;
yValueFFTPower = maxFFTPower;

%%
%Smooth Data

%Get the smoothed data for plotting
signal_abs = abs(signalData(:,3));
%stand_noise = mean(signalData(1:5000,3));

% Calculate the moving average power of the signal, 101 points average
signal_length = length(signal_abs);
smooth_signal = zeros(signal_length,1);

%Median Filter Data
smooth_signal = medfilt1(signal_abs,101);

%Moving Average Filter
for i = 51:signal_length - 50
    signal_abs_101 = signal_abs(i-50:i+50);
    smooth_signal(i) = sqrt(sum(signal_abs_101.^2)/101);
end
smooth_signal_abs = abs(smooth_signal);

% Calculate the moving average again for signal
smooth_signal_final = zeros(signal_length,1);

for i = 51:signal_length - 50
    smooth_signal_final(i) = mean(smooth_signal_abs(i-50:i+50));
end



%%
%Open Video Objects
```

```
open(videoObjectFFT)
open(videoObjectBurst)
%%
%Plot Data

%plot each FFT and save it as a frame
while (currentColumn < (numColumns + 1))

    %Adjust the size of the data to be plotted (e.g. remove unnecessary
    %zeros)
    while(count < numRows)

        %Check for the end of the data stream in the current column
        if(fftData(count,currentColumn) == 0 &&
fftData(count,currentColumn - 1) == 0)
            if(lengthColumn == 0)
                lengthColumn = count-1;
            end
        end

        %Iterate count
        count = count + 1;
    end

    if(lengthColumn == 0)
       lengthColumn = count;
    end

    %Get current start and stop times of the burst and the burst data
from
    %the data files
    startTime = startStopData(currentColumn/2,1);
    stopTime = startStopData(currentColumn/2,2);
    currentBurstData = signalData(startTime:stopTime,3);

    %Get frequency indices and fft data from data file
    frequencyIndex = fftData(1:lengthColumn, currentColumn - 1);
    currentFFTData = fftData(1:lengthColumn, currentColumn);

    %Remove DC values and find total spectral power
    currentFFTData(1:4) = [0 0 0 0];
    totalSpectralMagnitude(currentColumn/2) =
sum(currentFFTData.^(1/2));
    totalSpectralPower(currentColumn/2) = sum(currentFFTData);

    %Normalize Burst Data and FFT Data
    normalizationBurstFactor(currentColumn/2) =
max(max(currentBurstData,abs(min(currentBurstData))));
    normalizedBurstData = currentBurstData /
normalizationBurstFactor(currentColumn/2);
    normalizedFFTData = currentFFTData.^(1/2) / (sum(currentFFTData));
    normalizedFFTPowerData = currentFFTData / (sum(currentFFTData));

    %Calculate the moving average for FFT Magnitude data
    fft_length = length(normalizedFFTData);
    smooth_fft = zeros(fft_length,1);

    for i = 21:fft_length - 20
        fft_41 = normalizedFFTData(i-20:i+20);
        smooth_fft(i) = sqrt(sum(fft_41.^2)/41);
    end

    %Calculate the moving average for FFT Power data
    fft_power_length = length(normalizedFFTPowerData);
    smooth_power_fft = zeros(fft_power_length,1);

    for i = 21:fft_power_length - 20
        fft_41 = normalizedFFTPowerData(i-20:i+20);
        smooth_power_fft(i) = sqrt(sum(fft_41.^2)/41);
    end

    freqMedian = MedianPowerFrequency(frequencyIndex,
normalizedFFTData, lowerBound, upperBound);
    freqMean = MeanPowerFrequency(frequencyIndex, normalizedFFTData,
lowerBound, upperBound);

    freqMedianPower = MedianPowerFrequency(frequencyIndex,
normalizedFFTPowerData, lowerBound, upperBound);
    freqMeanPower = MeanPowerFrequency(frequencyIndex,
normalizedFFTPowerData, lowerBound, upperBound);

    %Calculate Skewness and Kurtosis
    [gaussianArray] = GenerateGaussian( frequencyIndex,
normalizedFFTPowerData );

    %Calculate Zero Crossing Rate
    [zeroCrossings,~] = size(find(diff(sign(currentBurstData))~=0));
%find number of zero crossings
    zeroCrossingRate(currentColumn/2) =
zeroCrossings/length(currentBurstData)*samplingFrequency;

    %Calculate Average Rectified Value
    averageRectifiedValue(currentColumn/2) =
mean(abs(currentBurstData));

    %Fill mean and median power and magnitude frequency arrays
    MDF(currentColumn/2) = freqMedian;
    MPF(currentColumn/2) = freqMean;

    MDFPower(currentColumn/2) = freqMedianPower;
    MPFPower(currentColumn/2) = freqMeanPower;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Start Plotting

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    clf(fig1)
    clf(fig2)
    %Set Figure to fullscreen
    screenSize = get(0,'ScreenSize');
    set(fig1,'position',screenSize);
    set(fig2,'position',screenSize);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %FFT Data Video

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    figure(fig1)

    %FFT Magnitude with MDF, MPF, and Smoothed FFT
    subplot(321)
    hold on

    %Plot the current FFT burst data
    plot(frequencyIndex, normalizedFFTData );
    plot(frequencyIndex, smooth_fft,'r');

    %Plot the Mean Magnitude Frequency
    stem(freqMean,0.9*yValueFFT,'r')

    %Plot the Median Magnitude Frequency
    stem(freqMedian,yValueFFT,'g')

    %Plot numbers for the MPF and MDF
    plot(freqMedian, yValueFFT ,'w.')
    text(freqMedian, yValueFFT , [' ' num2str(floor(freqMedian))])
    plot(freqMean, 0.9*yValueFFT ,'w.')
    text(freqMean, 0.9*yValueFFT , [' ' num2str(floor(freqMean))])
    hold off

    title('FFT Magnitude Data with MPF, MDF, and Smoothing');
    xlabel('Frequency (in Hz)');
    ylabel('Amplitude');
    legend('FFT Data','Smoothed FFT','MPF','MDF');
    axis([0 xValueFFT 0 1.1*yValueFFT]);
    grid on

    %FFT Power with MDF, MPF, and Smoothed FFT
    subplot(322)
    hold on

    %Plot the current FFT burst data
    plot(frequencyIndex, normalizedFFTPowerData );
%     plot(frequencyIndex, smooth_power_fft,'r');
    plot(frequencyIndex, gaussianArray,'r');

    %Plot the Mean Power Frequency
    stem(freqMeanPower,0.9*yValueFFTPower,'m')
    %Plot the Median Power Frequency
    stem(freqMedianPower,yValueFFTPower,'k')

    %Plot numbers for the MPF and MDF
    plot(freqMedianPower, yValueFFTPower ,'w.')
    text(freqMedianPower, yValueFFTPower , ['  '
num2str(floor(freqMedianPower))])
    plot(freqMeanPower, 0.9*yValueFFTPower ,'w.')
    text(freqMeanPower, 0.9*yValueFFTPower , ['  '
num2str(floor(freqMeanPower))])
    hold off

    title('FFT Power Data with MPF, MDF, and Smoothing');
    xlabel('Frequency (in Hz)');
    ylabel('Amplitude');
    legend('FFT Data','Smoothed FFT','MPFPower','MDFPower');
    axis([0 xValueFFT 0 1.1*yValueFFTPower]);
    grid on

    subplot(323)
    hold on
    plot(totalSpectralMagnitude(1:currentColumn/2));
    title('Total Spectral Magnitude and Total Spectral Power');
    xlabel('Burst Index');
    ylabel('Amplitude');
    legend('Total Spectral Magnitude');
    axis([0 numBursts 0 1.3*yMaxValueSpectral])
    grid on

%     [ax,~,~] =
plotyy(1:(currentColumn/2),totalSpectralMagnitude(1:currentColumn/2),1:
(currentColumn/2),totalSpectralPower(1:currentColumn/2));
%
%     title('Total Spectral Magnitude and Total Spectral Power');
%     xlabel(ax(1),'Burst Index');
%     ylabel(ax(1),'Amplitude');
%     ylabel(ax(2),'Amplitude');
%     legend('Total Spectral Magnitude','Total Spectral Power');
%     set(ax(1),'YLim',[0.9*yMinValueSpectral 1.1*yMaxValueSpectral]);
%     set(ax(2),'YLim',[0.9*yMinValueSpectralPower
1.5*yMaxValueSpectralPower]);

    %Mean and Median Magnitude Frequency
    subplot(324)

    hold on
    plot(MPF(1:currentColumn/2),'r');
    plot(MDF(1:currentColumn/2),'g');
    plot(MPFPower(1:currentColumn/2),'m');
    plot(MDFPower(1:currentColumn/2),'k');
    hold off

    title('Mean and Median Power Frequencies');
    xlabel('Burst Index');
    ylabel('Frequency (in Hz)');
    legend('MPF','MDF','MPFPower','MDFPower');
    legend BOXOFF
    axis([0 numBursts 0 200]);
    grid on

    %Skewness
    subplot(325)

    plot(skewnessFFT(1:currentColumn/2),'r');

    title('Skewness');
    xlabel('Burst Index');
    ylabel('Magnitude');
    legend('Skewness');
    axis([0 numBursts 0 1.3*yMaxSkewness]);
    grid on

    %Kurtosis
    subplot(326)

    plot(kurtosisFFT(1:currentColumn/2),'r');

    title('Kurtosis');
    xlabel('Burst Index');
    ylabel('Magnitude');
    legend('Kurtosis');
    axis([0 numBursts 0 1.3*yMaxKurtosis]);
    grid on


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Burst Data Video

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    figure(fig2)

    %Normalized Raw Data with smoothing and twice smoothing waveforms
    subplot(321)

    hold on
    time=0:(1/samplingFrequency):(numel(normalizedBurstData)-
1)/samplingFrequency;
    plot(time,normalizedBurstData);

%plot(time,smooth_signal_abs(startTime:stopTime)/normalizationBurstFact
or(currentColumn/2),'m')

plot(time,smooth_signal_final(startTime:stopTime)/normalizationBurstFac
tor(currentColumn/2),'r')
    %plot(zeros(length(smooth_signal_abs(startTime:stopTime)),1),'k')

%plot(stand_noise*ones(length(smooth_signal_abs(startTime:stopTime)),1)
,'r')
    hold off
```

```
title('Raw Data with Smoothing Alogrithms');
xlabel('Time (in seconds)');
ylabel('Amplitude');
legend('Raw Data', 'Double Smooth');
axis([0 xValueBurst/samplingFrequency -1 1]);
grid on

%Normalization Factor for Bursts
subplot(322)
hold on
plot(normalizationBurstFactor(1:(currentColumn/2)));
hold off

title('Normalization Factor');
xlabel('Burst Index');
ylabel('Amplitude');
legend('Normalization Factor');
axis([0 numBursts 0 1.5*yValueBurst]);
grid on

%FFT Magnitude with MDF, MPF, and Smoothed FFT
subplot(323)

plot(emgRMS(1:(currentColumn/2)));

title('RMS of the EMG');
xlabel('Burst Index');
ylabel('Amplitude');
legend('RMS');
axis([0 numBursts 0 1.1*yMaxRMS]);
grid on

%FFT Power with MDF, MPF, and Smoothed FFT
subplot(324)

plot(iEMG(1:(currentColumn/2)));

title('iEMG');
xlabel('Burst Index');
ylabel('Amplitude');
legend('iEMG');
axis([0 numBursts 0 1.1*yMaxIEMG]);
grid on

%Zero Crossing Rate
subplot(325)

plot(zeroCrossingRate(1:currentColumn/2));
title('Zero Crossing Rate');
xlabel('Burst Index');
ylabel('Amplitude');
legend('Zero Crossing Rate');
axis([0 numBursts 0 samplingFrequency/2])
grid on
```

```
    %Average Rectified Value
    subplot(326)

    plot(averageRectifiedValue(1:currentColumn/2));
    title('Average Rectified Value');
    xlabel('Burst Index');
    ylabel('Amplitude');
    legend('Average Rectified Value');
    axis([0 numBursts 0 0.3*yValueBurst])
    grid on


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Capture Frames

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    writeVideo(videoObjectFFT,getframe(fig1));
    writeVideo(videoObjectBurst,getframe(fig2));
    %Iterate the current Column
    currentColumn = currentColumn + 2;

    %Reset count and lengthColumn
    count = 1;
    lengthColumn = 0;
end

% %Iterate the current Column
% currentColumn = currentColumn + 2;
%
% %Reset count and lengthColumn
% count = 1;
% lengthColumn = 0;

close(videoObjectFFT);

close all
clear all
end
```

GaussFit

```
function [sigma, mu] = gaussfit( x, y, sigma0, mu0 )
% [sigma, mu] = gaussfit( x, y, sigma0, mu0 )
% Fits a guassian probability density function into (x,y) points using
iterative
% LMS method. Gaussian p.d.f is given by:
% y = 1/(sqrt(2*pi)*sigma)*exp( -(x - mu)^2 / (2*sigma^2))
% The results are much better than minimazing logarithmic residuals
%
% INPUT:
% sigma0 - initial value of sigma (optional)
% mu0 - initial value of mean (optional)
```

```
%
% OUTPUT:
% sigma - optimal value of standard deviation
% mu - optimal value of mean
%
% REMARKS:
% The function does not always converge in which case try to use
initial
% values sigma0, mu0. Check also if the data is properly scaled, i.e.
p.d.f
% should approx. sum up to 1
%
% VERSION: 23.02.2012
%
% EXAMPLE USAGE:
% x = -10:1:10;
% s = 2;
% m = 3;
% y = 1/(sqrt(2*pi)* s ) * exp( - (x-m).^2 / (2*s^2)) + 0.02*randn( 1,
21 );
% [sigma,mu] = gaussfit( x, y )
% xp = -10:0.1:10;
% yp = 1/(sqrt(2*pi)* sigma ) * exp( - (xp-mu).^2 / (2*sigma^2));
% plot( x, y, 'o', xp, yp, '-' );


% Maximum number of iterations
Nmax = 50;

if( length( x ) ~= length( y ))
    fprintf( 'x and y should be of equal length\n\r' );
    exit;
end

n = length( x );
x = reshape( x, n, 1 );
y = reshape( y, n, 1 );

%sort according to x
X = [x,y];
X = sortrows( X );
x = X(:,1);
y = X(:,2);

%Checking if the data is normalized
dx = diff( x );
dy = 0.5*(y(1:length(y)-1) + y(2:length(y)));
s = sum( dx .* dy );
if( s > 1.5 | s < 0.5 )
    fprintf( 'Data is not normalized! The pdf sums to: %f.
Normalizing...\n\r', s );
    y = y ./ s;
end

X = zeros( n, 3 );
```

```
X(:,1) = 1;
X(:,2) = x;
X(:,3) = (x.*x);


% try to estimate mean mu from the location of the maximum
[ymax,index]=max(y);
mu = x(index);

% estimate sigma
sigma = 1/(sqrt(2*pi)*ymax);

if( nargin == 3 )
    sigma = sigma0;
end

if( nargin == 4 )
    mu = mu0;
end

%xp = linspace( min(x), max(x) );

% iterations
for i=1:Nmax
%    yp = 1/(sqrt(2*pi)*sigma) * exp( -(xp - mu).^2 / (2*sigma^2));
%    plot( x, y, 'o', xp, yp, '-' );

    dfdsigma = -1/(sqrt(2*pi)*sigma^2)*exp(-((x-mu).^2) / (2*sigma^2));
    dfdsigma = dfdsigma + 1/(sqrt(2*pi)*sigma).*exp(-((x-mu).^2) /
(2*sigma^2)).*((x-mu).^2/sigma^3);

    dfdmu = 1/(sqrt(2*pi)*sigma)*exp(-((x-mu).^2)/(2*sigma^2)).*(x-
mu)/(sigma^2);

    F = [ dfdsigma dfdmu ];
    a0 = [sigma;mu];
    f0 = 1/(sqrt(2*pi)*sigma).*exp( -(x-mu).^2 /(2*sigma^2));
    a = (F'*F)^(-1)*F'*(y-f0) + a0;
    sigma = a(1);
    mu = a(2);

    if( sigma < 0 )
        sigma = abs( sigma );
        fprintf( 'Instability detected! Rerun with initial values
sigma0 and mu0! \n\r' );
        fprintf( 'Check if your data is properly scaled! p.d.f should
approx. sum up to \n\r' );
        exit;
    end
end
```

GaussFit License