Novel Applications for Phasor Measurement Units and Synchrophasor Data

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Bradley Kerwin Greene

August 2013

Acknowledgements

I would like to thank my family and friends for the encouragement and support to complete this thesis. Also the Power IT Lab, Dr. Yilu Liu, Dr. Richard Conners, Dr. Kevin Tomsovic, Dr. Kai Sun, and the faculty at the University of Tennessee for the great advice and mentorship throughout the program. I would also like to thank Dr. Matthew Gardner, Mr. Kyle Thomas and the rest of the Dominion Virginia Power Transmission Operations team for supporting some of the work in this thesis.

Abstract

The last decade has seen an intensified effort towards an improved, technologically advanced electric grid. This effort is largely in part to the need for cleaner and renewable sources of energy. Another motivator for this "smarter grid" is the need for a more reliable and efficiently operated of the wide scale electric infrastructure. The impact of these changes can expect to be seen at both the transmission and distribution level. At the transmission level phasor measurement units and synchrophasor data has emerged as one of the most enabling technologies for the smart grid movement. These devices measure and time synchronize, the magnitude and phase angle of the electrical quantities over wide areas of an electric grid. These measurements are then made available to utilities and system operators to facilitate new and improved applications that foster enhanced grid reliability, security and efficiency.

Synchrophasor data provides increased visibility into the phenomenon occurring within the electric grid, as measurements are taken at rates up to 60 Hz. This is a stark improvement compared to traditional supervisory control and data acquisition systems which operate at a rate of once every 2 to 5 seconds. This increased data rate therefore has the potential to feed applications that would not have traditionally been seen with SCADA, and improve the operation of those that are already functional. These applications can vary from real-time operation to off-line applications for post-event analysis and planning. Some of the more well known of these appplications are state estimation, inter-area oscillation, and wide area monitoring and control.

There are however, some novel applications of synchrophasor data less publicized and deployed within the industry. One such application developed by the Power IT lab at the University of Tennessee involves using synchrophasor data to authenticate digital audio recordings. Another application developed by Dominion Virginia Power involves the automatic calibration of instrument transformers across a system using this synchrophasor data. This thesis outlines these novel applications and the work I performed to facilitate their implementations.

Table of Contents

Chapter 1 Introduction 1

Chapter 2 Background Review 4

2.1 Synchrophasor Measurements 4

2.2 Phasor Measurement Units 6

2.3 Communications Requirements 8

2.4 Phasor Data Concentrators 9

2.5 Data Storage Requirements 11

Chapter 3 FNET and Audio Authentication 13

3.1 Project Overview 13

3.2 FNET Architecture 14

3.2.1 Frequency Disturbance Recorders (FDRs) 17

3.2.2 Information Management System 17

3.2.3 FNET-OpenPDC Integration 20

3.3 ENF Analysis Method 26

3.4 Graphical User Interface 32

3.5 Tampering Detection 38

Chapter 4 Three-phase Instrument Transformer Calibration 42

4.1 Project Overview 42

4.2 System Model 44

4.3 Methodology 46

4.4 Load Tap Changer (LTC) position Monitoring using a PMU 48

4.4.1 Hardware Description 51

4.4.2 Software Description 53

4.4.3 Results 55

Chapter 5 Conclusion 57

List of References 60

Appendix 64

Vita 89

List of Figures

Figure 1: Sinusoidal Waveform and Phasor Representation 5

Figure 2: PMU installation schematic 7

Figure 3: North American Power Grid Interconnections and PMU locations 8

Figure 4: Levels of PDCs 11

Figure 5: Location of FDRs across North America 15

Figure 6: FNET Structure 16

Figure 7: Frequency Disturbance Recorder 18

Figure 8: Hardware Architecture of FDR 19

Figure 9: OpenPDC adapter based architecture 21

Figure 10: FNET-OpenPDC Integrated Framework 23

Figure 11: PMU Connection Tester showing FNET data 24

Figure 12: OpenPDC Manager showing FNET data 25

Figure 13: Historian Data Viewer showing archived phasor data 27

Figure 14: 10 minute comparison of FDR data from FNET and OpenPDC 28

Figure 15: Frequency response of voice recording showing ENF 28

Figure 16: Principle of STFT 30

Figure 17: Extracted ENF and FDR Data Comparison 31

Figure 18: MSE for extracted ENF from recording 32

Figure 19: Original code structure for ENF Analysis 33

Figure 20: New code structure for ENF Analysis 34
Figure 21: ENF extraction GUI 35
Figure 22: Sample results of ENF analysis using GUI 37
Figure 23: Variation of ENF and phase angle at tampering point 39
Figure 24: Three phase model of two bus system for transducer calibration 45
Figure 25: LTC position monitoring schematic 49
Figure 26: Lab setup for LTC position monitoring scheme testing 50
Figure 27: LTC position monitoring schematic in parallel with SCADA application 51
Figure 28: Algorithm used to convert voltage to LTC position with SEL 421 54

Chapter 1

Introduction

The American electric grid is regarded as the most complex system in the world. It is made up of hundreds of thousands of generators and transformers, which transfer power over millions of miles of transmission lines to supply millions of varying load types. It is necessary for the generated power to constantly match the ever-changing load, so the procedures to ensure this balancing act has developed and become mature over time. With the changing power landscape due to increased load requirements, more restricting regulation policies, renewable generation, and a progressively aging infrastructure; the efficient, reliable and secure operation of the grid is becoming more and more difficult. New and improved methods must be developed to ensure the grid operates at optimal cost, and with a minimized risk of failure.

This notion of optimizing and modernizing the way the electric grid operates is being referred to as the creation of a ‘smart grid’. One must understand that the American electric grid is interconnected in large segments, and within these interconnections any event has an effect on the entire system. These events can range from a generator trip, addition of new transmission lines or changing loads to name a few. The three interconnections are the Eastern Interconnection, the Western Interconnection and ERCOT, which is primarily Texas. It becomes evident the vast areas upon which even the slightest change at any point within the grid, will have an effect on the overall system. It therefore becomes necessary for wide area monitoring

and control systems such as SCADA to play a role in trying to manage all of these widely dispersed components under constantly changing conditions.

Current SCADA systems however observe grid conditions every 2 to 6 seconds, which is too slow to track many dynamic events in the grid. For a system that operates at 60 times per second, one can imagine the speed at which quantities change, and any system in place to monitor these quantities must function at a similar rate. SCADA systems also do not monitor key values like the phase angles of voltages and currents, which can provide further insight into phenomena that's occurring within the grid. As time progresses and new and improved applications are developed, values like phase angles will become more and more important in the implementation of these very complex schemes. Furthermore, SCADA data is often not time-synchronized and not wide area accessible; hence restricting its usefulness in some applications that can provide wide area monitoring and control. Therefore SCADA does not provide real-time wide area visibility into grid operations across a wide area or region.

Synchrophasor technology has been developed to bridge the gap of some of the short comings seen in SCADA systems. In comparison, synchrophasor systems allow the collection and sharing of high-speed, real time, time-synchronized grid condition data across an entire system or interconnection. They provide an increased visibility into the dynamics of the system, giving system operators and planners unprecedented insight into what is happening on the grid at high resolution, time synchronized over a wide area, and where needed in real time. These capabilities are facilitated through a range of software applications use the synchrophasor data to implement a range of functions. The software applications can use full-resolution real-time data,

down-sampled real time data or historical data, along with grid models to support both operating and planning functions, and can be categorized according to one of the three groups below [1]:

(1) Applications to support real-time grid operations by providing wide-area visualization and increased state awareness.
(2) Applications to improve system planning and analysis, including power system performance baselining, event analysis and model validation.
(3) Response-based control applications that use real-time wide area information to take automated control actions on the power system.

The real-time applications require real-time data collection and processing with immediate analysis and visualization or used as control signals for real-time controls applications. Common examples of these are state estimators, and alarming and event detection. Planning and post event analysis applications use archived data and may be performed off-line weeks after the data is initially collected. Common examples of these are baselining system performance, and model validation and calibration. This thesis outlines two novel applications less well known in the field. FNET audio authentication, which can be roughly classed under Category 2, uses historical synchrophasor data to validate the genuineness of digital recordings. And the Dominion transformer calibration, which can be roughly classed under Category 1, uses real-time wide area information to perform calculations and ultimately enhance situational awareness.

Chapter 2

Background Review

To gain a proper understanding of the novel applications outlined in this thesis, general knowledge of the systems in place to measure and report synchrophasor data is first required. Following chapters are dedicated to outlining the details of the specific applications, but this chapter is intended to introduce the concepts of synchrophasor measurements, and the devices used to capture them, the phasor measurement unit (PMU). It also introduces how these measurements are concentrated and eventually stored for later use.

2.1 Synchrophasor Measurements

The voltage magnitudes and phase angles of all the system buses are the state variables of the power system. All state variables must be uniquely determined for effective management of the power grid [2]. If these values are known, active and reactive power flow through the system could be calculated using these voltage magnitudes and phase angles, along with system impedance. Determining these variables has proven to be a challenge due to the vast distances separating the system nodes. However, the solution to these challenges has emerged to be synchrophasor measurements. A phasor is a complex number that represents both the magnitude and phase angle of voltage and current sinusoidal waveforms (60 Hz) at a specific point in time. Figure 1 illustrates a sinusoidal waveform and its phasor representation. Synchrophasors are

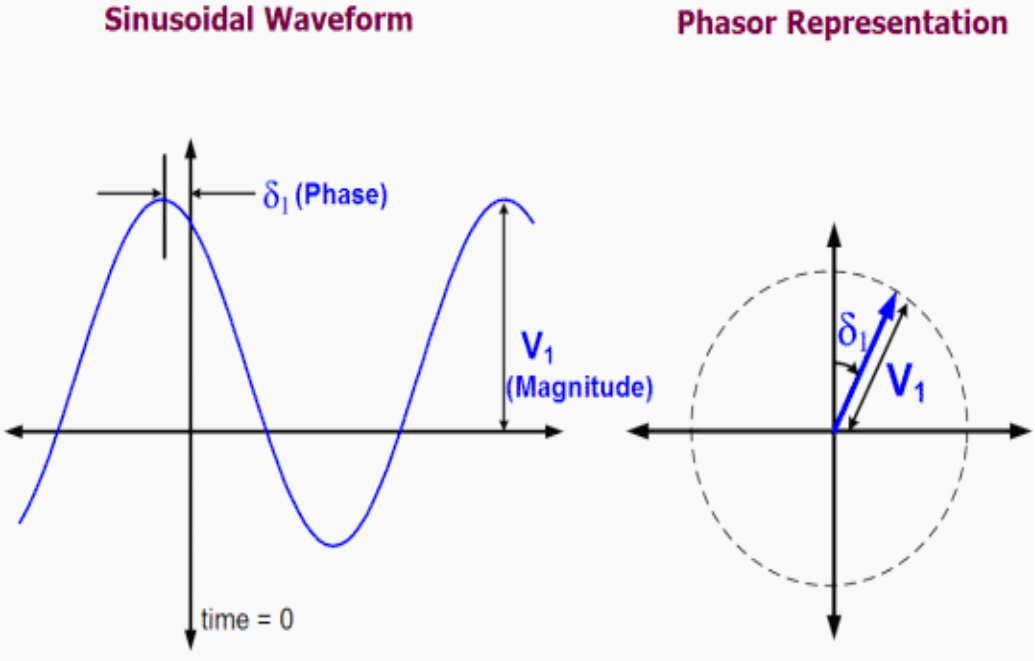


Figure 1: Sinusoidal Waveform and Phasor Representation [1]

precise time-synchronized measurements of certain parameters, including phasors, on the electricity grid.

PMUs measure voltage, current and frequency and calculate phasors, and this suite of time-synchronized grid condition data is called phasor data. Each phasor measurement is time-stamped against Global Positioning System universal time; when a phasor measurement is times-tamped, it is called a synchrophasor. This allows measurements taken by PMUs in different locations or by different owners to be synchronized and time-aligned, then combined to provide a precise, comprehensive view of an entire region or interconnection. The widely accepted standard as to how these measurements should be defined, time-stamped and communicated is the IEEE C37.118 [3]. Other standards such as FNET and BPA PDCStream are defined however, and these also facilitate important applications.

2.2 Phasor Measurement Units

A phasor measurement unit(PMU) is considered to be any device that uses digital signal processors that measure 50/60 Hz AC voltage and current waveforms. The analog AC waveforms are digitized by analog to digital converters and a phase-lock oscillator, and a Global positioning System (GPS) reference source provides high-speed time-synchronized sampling. A PMU calculates line frequencies, voltage and current phasors at a high sampling rate and streams the data, along with the associated time-stamp, over networked communication media. These devices take samples at rates up to 60 frames per second. These devices can take several forms, as the PMU capability need not be the sole function of the device. For example digital fault recorders (DFRs) primary purpose is to serve as a relay, although PMU capabilities can be incorporated.

PMUs are typically installed in a substation or at a power plant. Each phasor requires three separate electrical connections (one for each phase), to either measure a current (from a line or bank) or a voltage (from either line or bus PTs). Figure 2 shows a setup for a typical PMU installation. According to the North American Synchrophasor Initiative (NASPI), PMUs are scheduled for installation at the following locations by 2014 [4].

- Major transmission interconnections and interfaces
- All 500kV and above substations and most 200kV and above substations
- Key generating plants (all > 500 MW) in generator switchyards, even on some individual generators

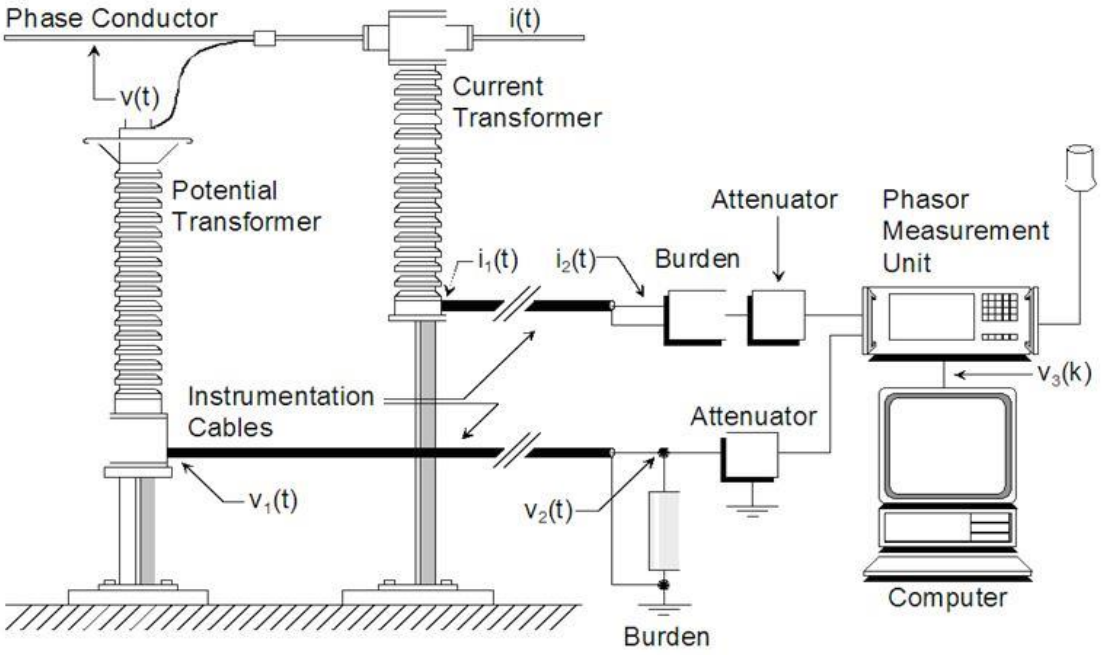


Figure 2: PMU installation schematic [1]

- Major load centers
- Large wind generators, solar and storage locations
- Other locations to assure visibility in areas with sparse PMU coverage

Figure 3 shows installed PMUs across the North American power grid as of September 2009. Other architectures exist however, such as frequency disturbance recorders (FDRs) which are part of the FNET framework. These devices measure single phase voltage phasors and can be installed at any 120V wall outlet, more on this concept later in Chapter 3.

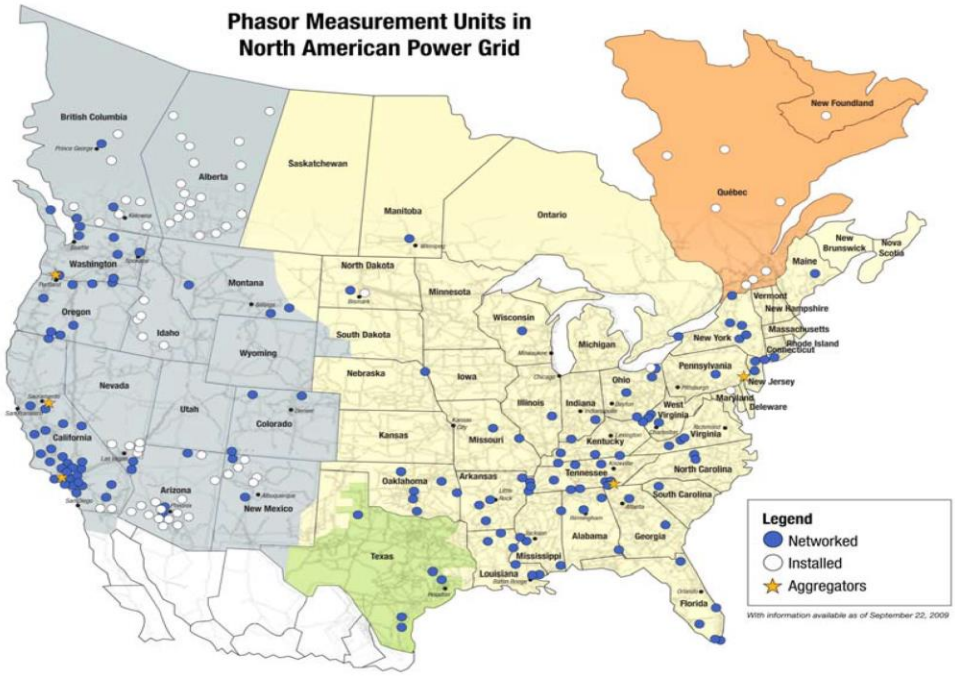


Figure 3: North American Power Grid Interconnections and PMU locations [1]

2.3 Communications Requirements

The measurements made and transmitted by PMUs can range from large to small, depending on the type and location of the device, and the applications it is intended to feed. These measurements typically include frequency, real, reactive power, raw voltage and current phasors for each transmission element. PMUs can also output individual phase RMS values, which is useful for detecting unbalanced conditions that may exist in the system. The current rule of thumb is that each PMU makes about 20 measurements, where sixteen of these are 8 phasor measurements with a magnitude and angle for each. The IEEE C37.118 standard allows for a

variety of sampling rates ranging from 10 to 120 per second, however most PMUs sample at a rate of 30 measurements per second.

As a result of these large number of measurements combined with the high sampling rates, a very robust communication system is needed to handle synchrophasor data. Consider a network in which one PMU sends 12,000 samples per second to the host computer. Each sample consists of a time stamp (minimum of four bytes) and the data value (typically eight bytes), and four more bytes for data storage overhead. Thus 40 PMUs feed data into the network at 192,000 bytes per second. This translates to about 15.5 gigabytes per day of disk space or about 5.6 terabytes (TB) per year. Assuming a super-PDC at an ISO may concentrate 10 companies' PDCs, this number multiplies to 56 TB a year per ISO. A network of nine super- PDCs would generate a cumulative total of 509 TB a year, or half a petabyte of data [1]. The appropriate communication media must ensure that these data rates can be successfully handled without compromising the system.

2.4 Phasor Data Concentrators

Phasor data concentrators (PDCs) collect phasor data from multiple PMUs or other PDCs and align the data according to time-stamps. The time-synchronized dataset created can then be passed on to other information systems. A PDC also performs data quality checks and flags missing or problematic data (waiting for a set period of time, if needed, for all the data to come in before sending the aggregated dataset on). Some PDCs also store phasor data and can down sample it so that phasor data can be fed directly to applications that use data at slower sample

rates, such as a SCADA system. While there are no formal standards yet for the functional requirements of a PDC, it is generally accepted that the minimum requirements of a PDC include the ability to [1]:

- Correlate phasor data by time tag and then broadcast the combined data to other systems
- Conform to streaming protocol standards (e.g., IEEE C37.118) for both the phasor data inputs and the combined data output stream
- Verify the integrity and completeness of data streams from PMUs and properly handle data anomalies
- Buffer input data streams to accommodate the differing times of data delivery from individual PMUs.

Figure 4 shows the three levels of PDCs. The functions of a PDC can vary depending on its role or its location between the source PMUs and the higher-level applications. A local PDC may be located physically close to PMUs (typically at a substation) to manage the collection and communication of time-synchronized data from local PMUs, send it to higher level concentrators, and store the data for use within the substation. A super-concentrator (super-PDC) operates on a regional scale, handling phasor measurements from several hundred PMUs and multiple PDCs. It collects and correlates phasor data from remote PDCs and PMUs and makes them available as a coherent, time-synchronized dataset to applications such as wide-area monitoring and visualization software, energy management systems and SCADA applications. A super-PDC also feeds the data into a central database for long-term data archiving (data historian

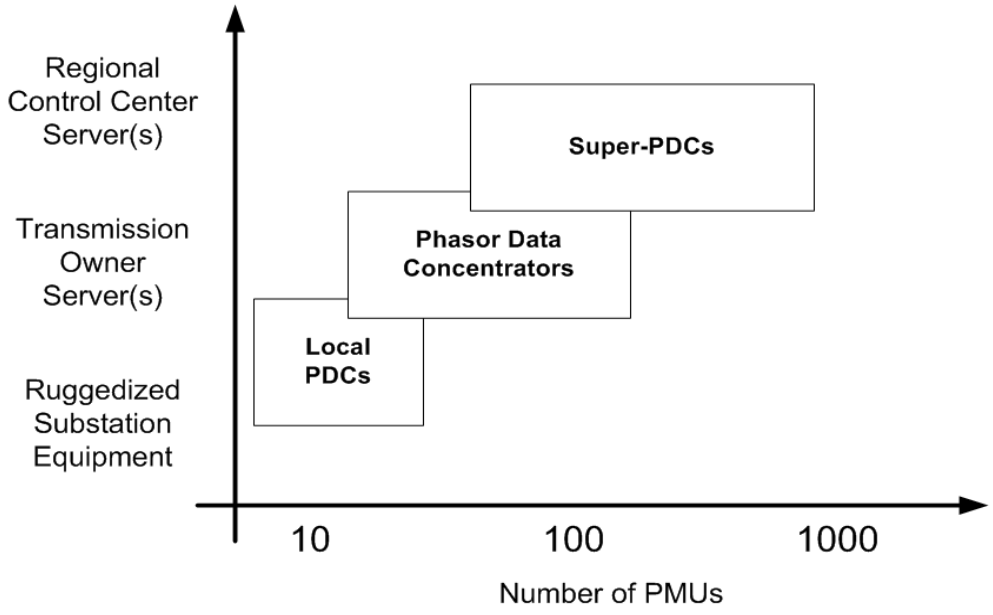


Figure 4: Levels of PDCs [1]

function). Super-PDCs are software implementations, running on mainstream server hardware, as these larger devices need to scale rapidly to serve growing utility and regional deployment of PMUs and diverse phasor data applications.

2.5 Data Storage Requirements

Data historians, optimized to effectively handle large volumes of time-stamped measurement data, are typically used to save and retrieve phasor data. Traditional relational databases can be used to manage phasor data for a short period of history or for implementations that contain a small number of PMUs. Distributed non-relational databases are used to manage big datasets by large internet firms like Google and Yahoo. These systems have some relational database features that can be used effectively with data systems that contain many, many

terabytes of information. They divide the data up into smaller blocks that are processed in parallel. This technology is being explored for use in synchrophasor data systems as well.

The volume of streaming phasor data can make storage requirements add up quickly. For ease of retrieval and use, data is stored as time-value pairs either in a data historian or in a relational database. Not counting data structure overheads (that include the point ID), the minimum possible size storage is 10 bytes per time-value pair (4 bytes for time, 4 for data and 2 for flags) within a historian. Thus a PDC that collects data from 100 PMUs of 20 measurements each at 30 samples per second, will require a little over 50 GB/day14 or 1.5 TB/month of storage within a historian system. Long-term storage requirements for a relational database would be significantly higher.

Chapter 3

FNET and Audio Authentication using Synchrophasor Data

3.1 Project Overview

Forensic audio refers to audio material that may provide evidence in legal proceedings. The authenticity of audio evidence must be verified to ensure it has not been edited in any way since the original production. The field of audio authentication is a complex forensic science which forms an important function of a forensic audio laboratory. Strategies for the analysis of analog recording are well established; however methods for the now more commonplace digital recordings are not so reputable. The signal characteristics associated with digital recordings are completely different than that of analog, and additional techniques for assessing the integrity of a suspect digital recording are required [5]. One such technique for digital recordings is the Electric Network Frequency (ENF) criterion. ENF embedded in digital recordings relate to the frequency of the power system at the location where the recording was taken.

ENF variation is a naturally-occurring wide-area phenomenon that is caused by imbalance between the generated power and the load on the network. The variations that do occur under stable operating conditions are the same for all points in the network or grid. The frequency variations that do occur are random. As such, for long enough periods of time, it is assumed that these patterns are unique and, hence, can be used in the verification of digital recordings [6]. The ENF criterion is based on estimating the ENF that may occur in a digital

recording. This estimated frequency is then compared either to the ENF that has been directly measured from the electrical network or to the ENF taken from a recording of known validity. Thus, extracted ENF data from a digital recording, the target *ENF*, may be compared to an available reference database of ENF information, called the reference ENF, allowing the date and time of the recording to be ascertained. For this project the target ENF is acquired via a technique called the Short-Time Fourier Transform, and the reference ENF is synchrophasor data acquired from the FNET database.

3.2 FNET Architecture

The Frequency Monitoring Network (FNET) is a wide-area frequency monitoring system currently maintained and operated by the Power IT Laboratory at the University of Tennessee. The system makes use of measurements taken by frequency disturbance recorders (FDRs) deployed across all three North American interconnections. Figure 5 shows the locations of FDRs as of June 2011. Each FDR itself is actually a single-phase PMU in the sense that it measures the voltage phase angle, amplitude, and frequency from a single-phase voltage source. The fact that the FDRs are installed at the 120-V distribution level reduces their manufacturing and installation costs significantly [7]. The frequency measurement algorithm in the FDRs makes use of phasor analysis and signal resampling techniques, which have virtually zero algorithm error in the range between 52 and 70 Hz. The produced frequency measurement is a universal parameter across the entire interconnected system [8], an ideal characteristic for the audio authentication application.

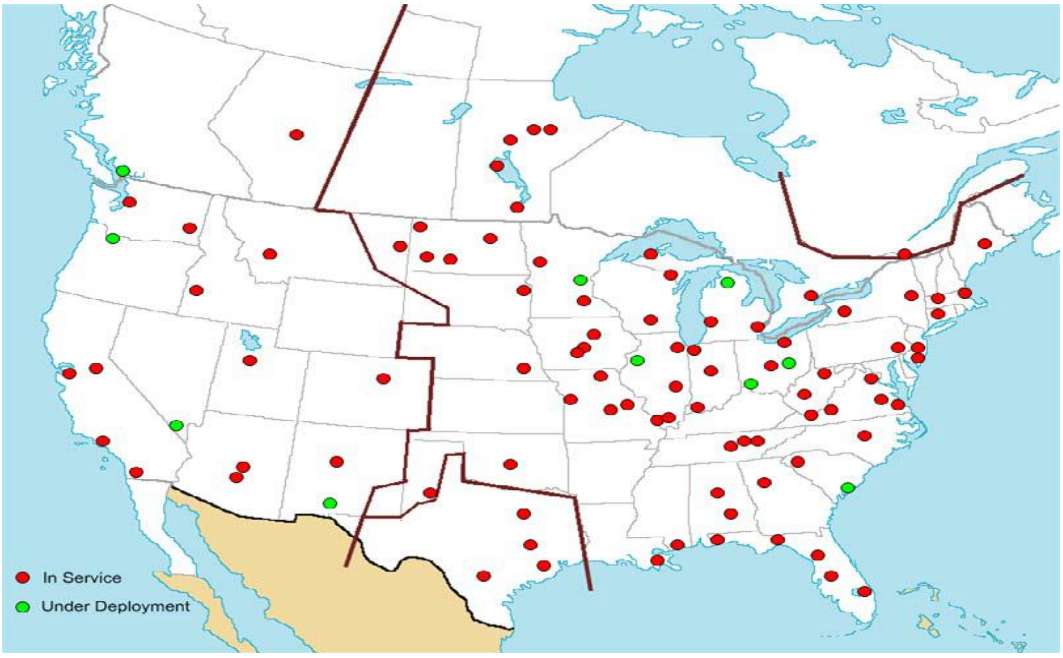


Figure 5: Location of FDRs across North America

It can also provide information about generation electromechanical transients, generation demand dynamics, and system operations, such as load shedding, break reclosing, and capacitor bank switching. Hence allowing frequency monitoring using FNET to be as informative at the distribution level as it is at the transmission level. Phasor calculation in the FDRs is derived from real-time voltage signal sampling and its discrete Fourier transform–based complex representation. Voltage phase angles and magnitudes are also measured by FDRs, which can provide useful information for power system event recognition and status estimation. Figure 6 illustrates the key components of the FNET system. Phasor measurements from the North American power grids are collected by widely installed sensors, known as FDRs, and are transmitted via the Internet to a local client or remote data center. In most of the cases, FDR data are transmitted to the FNET data center for processing and long-term storage [7].

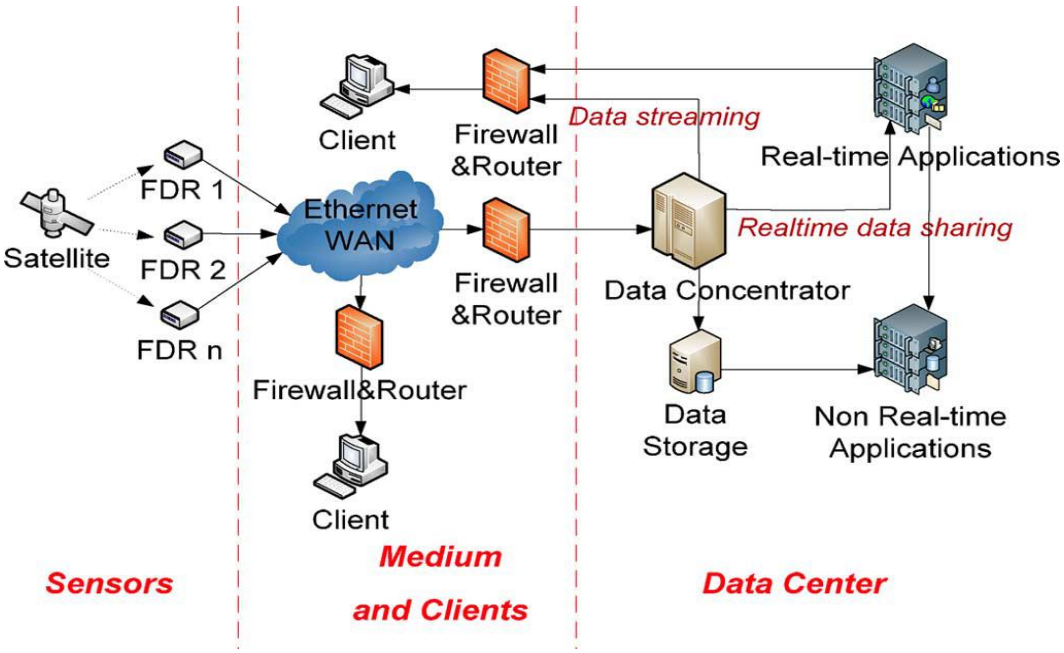


Figure 6: FNET Structure [7]

FDRs can be installed at a variety of locations, such as power plants, substations, office buildings, and private residences. Each FDR is equipped with a GPS receiver, which is used to provide the accurate timing signal needed for synchrophasor calculation. The units are powered by ordinary 120-V electrical outlets, which also provide the voltage signal used to compute the phasor value. Internet access is required in order for FDRs to transmit their measurement results via ethernet. Measurement data from FDRs are managed in the data center by multilayer agents. The top layer is the FNET data concentrator, whose primary functions is to receive data from the FDRs, create GPS time-aligned records, share data with the real-time application agent as soon as the records are made, and forward the data records to the data storage agent and subscribed clients. The real-time application agent and data storage agent are in the second layer of the data

center hierarchy, which provide the framework for acquiring the reference data for the audio authentication process. The third layer is the non-real-time application agent [7].

3.2.1 Frequency Disturbance Recorders (FDRs)

An FDR is essentially a digital signal processor containing the phasor value estimation algorithms, coupled with GPS time synchronization and Ethernet communications capability [9]. These devices are now in their second generation and Figure 7 illustrates an example of the current FDR unit. The unit takes a regular power outlet's 120-V sinusoidal voltage signal, and this voltage is stepped down to 10 V by the internal transformer of the FDR. High-frequency noises are then blocked by the low-pass filter. The ADC periodically samples the 10-V signal according to the GPS clock-regulated oscillator pulses. (Oscillator pulses are synchronized with a one pulse per second (PPS) signal, provided by the GPS module.) Phasor values such as voltage magnitude, phase angle, and frequency are calculated by the digital signal processing unit and then transmitted over the Internet by the device server. Figure 8 illustrates the hardware architecture of the frequency disturbance recorder.

3.2.2 Information Management System

The FNET information management system is a multilayer data management and utilization system operated by several dedicated server computers. The most powerful component of the data center is the FNET data concentrator. All active FDRs send data to specific TCP/IP ports, which are then stored into a predefined static memory section of the data concentrator. The saved data are in the format of records, each contains all FDRs' measurements

Figure 7(a): Frequency Disturbance Recorder (exterior) [9]

Figure 7(b): Frequency Disturbance Recorder (interior) [9]

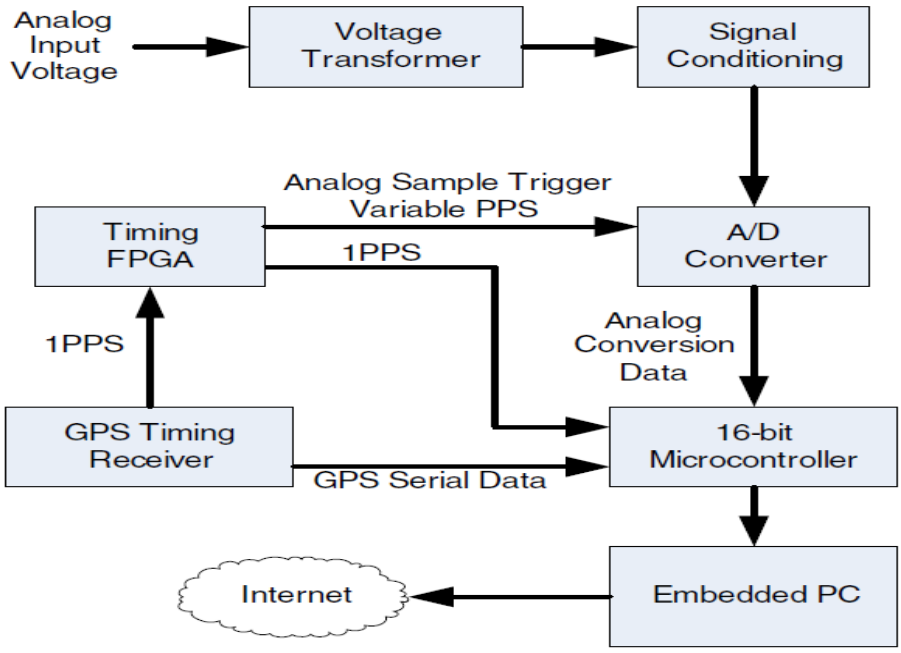


Figure 8: Hardware Architecture of FDR [9]

with the same GPS time stamp. Each record is streamed to the real-time application server at the time it is created. Whenever the size of the records reaches the assigned limit within the memory cache, all the records are written into a Microsoft Access database and saved in the data concentrator. The data collection and record padding is not interrupted by the saving procedure because the memory cache is designed as a stack, only the oldest portion of the data is saved. Each midnight, an Access database file is created with all the FDR data from the previous day and saved to the data storage server. The data concentrator possesses the ability to flag bad data, report missing data, and alarm abnormal interruptions to ensure accurate FNET operations and complete data recordings.

The second layer of the data center hierarchy is composed of the real-time application server and the data storage server. The real-time application server is implemented with the real-time modules including the frequency interface, frequency event trigger, event location, and oscillation trigger. Like the data concentrator, there is a section of memory cache that collects the real-time FDR records. But only the data that are used by any of the applications are saved. The function of the data storage server is to simply receive and store all the historical FNET data. The third layer is a non-real-time analysis server. Applications implemented on this server are operate on the saved data from the data storage server or the real-time application server such as event visualization, oscillation modal analysis, and Web service [7].

3.2.3 FNET-OpenPDC Integration

The Power IT Lab at the University of Tennessee is currently testing the practicability of using OpenPDC as the phasor data concentrator for FNET. OpenPDC is widely becoming the accepted phasor data management platform in the industry, and the adaptation of the technology by FNET could garner great benefits moving into the future. Apart from the FNET phasor data protocol, OpenPDC also supports IEEE C37.118-2005 and 2011, IEEE1344, BPA PDCStream, Macrodyne, SEL Fast Message, etc. OpenPDC is an open source PDC, which was developed and made available to public by the Tennessee Valley Authority (TVA), in October 2009. It collects processes and manages data from PMUs or other PDCs. OpenPDC architecturally consists of three layers: the Input, Action and Output Layers. Each layer performs a specific set of functions. Figure 9 shows the adapter based architecture of the OpenPDC.

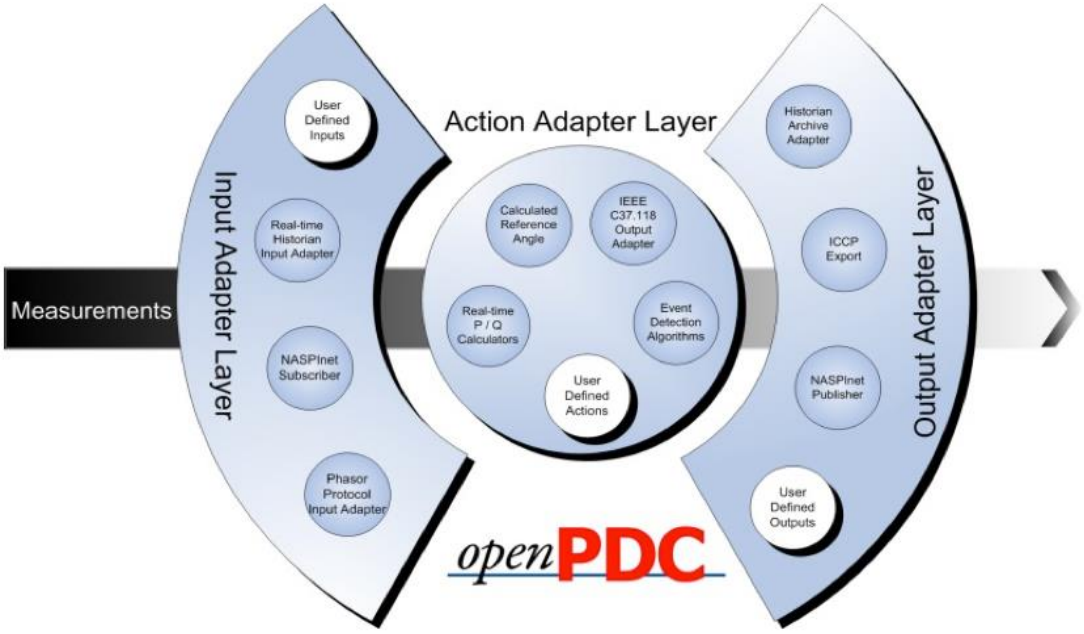


Figure 9: OpenPDC adapter based architecture [10]

The Input Adapter Layer reads streaming data from the measurement devices that may use different protocols. In fact, it allows the parsing of protocol and provides a generic data format. After assigning an ID to the measurement, it will transfer them to the Action Layer. The Action Adapter Layer deals with concentration and processing of the input measurements. One of the important functions of this layer is the phasor time alignment adapter. In this function the measurements received from the Input Layer are sorted by their associated GPS time-stamps and time aligned before transferring them to the next layer. The Action Adapter Layer also provides two more essential functions, which are real-time measurement calculation and real-time event detection. The real-time measurement calculation function uses the measured

parameters of the PMUs to calculate new parameters that are not directly collected for instance, active and reactive power [10].

It is evident that all the required parameters for the calculation of new parameters must be measured and made available by the PMU. The real-time event detection function monitors the incoming measurements and ensures that they are within the specified limits. If the measurements exceed their specified limits, an event will be triggered to signal a problem in system operation. Finally, the Output Adapter Layer of the OpenPDC receives all measured and calculated parameters at the end and queues up data and forwards them to either a data historian system to archive them for future off-line analysis, or to any other defined client such as a real-time application. The Output Adapter can also re-encapsulate the data in several protocols. Example of OpenPDC output types are IEEE C37.118 concentrator output stream, Inter Control Center Protocol (ICCP), Comma Separated Values (CSV) File Export, and Historian Archiving output [10].

One of the primary benefits of integrating FNET into OpenPDC is the ability to bring together the several layers of the data center and not have to worry about separate servers to perform the different functions. Figure 10 shows the FNET-OpenPDC integrated framework. The data concentration and some of the applications will all be performed under the umbrella of the OpenPDC phasor data management system. Furthermore, OpenPDC has various features that make the monitoring of the power systems more convenient. One of them is the ability to replay data streams from historical data. In this mode, the stored data from the Historian will be retransmitted to the OpenPDC software and shows the data as if it is coming from a PMU. Another feature of OpenPDC is that each of the adapter layers functionalities can be improved or

extended by users or developers. For example, a new Input Adapter can be developed to support new protocols, or a new Action Adapter can be defined to provide a particular function, such as calculation of new parameter or detection of system event [11].

To test the feasibility of using OpenPDC to host FNET data and applications, a test system was setup. The system entailed using a single FDR unit housed at the Min Kao building on the University of Tennessee campus to stream data to OpenPDC, acting as a local phasor data concentrator. The FDR unit had to be configured to send data directly to the IP address of my desktop hosting the OpenPDC software. To confirm whether the data stream was successfully being received at my desktop, I used PMU Connection Tester to view the phasor data. PMU Connection Tester is a software tool that verifies whether the data streams from known phasor

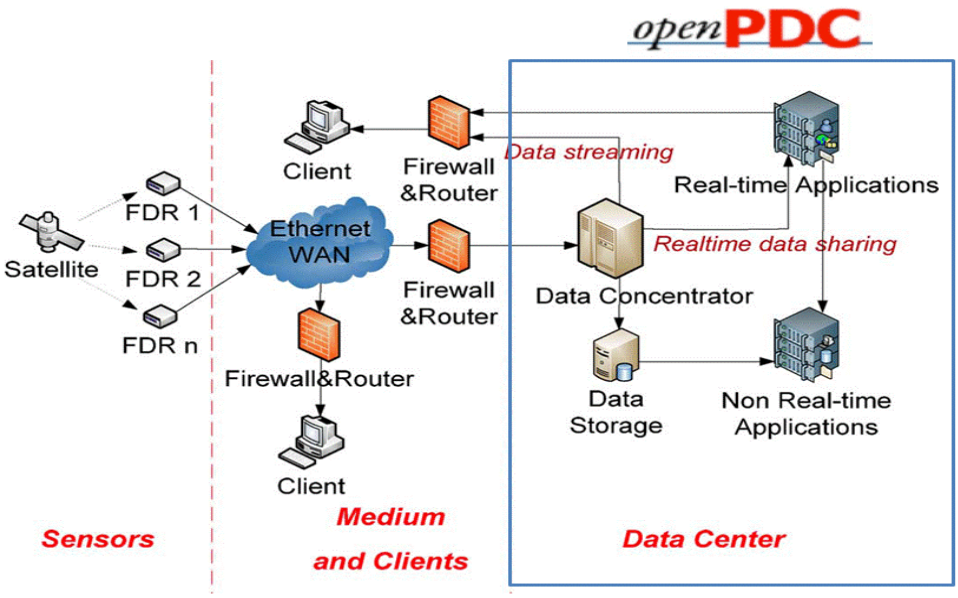


Figure 10: FNET-OpenPDC Integrated Framework

measurement devices are being received. It supports all of the major phasor data protocols including FNET. The software provides the time series phasor information related to currents and voltages along with associated frequency information. The data is available in graphical displays and also in exported "csv file" formats for use in other applications. Figure 11 shows the frequency data within the FNET data stream being received by the PMU Connection Tester.

Although PMU Connection tester has the capability to view phasor data streams, it cannot archive the data or develop applications and this is where the OpenPDC Manager comes in. OpenPDC Manager is a silverlight based application which allows users to remotely configure and manage multiple deployed instances of the OpenPDC from a single simple to use

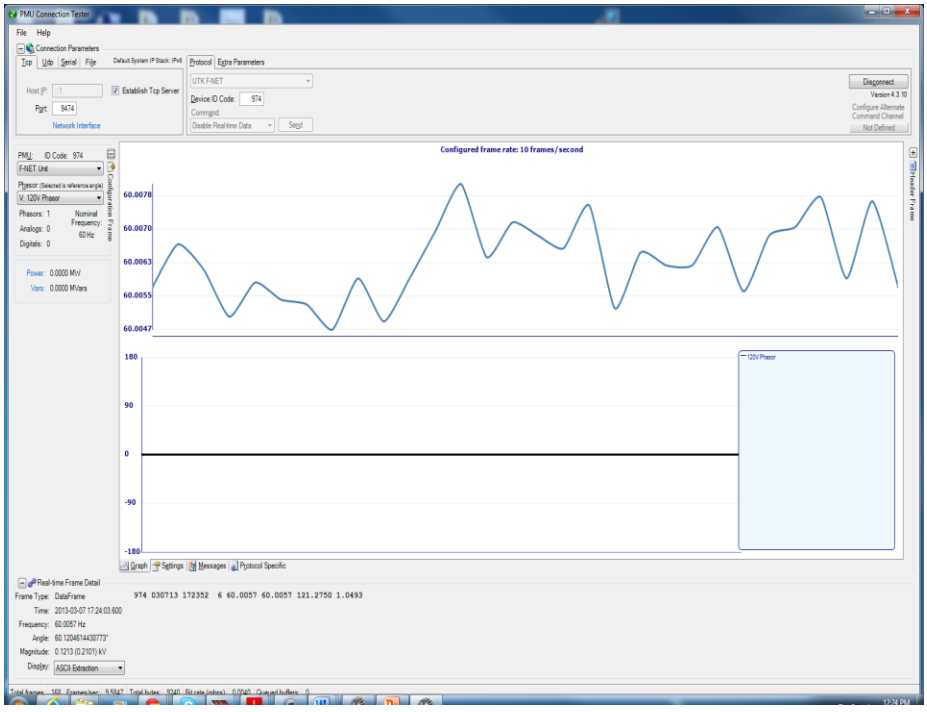


Figure 11: PMU Connection Tester showing FNET data

web application. This application is used to manage and configure system devices, custom system input, action and output adapters as well as to set operational parameters. It also allows for automated device configuration using an XML configuration file captured using PMU connection tester [10]. This configuration file was then used to configure the manager to receive the FNET data. Figure 12 shows the OpenPDC manager receiving and displaying FNET voltage magnitude and frequency data for the FDR unit. Although the data was being received by the manager, a Historian had to be created in order to archive the data for future use. The default OpenPDC Historian was used to archive the received FNET data. The historian is a system for collecting, archiving, retrieving and displaying time-series event data in real-time from multiple

Figure 12: OpenPDC Manager showing FNET data

data sources. Historian Data Viewer is a tool that allows you to retrieve and plot phasor data archived within the OpenPDC historian, by specifying the time frame and the data you wish to view. Figure 13 shows voltage magnitude, phase angle and frequency of the FDR unit, as viewed through the historian data viewer.

To ensure the accuracy of the OpenPDC system in managing the FNET data, a comparison between the data received by OpenPDC and the data received by the legacy FNET system was made. Figure 14 shows a 10-minute comparison of frequency data from two separate FDR units within the Eastern Interconnection, one streaming data to OpenPDC and the other to FNET. As is evident the general trend is the same between the two systems meaning the OpenPDC system does in fact receive accurate FNET data. There was however a time delay in the FNET data that needed to be corrected. The 11.5 second delay was due to the buffering of the FDR data before being archived in the FNET database, which is where the data for this comparison was derived.

3.3 ENF Analysis Method

Before the authenticity of audio recordings could be verified, the ENF embedded in the recordings must first be extracted. Figure 15 shows the spectrogram of a voice recording, where the 60 Hz component corresponding to the ENF can clearly be seen. After this ENF is extracted, it can then be compared to the frequency in the FNET database, and the authenticity of the recording determined. Currently there are three main methods used to extract ENF from a digital audio recording: 1) time/frequency domain analysis-based on the spectrogram, 2) frequency

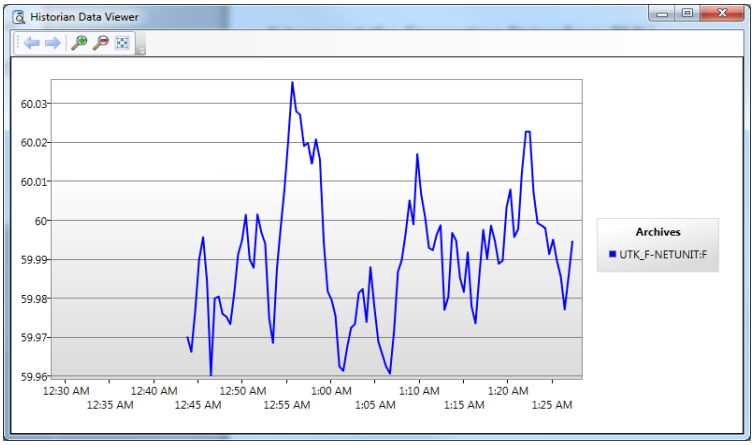


Figure 13(a): Historian Data Viewer showing archived phasor data (frequency)

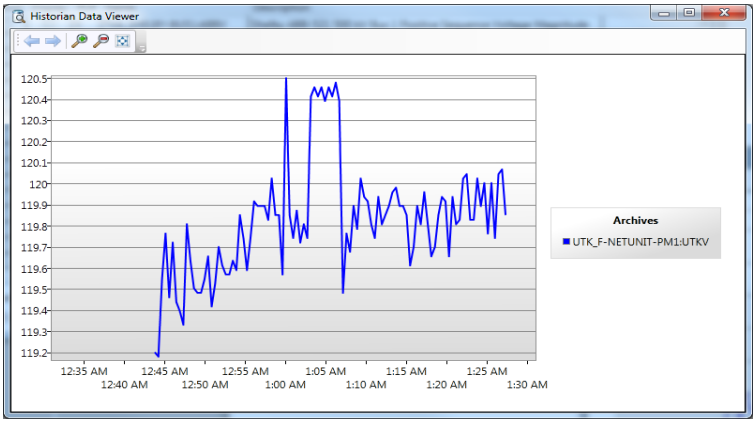


Figure 13(b): Historian Data Viewer showing archived phasor data (voltage magnitude)

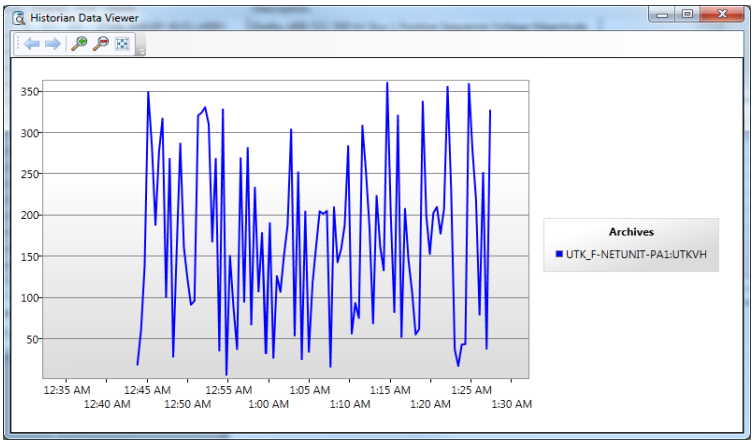


Figure 13(c): Historian Data Viewer showing archived phasor data (voltage angle)

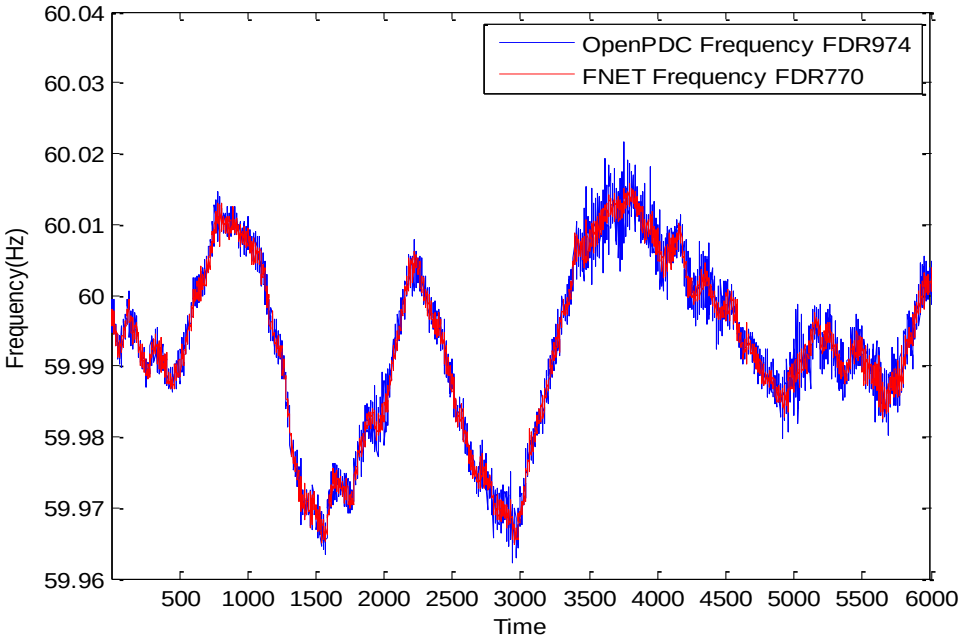


Figure 14: 10 minute comparison of FDR data from FNET and OpenPDC

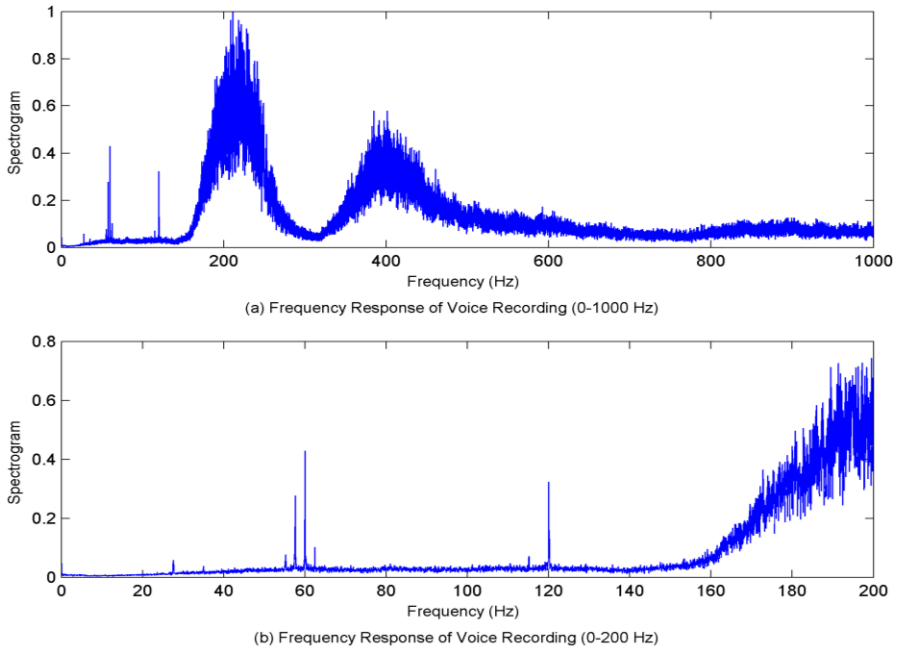


Figure 15: Frequency response of voice recording showing ENF

domain analysis-based on selecting the maximal magnitude of a series of power spectra calculated from consecutive time segments of data, and 3) time domain analysis-based on zero-crossing measurements of a band-pass filtered signal [6]. However the method used for the purposes of this project is a frequency domain based approach. Before the ENF extraction scheme is outlined a couple concepts must first be introduced.

Short-time Fourier Transform (STFT)

The definition of STFT for a given signal $x(t) \in L^2(R)$ can be expressed as:

$$STFT_x(t,\Omega) = \int x(\tau) g *_{t,\Omega} (\tau) d\tau = \int x(\tau) g * (\tau - 1) e^{-j\Omega\tau} d\tau \quad (1)$$

where

$$g_{t,\Omega}(\tau) = g(\tau - t) e^{j\Omega t}$$

$$\|g(\tau)\| = \left\|g_{t,\Omega}(\tau)\right\| = 1$$

The idea behind of (1) is to truncate the signal $x(\tau)$ using a window function $g(\tau)$ in the time domain and to calculate the Fourier transform of this truncated signal. Note the time index in $x(t)$ obtained by continuously moving the time parameter t, namely, the central location of the window function. All these Fourier transform results consist of the $STFT_x(t,\Omega)$. In order to realize the STFT on a computer, the signal must be discrete and of finite length. When both time and frequency are discretized, (1) becomes a standard discrete Fourier transform (DFT). Figure 16 shows the principal of realization of STFT, where the signal is separated into N frames. Hop size and window size are two important parameters used to determine the shift and length of a selected window function. It is obvious that the length of a truncated signal equals to the length of window function.

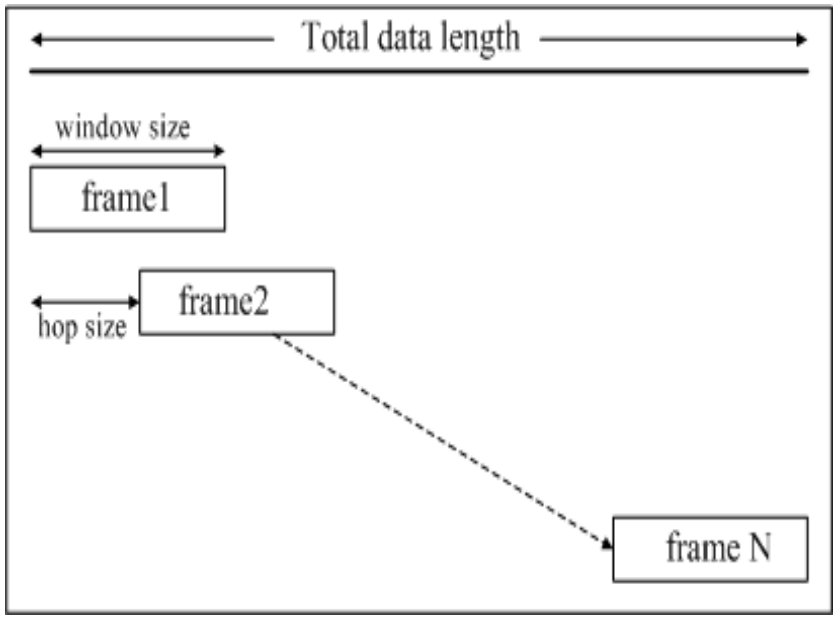


Figure 16: Principle of STFT [12]

Spline Interpolation

From each frequency spectrum, frequency corresponding to the maximal magnitude is selected. Since it is unlikely that this corresponding frequency coincides exactly with a DFT frequency bin, spline interpolation is introduced to rectify this frequency to obtain the final frequency. It should be pointed out that for a DFT of each frame, only the frequencies within the range of [59.5, 60.5] are of interest. Maximum magnitude and its corresponding frequency and the final rectified frequency are based on this window of interest.

The ENF extraction method can then be implemented as follows [12]:

1. The signal is decimated and band-pass filtered to select frequencies [59.5, 60.5].
2. The filtered signal is separated into *N* frames according to window size and hop size.

3. For each windowed frame, DFT is calculated and the peak frequency according to the magnitude of frequency spectrum of interest is found.
4. This peak frequency is rectified using the spline interpolation.
5. Repeat this process for all the frames of the recording, and the result will be the extracted ENF.

After the extraction is complete, the ENF sequence is matched against the FNET database. Figure 17 shows a comparison between an extracted ENF and FNET data taken at the same point in time. The accuracy of the extraction method becomes apparent. Mean square error (MSE) is used to measure the error between the ENF sequence and the reference frequency sequence of the same time length. The MSE is computed as follows:

$$\epsilon = \log(\frac{1}{M}\sum_{i=1}^{M}(ENF(i) - ref(i))^2)$$

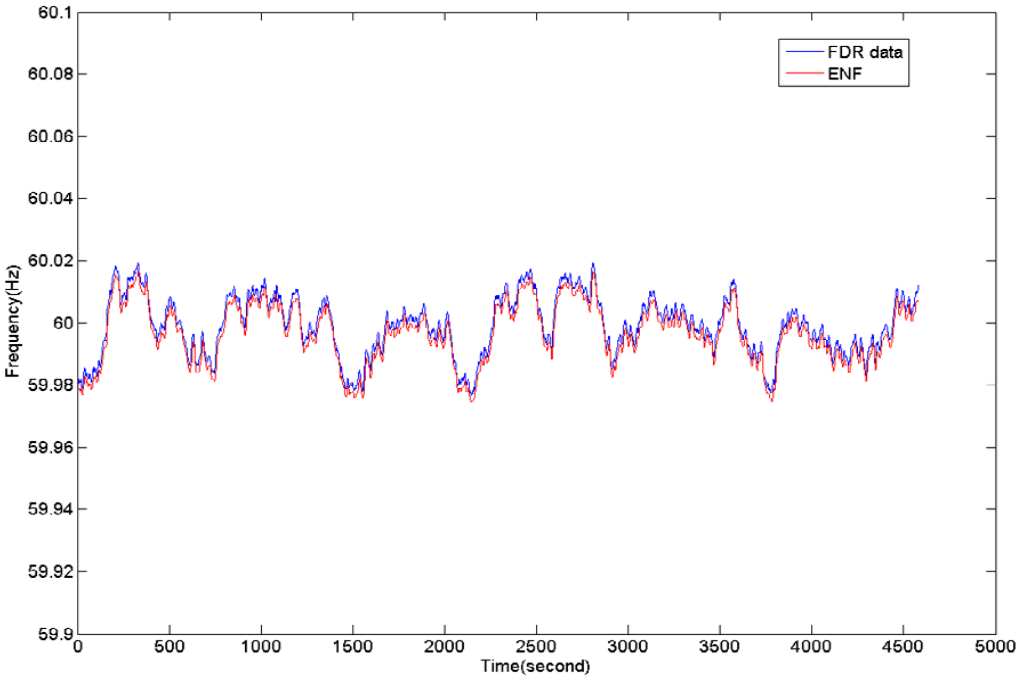


Figure 17: Extracted ENF and FDR Data Comparison [10]

A perfect match will naturally have the minimum MSE, so this value could be used to determine the time of creation of a recording. If the time of recording is known and the calculated MSE does not correspond to the minimum, this can be an indicator of an inauthentic recording. Figure 18 shows the MSE of an extracted ENF signal, and the point that corresponds to minimum MSE can be seen.

3.4 Graphical User Interface

The coding and functions to implement the extraction procedure was developed in MATLAB. The code structure was such that there were first, second and third-level functions that needed to be executed before the ENF would ultimately be extracted. Figure 19 shows the

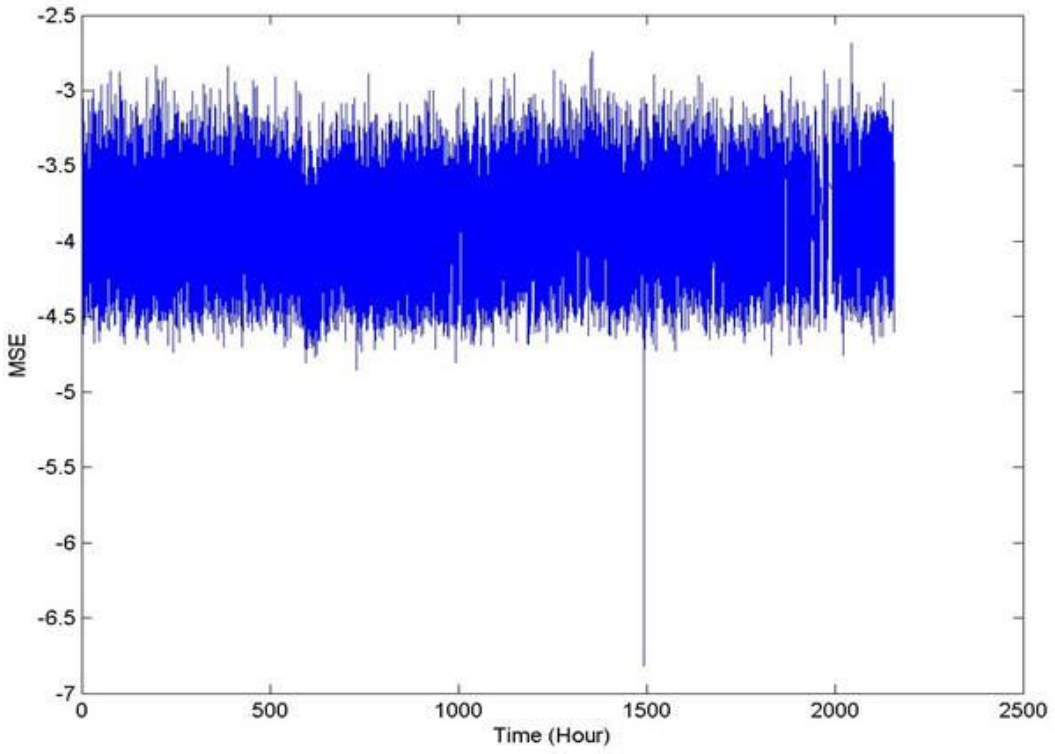


Figure 18: MSE for extracted ENF from recording [12]

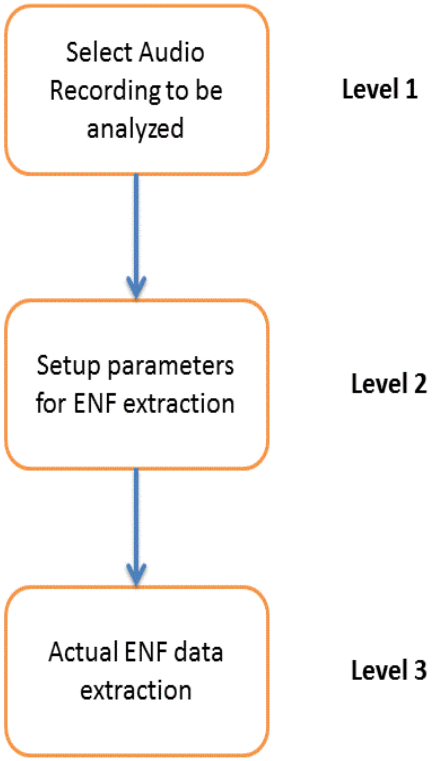


Figure 19: Original code structure for ENF Analysis

structure of the ENF extraction algorithm, and the order in which the different functions performed their role in the process. At Level 1 the function had the responsibility of selecting the recording to be analyzed. This particular function also had the responsibility of performing different analyses related to the audio recording such as viewing its power spectral density and spectrogram. At Level 2 parameters related to the ENF extraction procedure are selected, such as window size and the length of recording to be analyzed. At Level 3 the actual ENF extraction procedure using the STFT is performed. After beginning to use this code structure in ENF

analysis, it quickly became quite cumbersome to make code changes as it often led to debugging issues, and the process of parameter selection was not very user friendly.

This prompted me to first develop a single MATLAB script to implement the extraction algorithm without the several level of functions. This led to a reduction in the debugging time after code changes, and the overall time it took to analyze different recordings. Figure 20 shows the structure of the newly developed ENF extraction procedure. Moreover, I develop a Graphical User Interface based on the new structured that would facilitate the easy editing of parameters

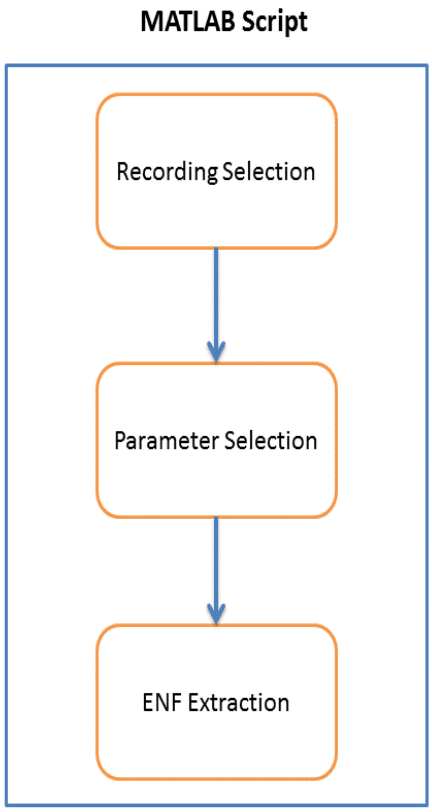


Figure 20: New code structure for ENF Analysis

and further automate the ENF extraction procedure. Figure 21 shows the ENF extraction GUI default window. The application was developed using MATLAB GUIDE (GUI development environment). Appendix A shows the code used and an instruction manual on how to utilize the GUI. It contains several push buttons, data input windows and a plotting window that allows the user to straightforwardly select the recording and parameters. The user can then perform the ENF extraction procedure, then plot the results or save the data for later use. The user can also plot the .wav file selected for analysis directly, or plot the phase angle that comes as a bi-product of the ENF extraction algorithm. The following is a description of the different inputs and functions that make up the GUI:

- **Browse for File** – locate the audio recording they would like to analyze.
- **Enter Window Size** – window size to be used in STFT procedure
- **Enter Starting Point –** starting time in seconds of recording to be analyzed

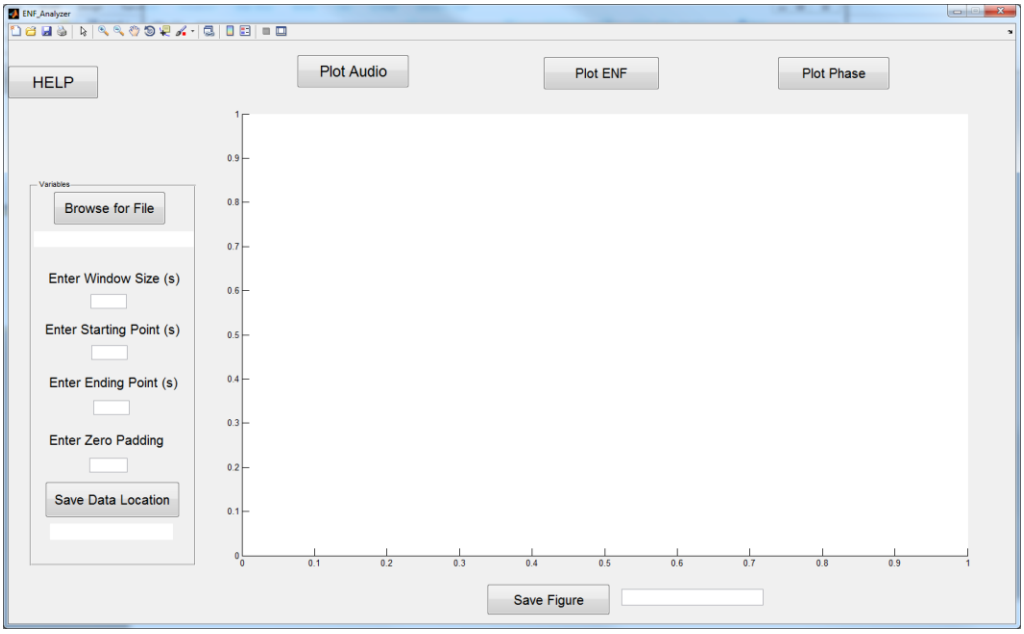


Figure 21: ENF extraction GUI

- **Enter Ending Point –** end time in seconds of recording to be analyzed
- **Enter Zero Padding** – zero padding factor to be used in STFT procedure
- **Save Data Location** – designate location where extracted ENF and phase data will be stored.
- **Plot Audio** – plot the audio recording in the form of wave file
- **Plot ENF** – plot the extracted ENF signal
- **Plot Phase** – plot the extracted phase signal
- **Save Figure** – save the figure that is currently plotted on axes
- **Help** – opens word file that contains instruction on how to operate the GUI, and other related information.

Figure 22 shows the results of a sample audio recording that was analyzed for ENF using the GUI. The figure shows a plot of the raw audio data, the extracted ENF, and the extracted phase angle. The ENF and phase data extracted via the GUI could then be used in the audio authentication procedure, through comparison with the reference FDR data. The ultimate goal of this project is to one day have law enforcement officials utilize the ENF criterion for forensic authentication of digital recordings. Because many of the relevant authorities may not have extensive experience in handling complex code, it will be ideal for an application similar to the one above to be developed, which will allow them to apply the procedure without necessarily getting perplexed by the complex algorithms. This GUI is a first step in that direction, and as the theory continues to improve and the algorithm perfected, this GUI will continue to be enhanced until it eventually reaches production grade for law enforcement officials.

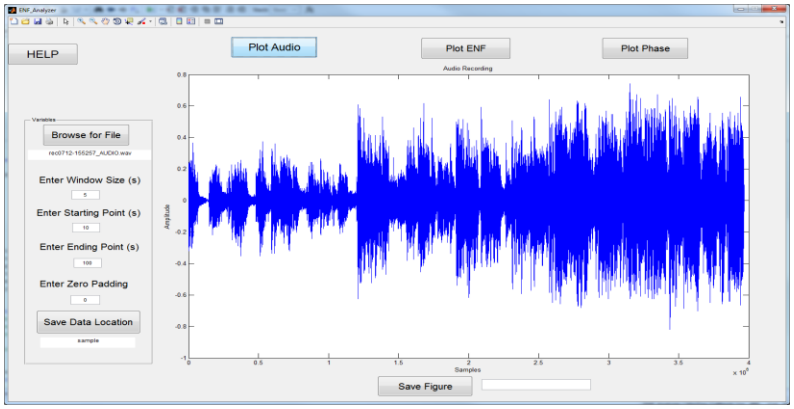


Figure 22(a): Sample results of ENF analysis using GUI (audio data)

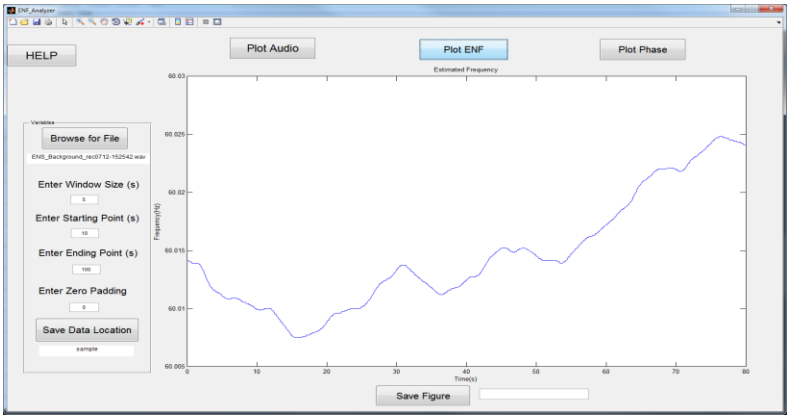


Figure 22(b): Sample results of ENF analysis using GUI (extracted frequency)

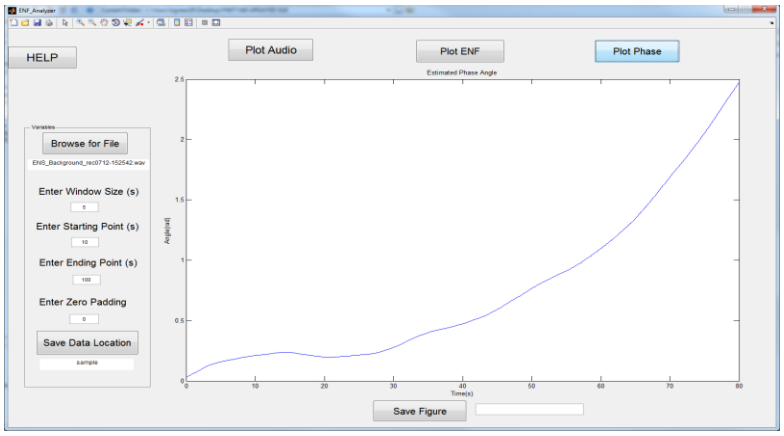


Figure 22(c): Sample results of ENF analysis using GUI (extracted phase)

3.5 Tampering Detection

As mentioned above the tampering of recordings can be detected through comparison of extracted ENF and reference ENF from FNET database using the mean square error (MSE) criterion. If the time of recording is known and the calculated MSE does not correspond to the minimum, this can be an indicator of an inauthentic recording. This approach however, is only useful in situations where the time of recording is known, as this is the only way the retrieval of the required reference data from the FNET database would be possible. Therefore in situations where the time of recording is not known, a different approach had to be proposed for tampering detection.

The tampering of an audio recording is described as situations where the recording has been modified from its original state. These modifications can be classed into two categories: (1) cases where a segment of the recording which was originally present has been removed (deletion) and (2) cases where a segment of the recording which was not originally present has been attached (insertion). These modifications of the recordings consequently lead to discrepancies in the embedded ENF signals. And as a result, both the extracted ENF and phase have noticeable variations at the tampering points. Figure 23 shows a case of a 5-minute deletion for a 30-minute long audio recording. It is evident that both the extracted ENF and phase oscillate for a short period of time before returning to normal. These deviations at the tampering points can therefore be used to develop an automatic tampering detection procedure, where the breaking of set thresholds will be used to indicate possible tampering.

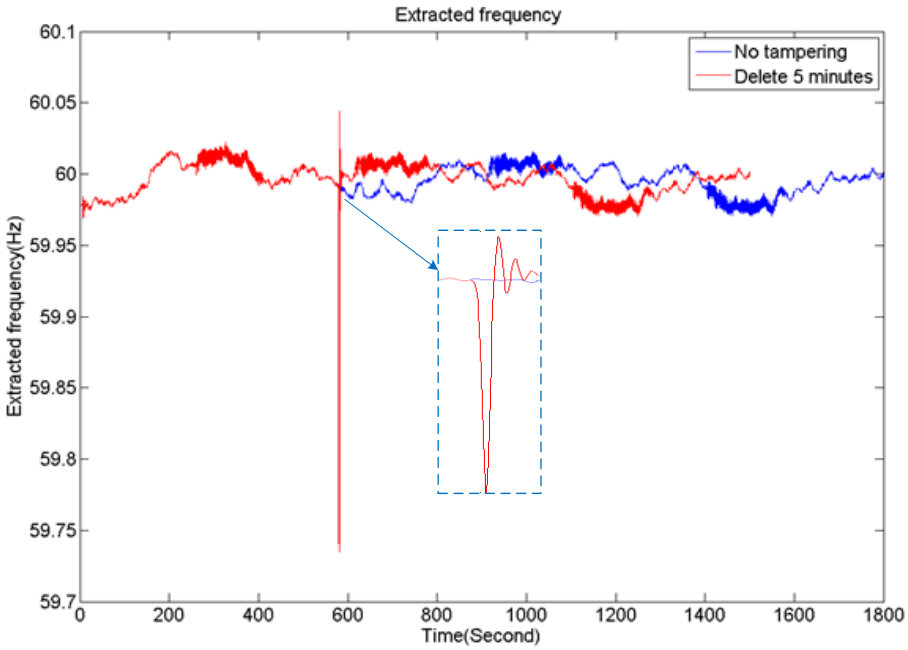


Figure 23(a): Variation of ENF at tampering point

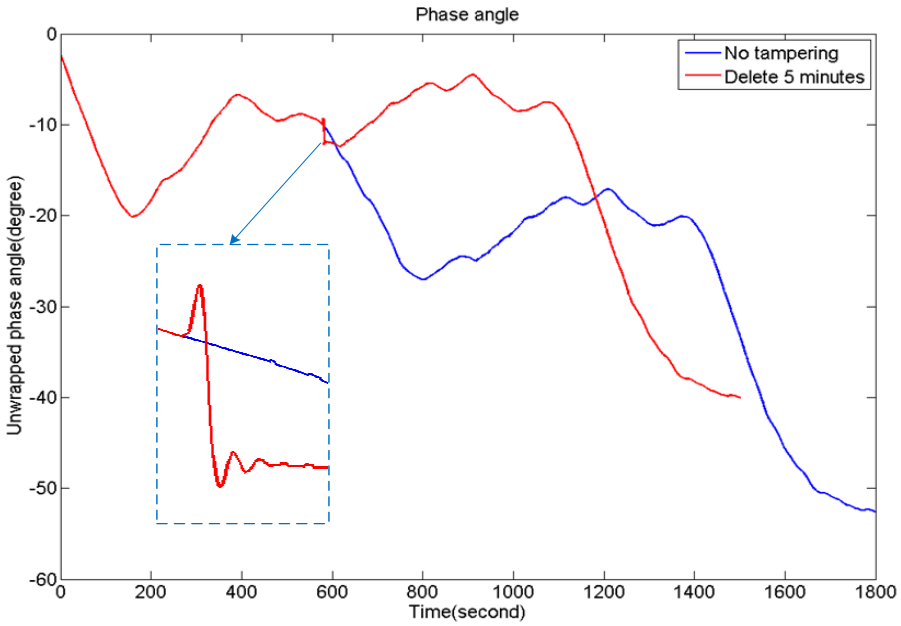


Figure 23(b): Variation of phase angle at tampering point

Suppose we define a hypothesis group $\{H_o, H_t\}$ where H_o and H_t represent the hypothesis for an analyzed case being original and tampered respectively; then a decision can be made according to:

$$H_o := Index < \gamma$$
$$H_t := Index \geq \gamma$$

where *Index* is a feature or measure set defined to describe the variations of frequency and phase angle and γ stands for a threshold set.

Given the *Index*, the threshold γ can be determined by trial-and-error based on a known sample set. This threshold should have the maximal ability to discriminate original and tampered cases in the sample set. Once γ is determined, the hypothesis test is carried out on the remaining cases. We can predict that the test results are of three types: a) *successful detection*, which means a tampered case is successfully detected; b) *false detection*, which means an original case is labeled wrongly as a tampered case; c) *missing detection*, which means a case is classified as an original case whereas it has indeed been tampered. Let P_{sd}, P_{fd} and P_{md} be the probabilities of the above mentioned three types respectively and are expressed as:

$$P_{sd} = P(Index \geq \gamma | H_t)$$

$$P_{fd} = P(Index \geq \gamma | H_o)$$

$$P_{md} = P(Index < \gamma | H_t)$$

Three probabilities above are easy to obtain since the numbers of three types and the total number of tested cases are known. According to them we may evaluate the performance of *Index*

and γ in the hypothesis test, and choose the effective feature and threshold. This can be a feasible way for automatic tampering detection, though it may be calculation intensive if the tested cases are large enough.

Chapter 4

Three-phase Instrument Transformer Calibration using Phasor Measurements

4.1 Project Overview

The current and voltage values at which the electrical gird operates are too high to directly integrate with intelligent electronic devices like relays and PMUs. As a result these primary values must be stepped down using instrument transformers. Instrument transformers exist in two forms: Voltage Trasnformers (VTs) and Current Transformers (CTs). The primary circuit of a VT is connected in parallel at the monitoring point, while the CT is connected in series. For VTs the secondary circuit is typically stepped down to 120V and CTs are typically stepped down to 5A. The ratio by which the primary circuit signals are stepped down to the secondary circuit signals is called the conversion ratio. The nominal conversion ratios as specified on device documentation are usually different from the actual conversion ratio, as the prevailing burden, age and environmental conditions usually have an effect on the instruments. This deviation from the nominal values is defined as the Ratio Correction Factor (RCF) of the instrument transformer. RCF is a complex number, and may be expressed as magnitude correction factor (MCF) and a phase angle correction factor (PACF). In an ideal transformer, the MCF equals one and the PACF equals zero.

It is possible for instrument transformers to be accurately calibrated to compensate for the errors created by RCF. Many methods of instrument transformer calibration have been proposed, however not many are geared towards three phase calibration. On-site calibration is the most straightforward method, but unfortunately it is too labor intensive and requires the transducers be out of service during calibration. One method to remotely calibrate transducers uses telemetered transducer measurements from system substations to estimate the calibration parameters [15]. The RCFs are basically determined by requiring Kirchoff's current and voltage law be satisfied. Another method was to use different load scans based on PMU measurements [14]. However, neither of these methods took into consideration three-phase transducer calibration, which is important because power systems operate under unbalanced conditions, hence the need for accurate measurements for three-phase studies. In this chapter I will propose a method of three-phase instrument transformer calibrations using phasor measurements, where calibration refers to the accurate assessment of correction factors MCF and PACF.

Furthermore for the successful implementation of the instrument transformer calibration scheme, it is necessary to have at least one set of three phase voltage transformers (to be used as reference) with a known and accurate RCF. A Potential Transformer (PT) can be used to serve this purpose. However for the Dominion 500 kV network, which is the system being used to test the application, the location of the accurate PT measurement is at a LV bus while the PMU measurement is taken at a HV bus. Therefore, I had to translate the accurate LV measurement to the HV, using the impedance network at the substation. Part of this impedance network entails an LTC-controlled transformer, and in order to calculate the most accurate impedance network, the

LTC positions must be known. This chapter also outlines the technique I used to capture the LTC positions with time stamps using the PMU.

4.2 System Model

Because the calibration scheme involves three phases, a three-phase power system model is used. The scheme is implemented on a two bus system, but a discussion follows on how it can be extended to a larger network model. Consider the two bus model shown in Figure 24. A three phase π-type transmission line model connects buses P and Q. The parameters of the line are as follows [15]:

$$Z_{pq}^{abc} = \begin{bmatrix} Z_{pq}^{a} & Z_{pq}^{ab} & Z_{pq}^{ac} \\ Z_{pq}^{ab} & Z_{pq}^{b} & Z_{pq}^{bc} \\ Z_{pq}^{ac} & Z_{pq}^{bc} & Z_{pq}^{c} \end{bmatrix} \Rightarrow \quad Y_{pq}^{abc} = (Z_{pq}^{abc})^{-1} = \begin{bmatrix} Y_{pq}^{a} & Y_{pq}^{ab} & Y_{pq}^{ac} \\ Y_{pq}^{ab} & Y_{pq}^{b} & Y_{pq}^{bc} \\ Y_{pq}^{ac} & Y_{pq}^{bc} & Y_{pq}^{c} \end{bmatrix} \quad (1)$$

$$B_{p0}^{abc} = j \begin{bmatrix} B_{p0}^{a} + B_{p0}^{ab} + B_{p0}^{ac} & -B_{p0}^{ab} & -B_{p0}^{ac} \\ -B_{p0}^{ab} & B_{p0}^{b} + B_{p0}^{ab} + B_{p0}^{bc} & -B_{p0}^{bc} \\ -B_{p0}^{ac} & -B_{p0}^{bc} & B_{p0}^{c} + B_{p0}^{ac} + B_{p0}^{bc} \end{bmatrix} \quad (2)$$

$$B_{q0}^{abc} = j \begin{bmatrix} B_{q0}^{a} + B_{q0}^{ab} + B_{q0}^{ac} & -B_{q0}^{ab} & -B_{q0}^{ac} \\ -B_{q0}^{ab} & B_{q0}^{b} + B_{q0}^{ab} + B_{q0}^{bc} & -B_{q0}^{bc} \\ -B_{q0}^{ac} & -B_{q0}^{bc} & B_{q0}^{c} + B_{q0}^{ac} + B_{q0}^{bc} \end{bmatrix} \quad (3)$$

Where,

Z - Line series impedance matrix

Y – Line series admittance matrix

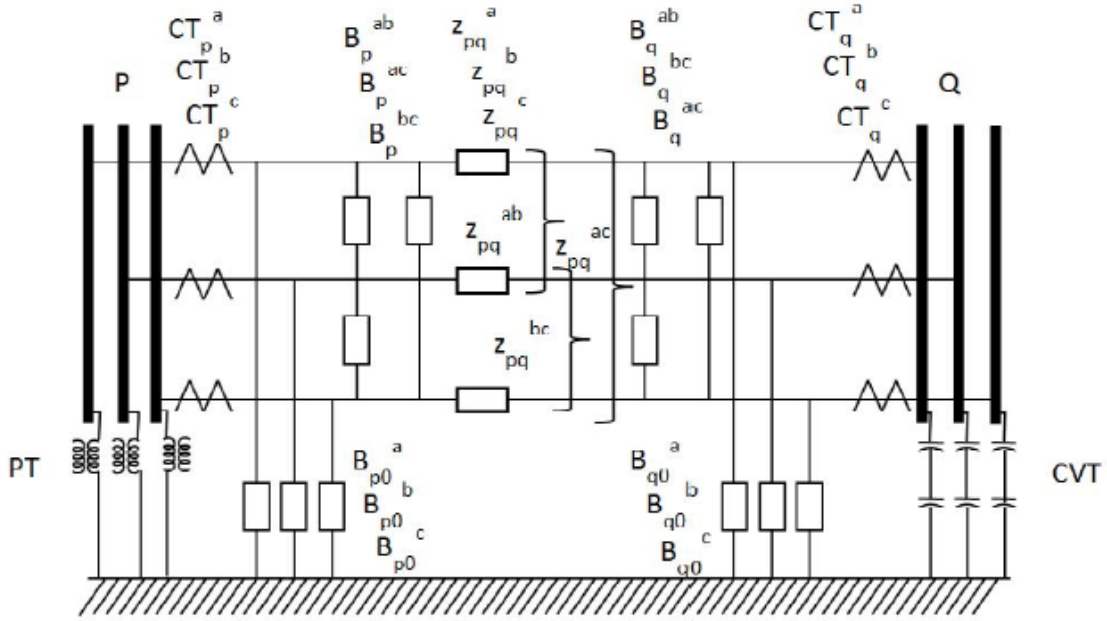


Figure 24: Three phase model of two bus system for transducer calibration

B – Line shunt susceptance matrix

p,q – subscripts referring to bus index

a,b,c – subscripts referring to phase index

0 – refers to ground

Four instrument transformers are also included in the model. Three phase voltage transformers (PT and CVT) are connected to buses P and Q respectively, and three phase current transformers (CTs) are connected in series at each terminal of the transmission line. It is also assumed that a PMU is measuring each of the secondary voltages and currents coming from the instrument transformers [15]. The current in the transmission line is designated as I_{pq}^{abc} and I_{qp}^{abc} at the two buses, and are measured by PMUs connected to the CTs. Similarly, voltages E_p^{abc} and

E_q^{abc} are also assumed to be measured by PMUs through the voltage transformers. Hence, the relationship between the system currents and voltages can be described by the following equations:

$$Y_{pq}^{abc} E_{q0}^{abc} = Y_{pq}^{abc} E_{p0}^{abc} + B_{p0}^{abc} E_{p0}^{abc} - I_{pq0}^{abc}, \quad (4)$$

$$Y_{pq}^{abc} E_{p0}^{abc} = Y_{pq}^{abc} E_{q0}^{abc} + B_{q0}^{abc} E_{q0}^{abc} - I_{qp0}^{abc}$$

4.3 Methodology

Let us define an equation relating the Ratio Correction Factors of the instrument transformers to the PMU measurements by [15]:

$$diag(K_x^{abc}) X_m^{abc} = X^{abc} \quad (5)$$

Where,

X^{abc} – Vector of three phase phasors (voltages or currents) in per unit at the input of instrument transformers.

X_m^{abc} – Vector of three phase phasors in per unit measured by PMUs.
K_x^{abc} – Vector of overall three phase transducer Ratio Correction Factor

It is important to note that the K factors used in (5) are the inverse of the traditional Ratio Correction Factors, and defined as input signal divided by output signal. It is defined this way in order to simplify the equations used in the calculation of K [15]. In the case of the two bus system, the respective current and voltage PMU measurements as related to the RCFs can be defined as:

$$diag(K_e^{abc})E_m^{abc} = E_0^{abc}, \qquad (6)$$

$$diag(K_i^{abc})I_m^{abc} = I_0^{abc}$$

Therefore, replacing the true values of currents and voltages in equation (4) can be replaced by PMU measured values, by substituting equation (6) into equation (4):

$$Y_{pq}^{abc}diag(K_{eq}^{abc})E_{qm}^{abc} = Y_{pq}^{abc}diag(K_{ep}^{abc})E_{pm}^{abc} + B_{p0}^{abc}diag(K_{ep}^{abc})E_{pm}^{abc} - diag(K_{ipq}^{abc})I_{pqm}^{abc}$$

$$Y_{pq}^{abc}diag(K_{ep}^{abc})E_{pm}^{abc} = Y_{pq}^{abc}diag(K_{eq}^{abc})E_{qm}^{abc} + B_{q0}^{abc}diag(K_{eq}^{abc})E_{qm}^{abc} - diag(K_{iqp}^{abc})I_{qpm}^{abc}$$

(7)

As mentioned before in order for the calibration scheme to work, it is necessary to have a measurement with a known and accurate RCF. It is assumed that the voltage transformer at bus P, is an ideal PT and therefore has an RCF of 1.0 + j0. Using this fact, equation (7) can be re-written as:

$$\begin{bmatrix} Y_{pq}^{abc}diag(E_{qm}^{abc}) & diag(I_{pqm}^{abc}) & 0 \\ -Y_{pq}^{abc}diag(E_{qm}^{abc}) - B_{q0}^{abc}diag(E_{qm}^{abc}) & 0 & I_{pqm}^{abc} \end{bmatrix} \text{X} \qquad (8)$$

$$\begin{bmatrix} K_{eq}^{abc} \\ K_{ipq}^{abc} \\ K_{iqp}^{abc} \end{bmatrix} = \begin{bmatrix} -Y_{pq}^{abc} - B_{p0}^{abc} \\ Y_{pq}^{abc} \end{bmatrix} diag(E_{pm}^{abc})$$

Equation (8) represents 6 algebraic equations with 9 unknown ratio correction factors: 3 for voltages at bus Q (K_{eq}^{abc}), and 3 each for current transformers at buses P and Q (K_{ipq}^{abc} and K_{iqp}^{abc}). The remaining equations can be acquired by making measurements on the network for different loading conditions [15]. Therefore, using subscripts 1 and 2 to denote the two loading conditions, the set of measurements can be described as:

$$\left[\left(E_{qm(1)}^{abc} \quad I_{pqm(1)}^{abc} \quad I_{qpm(1)}^{abc}\right); \quad \left(E_{qm(2)}^{abc} \quad I_{pqm(2)}^{abc} \quad I_{qpm(2)}^{abc}\right)\right] \text{ And } \left[E_{p(1)}^{abc} \quad E_{p(2)}^{abc}\right]$$

The measurements of these two loading conditions will result in 12 algebraic equations with 9 unknowns and the unknown ratio correction factors can be calculated using least squares technique on this over determined set of equations [15]. A similar process can be applied to calibrate a set of instrument transformers across a network. With the same one perfect measurement provided by a voltage transformer, a network with n buses and m lines will lead to [6m] equations and [3(n-1) + 6m] unknowns similar to the equations seen in (8). As in the two bus case, measurements made under multiple loading conditions will provide sufficient number of equations in order to provide an over-determined set of equations from which all the unknown ratio correction factors can be calculated. It is important to note that the only instrument transformers that can be calibrated are the ones on the buses and lines where PMUs are connected.

4.4 Load Tap Changer (LTC) position Monitoring using a PMU

This application has proven to very accurately calibrate instrument transformers across a network in theory and simulations. The Dominion Virginia Power 500 kV network was chosen as an actual system upon which the application would be tested for practicability. As was mentioned, this technique requires access to at least one set of three phase voltage transformers (to be used as reference) with a known and accurate RCF. A Potential Transformer (PT) can be used to serve this purpose. However in the case of the Dominion 500 kV application, the location of the designated PT measurement is at a LV bus while the PMU measurement is taken at a HV bus. Therefore, we have to translate the accurate LV measurement to the HV, using the impedance network at the substation where the devices are located. Part of this impedance

network entails an LTC-controlled transformer, and in order to calculate the most accurate impedance network, the LTC positions must be known.

Figure 25 shows a devised scheme that that would allow these LTC positions to be monitored, with an appropriate time stamp. The SEL 421 contains a DC battery monitor that is capable of monitoring a DC voltage. INCON position monitoring devices are used to translate LTC positions into a DC output, which is then fed into the SEL 421. The SEL 421 is then programmed to take the input voltage and perform a series to calculations, ultimately interpreting the LTC position based on the read input voltage value. Firstly, a series of tests were conducted in a lab to test the feasibility of this scheme, using the same devices that would be used in the

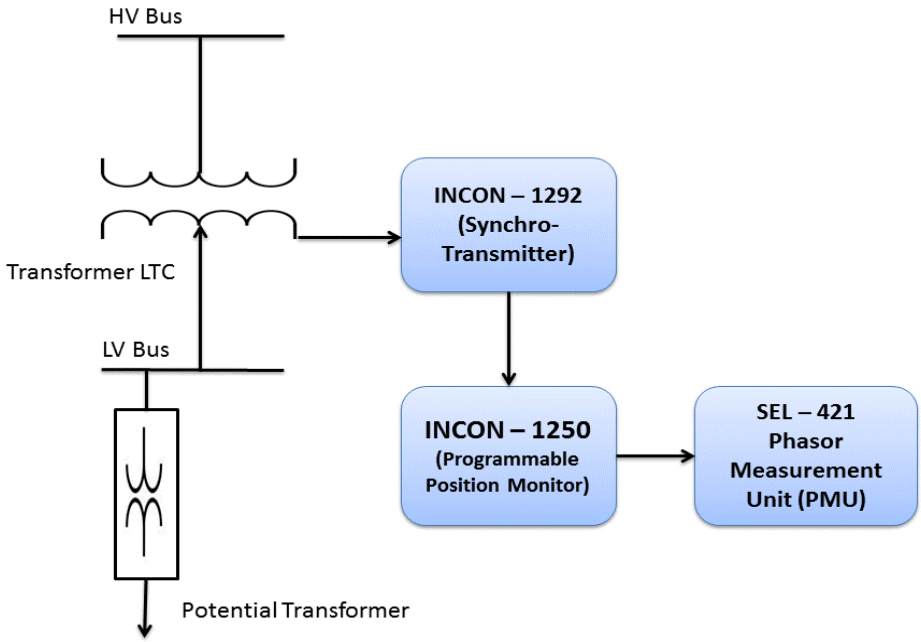


Figure 25: LTC position monitoring schematic

actual application. Figure 26 shows the setup of these devices in the lab. The movement of the transformer taps was simulated by the manual rotation of the synchro-transmitter shaft.

Furthermore, a similar monitoring system for a SCADA application is already installed on location, so tests were also carried out to examine if this new scheme could be implemented in parallel with SCADA without affecting the original SCADA application. Figure 27 shows the setup of both applications in parallel. The SCADA system uses a General Electric D20 Remote terminal Unit to monitor to DC output coming from the INCON 1250 device. A second INCON 1250 is used to connect to the same synchro-transmitter feeding the INCON 1250 of the D20. The output from this new INCON 1250 is then fed into the SEL 421. A series of before and after tests were carried out to ensure there was no change in the LTC readings by the D20. If the

Figure 26: Lab setup for LTC position monitoring scheme testing

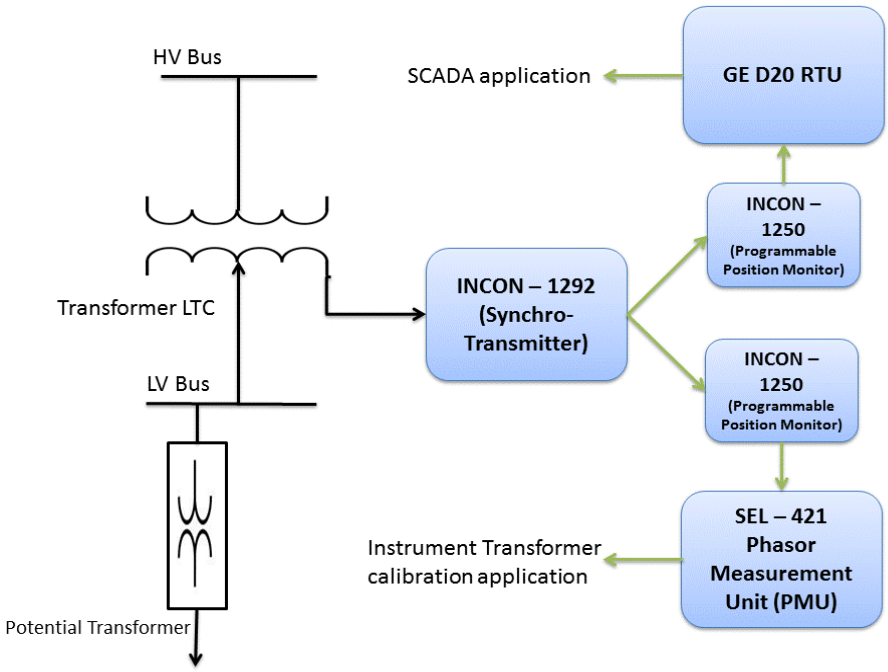


Figure 27: LTC position monitoring schematic in parallel with SCADA application

above scheme could be successfully implemented, the LTC position could be transmitted as an analog value as part of the IEEE C37.118 message [16]. This value would then be used in the translation of the PT measurement on the LV bus to the HV bus, and the instrument transformer calibration scheme can be realized. The following is a more detailed look at the hardware and software used in the LTC position monitoring scheme.

4.4.1 Hardware Description

A) SEL 421 Phasor Measurement Unit

The SEL 421 relay provides high speed distance and directional protection transmission line protection. The relay features extensive metering and data recording including high

resolution data capture and reporting. The relay can function as a PMU if a high accuracy time source is connected to the relay, hence providing time synchronized measurements for a wide range of applications. The SEL 421 supports the IEEE C37.118, Standard for Synchrophasors for Power Systems [17]. The SEL 421 also features a Substation Battery Monitor for DC Quality Assurance. It measures and reports the substation battery voltage for two battery systems. The measured DC voltage is reported and can be accessed through the METER display via serial port communications, on the LCD, and in the Event Report. For this application, math and control variables are used to monitor and process the battery voltage reading.

B) INCON Position Monitoring Devices

The INCON 1292S is a synchro transmitter used to provide an electrical signal corresponding to rotary position. Synchro transmitters are inherently absolute position sensors, which mean the model 1292KS indicates rotary position from a known reference without the need to re-zero each time power is applied. Therefore the absolute position of the shaft is not lost if power is removed from the unit. The device generates three output signals, and the difference in amplitude of these signals indicates the angular position of the shaft [18]. These devices are used in applications to monitor LTCs where position changes can be made to rotate the synchro shaft. The INCON 1292KS is designed to be used in conjunction with the solid state synchro receiver INCON 1250.

The INCON 1250 LTC Programmable Position Monitor is a highly advanced solid state instrument, which measures the absolute position of a synchro transmitter. It provides both a user

definable visual panel indication and optional analog and digital signal outputs suitable for a variety of monitoring and control applications [19]. The device is designed specifically for monitoring power transformer LTC positions. The synchro transmitter is attached to an operating shaft on the LTC and the 1250 is programmed to output the tap positions along with a corresponding current in the range -1 to 1 mA. A 9kΩ resistor was placed across the terminal of this current output, producing a voltage output in the range -9 to 9V to be fed into the SEL 421 PMU.

4.4.2 Software Description

A) SEL AcSELerator Quickset

The SEL AcSELerator Quickset is the software package used to program the control, protection and automation functions into SEL devices [20]. It allows the user to create, test and manage relay settings with a Windows® interface. Users have access to the logic variables that control and monitor all relay input and output terminals. Thus facilitating the ability to develop Boolean or mathematical expressions to perform the desired protection or control function. For this project the software was used to monitor battery monitoring terminals to read the input DC voltage from the INCON devices. After the voltage input is read a number of the SEL 421's Protection Math Variables (PMV) and Protection Control Equation Variables (PSV) were used in the logic necessary to convert the input voltage to the corresponding LTC position. Figure 28 shows a flowchart depicting the coded algorithm used for the conversion. See Appendix B for the details of the code used to convert the voltages to LTC positions.

B) PMU Connection Tester

PMU Connection Tester is a software tool that verifies whether the data streams from known phasor measurement devices are being received. Any device that supports one of the following phasor protocols may be tested using PMU Connection Tester: IEEE C37.118.2-2011, IEEE C37.118-2005, IEC 61850-90-5, IEE 1344-1995, BPA PDCstream, UTK F-NET, SEL Fast Message or Macrodyne. The software provides the time series phasor information related to currents and voltages along with associated frequency information. The data is available in graphical displays and also in exported "csv file" formats for use in other applications. The software can also monitor the analog and digital values associated with a protocol, as was the case for this project. PMV64 is the variable that contains the calculated LTC position after the

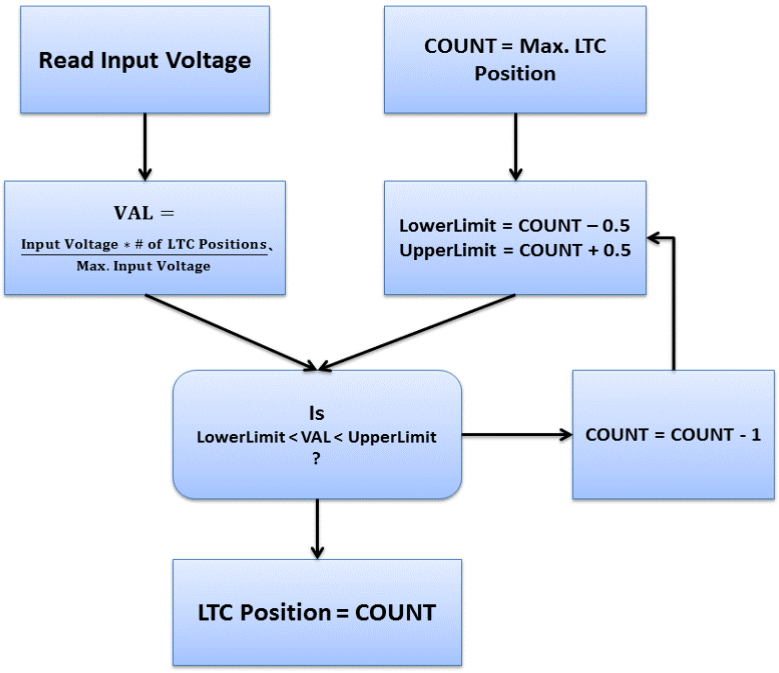


Figure 28: Algorithm used to convert voltage to LTC position with SEL 421

routine is performed on the read input voltage by the above mentioned algorithm. The analog PMV64 variable is viewed through one of the available data frames in the software.

4.4.3 Results

The model LTC used for the tests contained 33 tap positions with 1 neutral at segment number 1. As a stand-alone system, the scheme was able to successfully monitor all 33 LTC positions. To test whether the parallel connection affected the SCADA system, a reference case for the SCADA monitoring system had to be established. The SCADA D20 takes the -1 to 1mA output from the INCON 1250, and translates it into a digital count in the range of -29792 to 29857. These count values are then translated into the LTC position using the formula: **x*16/29788**, where 29788 counts correspond to 1mA. An experiment using just the SCADA system was carried out, and the counts corresponding to all 33 positions was recorded. This experiment was re-run, but this time with the SEL 421 included. Table 1 shows the results comparing both tests.

As can be seen from the difference column, for the majority of LTC positions the counts seen by the D20 RTU remains unchanged with the inclusion of the PMU monitoring device. However, for 4 of the 33 positions the D20 counts were off by 32 counts. To put this in perspective, from the results of the experiments the average difference of counts between each tap position is: 29857 – (-29792)/33 = 1808 counts. A difference of 32 counts represents only 32/1808*100 = 1.8%, which is a very small percentage of this threshold between position

readings, hence rendering it inconsequential in affecting the interpretation of the tap position by the SCADA system.

Table 1: Comparison of results from LTC monitoring experiments

LTC Position #	Counts with SEL 421	Counts without SEL 421	Difference
16	29857	29857	0
15	28001	28001	0
14	26113	26113	0
13	24257	24257	0
12	22401	22401	0
11	20513	20545	-32
10	18657	18657	0
9	16801	16801	0
8	14945	14945	0
7	13089	13089	0
6	11201	11233	-32
5	9345	9345	0
4	7489	7489	0
3	5601	5633	-32
2	3745	3745	0
1_1	1889	1889	0
0	33	33	0
-1	-1824	-1824	0
-2	-3680	-3680	0
-3	-5536	-5536	0
-4	-7424	-7392	-32
-5	-9376	-9376	0
-6	-11136	-11136	0
-7	-13088	-13088	0
-8	-14944	-14944	0
-9	-16800	-16800	0
-10	-18656	-18656	0
-11	-20512	-20512	0
-12	-22368	-22368	0
-13	-24224	-24224	0
-14	-26080	-26080	0
-15	-27936	-27936	0
-16	-29792	-29792	0

Chapter 5

Conclusion

This thesis outlines two novel applications of synchrophasor data that have not yet received widespread attention in the power industry and academia. This is because they are still in their developmental stages, and a lot of work is still needed to bring them up to production grade. I still think it is important to give some exposure to these concepts however, and this thesis also outlines the work I performed in assisting the process of these applications becoming fully functional.

The FNET digital audio authentication has the potential to change the way fraudulent digital recordings are detected. As the digital signal processing techniques used to extract the ENF signal from the recordings improve and the data mining techniques to search synchrophasor databases, this concept will grow to eventually realize its potential. The forensic specialists whom will be using this technique will need a tool however, and I have developed a graphical user interface that will provide the framework around which the future implementation of this concept can be developed. I have also laid out the foundation of an automatic tampering detection procedure that could potentially be used in the detecting of fraudulent audio recordings.

The Dominion Virginia Power three phase instrument transformer calibration scheme has the potential to change the way instrument transformer's ratio correction factors are estimated. It is definitely a step up from manual calibration, or the single phase calibration methods using

telemetered transducer measurements. In theory and simulations the scheme worked well, but some issues arose when it came to the practical application using the Dominion 500 kV network. A very accurate potential transformer measurement was needed for the successful implementation of the scheme, but the transducer was located on the wrong bus. I have outlined the process by which I was able to acquire this PT measurement, and also integrate my technique without affecting an existing SCADA monitoring system.

List of References

[1] North American Electric Reliability Corporation (NERC), “Real-Time Application of Synchrophasors for Improving Reliability” October 18, 2009.

[2] Brian Ray Miller, “Concept for Next Generation Phasor Measurement: A Low-Cost, Self-Contained, and Wireless Design,” M.S. thesis, EECS, U. of Tennessee, Knoxville, TN, 2010.

[3] M. Adamiak, B. Kasztenny, and W. Premerlani, “Synchrophasors: Definition, measurement, and application,” presented at the 59th Annu. Georgia Tech Protective Relaying, Atlanta, GA, Apr. 27–29, 2005.

[4] North American SyncroPhasor Initiative (NASPI). “Synchrophasor Technology Roadmap.” March 13, 2009.

[5] E. B. Brixen, “Techniques for the Authentication of Digital Audio Recordings,” 122nd Audio Engineering Society Convention, Vienna, Austria, 2007.

[6] A. Cooper, “The Electric Network Frequency (ENF) As An Aid To Authenticating Forensic Digital Audio Recordings – An Automated Approach,” Proceedings of AES 33rd International Conference, Denver, CO, USA, 2008.

[7] Y. Zhang, P.N. Markham, T. Xia, et al., "Wide-Area Frequency Monitoring Network (FNET) Architecture and Applications," *IEEE Transactions on Smart Grid,* vol.1, no.2, pp.159-167, Sept. 2010.

[8] H. Karimi, “Estimation of frequency and its rate of change for applications in power systems,” *IEEE Trans. Power Del.,* vol. 19, no. 2, pp. 472–480, Apr. 2004, etc.

[9] L. Wang, J, Burgett, J. Zuo, C. Xu, B.J. Billian, R.W. Conners, Y. Liu, "Frequency Disturbance Recorder Design and Developments," *Power Engineering Society General Meeting*, June, 2007.

[10] "OpenPDC: The Open Source Phasor Data Concentrator," online: http://openpdc.codeplex.com/.

[11] M. Golshani, G.A. Taylor, I. Pisica, P. Ashton, " Laboratory-based deployment and investigation of PMU and OpenPDC capabilities," in *10th IET International Conference on AC and DC Power Transmission,* Birmingham, 2012, pp 1-6.

[12] Y. Liu, Z. Yuan, P.N. Markham, R. Conners, Yilu Liu, "Wide-area Frequency as a criterion for Digital Audio Recording Authentication," *Power and Energy Society General Meeting,* San Diego, CA, 2011, pp 1-7.

[13] M. M. Adibi and R. J. Kafka, "Minimization of uncertainties in analog measurements for use in state estimation", Power Systems, IEEE Transactions on, vol. 5, pp.902-910,1990.

[14] A. G. Phake, J.S. Thorp, R. F. Nuqui, M. Zhou, "Recent Developments in state estimation with phasor measurements", Power Systems Conference and Exposition 2009, PSCE '09.

[15] Zhongyu Wu, Kyle Thomas, Rui Sun, Virgilo Centeno, Arun G. Phadke, "Three Phase Instrument transformer Calibration with Synchronized Phasor Measurements," ISGT, 2012, pp. 1-6.

[16] B. Flerchinger, R. Moxley and E. Ersonmez, "All the Data Fit to Print – Applying All the Available Synchrophasor Information," Scwheitzer Engineering Laboratory, 2011.

[17] *SEL-421 Relay Protection and Automation System Instruction Manual*, Schweitzer Engineering Laboratories INC., Pullman, WA, 2011.

[18] *INCON 1292 Application Bulletin*, Intelligent Controls INC., Saco, ME, 2008.

[19] *Installation and Programming Manual for Model 1250-LTC Programmable Position Monitor*, Intelligent Controls INC., Saco, ME, 2009.

[20] *AcSELerator Quickset Designer Instruction Manual*, Schweitzer Engineering Laboratories INC., Pullman, WA, 2006.

Appendix

Appendix A

GUI Code

```
function varargout = ENF_Analyzer(varargin)
% ENF_ANALYZER MATLAB code for ENF_Analyzer.fig
%      ENF_ANALYZER, by itself, creates a new ENF_ANALYZER or raises the
existing
%      singleton*.
%
%      H = ENF_ANALYZER returns the handle to a new ENF_ANALYZER or the
handle to
%      the existing singleton*.
%
%      ENF_ANALYZER('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ENF_ANALYZER.M with the given input
arguments.
%
%      ENF_ANALYZER('Property','Value',...) creates a new ENF_ANALYZER or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before ENF_Analyzer_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to ENF_Analyzer_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ENF_Analyzer

% Last Modified by GUIDE v2.5 31-May-2012 10:42:25

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ENF_Analyzer_OpeningFcn, ...
                   'gui_OutputFcn',  @ENF_Analyzer_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

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

```
% --- Executes just before ENF_Analyzer is made visible.
function ENF_Analyzer_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ENF_Analyzer (see VARARGIN)

% Choose default command line output for ENF_Analyzer
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ENF_Analyzer wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ENF_Analyzer_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in BrowseTag.
function BrowseTag_Callback(hObject, eventdata, handles)
% hObject    handle to BrowseTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    file = uigetfile;
    set(handles.FileTextTag,'String',file);
    set(handles.BrowseTag,'UserData',file);

% --- Executes on button press in AudioTag.
function AudioTag_Callback(hObject, eventdata, handles)
% hObject    handle to AudioTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    file = get(handles.BrowseTag,'UserData');
    nStartSeconds = get(handles.BeginTag,'UserData'); % Start Time in seconds
    nEndSeconds = get(handles.EndTag,'UserData'); % End Time in seconds
    [~, Fs, ~, ~] = wavread(file, [1 1000]);
```

```
    nStartIndex = nStartSeconds * Fs;
    nEndIndex = nEndSeconds * Fs;

    nSize = wavread(file, 'size');
    if nEndIndex > nSize(1)
        warndlg('End point exceeds length of recording','Bad Input')
        return
    end
    datalength = 60*Fs;
    data = [];
    for i = nStartIndex : datalength : nEndIndex

        nEnd = i + datalength - 1;
        if  nEnd > nEndIndex
            nEnd = nEndIndex;
        end
        vals = wavread(file,[i nEnd]);
        data = [data; vals];
    end
    data = data(:,1);
    plot(data)
    title('Audio Recording')
    xlabel('Samples')
    ylabel('Amplitude')


% --- Executes on button press in ENFTag.
function ENFTag_Callback(hObject, eventdata, handles)
% hObject    handle to ENFTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    strFile = get(handles.BrowseTag,'UserData');
    isWindow = 1;       % 0: no window function(or defaulted rectangle
window); 1: window function
    nCenterFreq = 60;   % Hz
    zeroPadding = get(handles.ZeroTag,'UserData');    % Padding zero at the
end of a signal
    winSize = get(handles.WinTag,'UserData');        % Second
    hopSize = 0.1;      % Second

    % Obtain the sampling rate
    [~, Fs, ~, ~] = wavread(strFile, [1 1000]);

    % Design the filter parameters
    % All frequency values are in Hz.
    if Fs == 8000  % the sample rate is 8000 Hz
        nDown = 8;
        N    = 151;      % Order
    elseif Fs == 11025   % the sample rate is 11025 Hz
        nDown = 9;
```

```
        N    = 151;       % Order
    elseif Fs == 44100    % the sample rate is 44100 Hz
        nDown = 30;
        N    = 251;       % Order
    end

    Fc   = Fs/(2*nDown)*0.5;      % Cutoff Frequency
    flag = 'scale';  % Sampling Flag
    % Create the window vector for the design algorithm.
    win  = hamming(N+1);
    % Calculate the coefficients using the FIR1 function.
    b    = fir1(N, Fc/(Fs/2), 'low', win, flag);
    antialias = dfilt.dffir(b);

    % All frequency values are in Hz.
    FsDown = Fs/nDown;  % Sampling Frequency for bandpass filter

    Fstop1 = nCenterFreq - 1;          % First Stopband Frequency
    Fpass1 = nCenterFreq - 0.4;        % First Passband Frequency
    Fpass2 = nCenterFreq + 0.4;        % Second Passband Frequency
    Fstop2 = nCenterFreq + 1;          % Second Stopband Frequency
    Astop1 = 80;          % First Stopband Attenuation (dB)
    Apass  = 0.5;         % Passband Ripple (dB)
    Astop2 = 80;          % Second Stopband Attenuation (dB)
    match  = 'passband';  % Band to match exactly

    % Construct an FDESIGN object and call its CHEBY2 method.
    h  = fdesign.bandpass(Fstop1, Fpass1, Fpass2, Fstop2, Astop1, Apass, ...
                          Astop2, FsDown);
    bandpass = design(h, 'cheby2', 'MatchExactly', match);

    %[antialias, bandpass, nDown] = DesignFilter(Fs, nCenterFreq);

    %Get input variables
    nSection = 60; %get(handles.VarTag2,'UserData');   % Constant interval by
which data will be read from file
    nStartSeconds = get(handles.BeginTag,'UserData'); % Start Time in seconds
    nEndSeconds = get(handles.EndTag,'UserData'); % End Time in seconds
    Datalength = nSection * Fs;
    nStartIndex = nStartSeconds * Fs;
    nEndIndex = nEndSeconds * Fs;
    %dataseg = 300*Fs;
    nSize = wavread(strFile, 'size');
    % Warning if endpoint exceeds length of recording
    if nEndIndex > nSize(1)
        warndlg('End point exceeds length of recording','Bad Input')
        return
    end

    a = 1;  %Iteration Storer for results
    overlapForBP = 10;     % second
```

```
    extendForWindow = 10;  % second

    % Get the data section from wave and FFT analysis
    for i = nStartIndex : Datalength : nEndIndex  % for-loop 1

        nStart = i - overlapForBP*Fs + 1;
        if nStart < 0
            nStart = nStartIndex;
        end

        % Here we select the data which extend 10 more seconds
        nEnd = i + Datalength - 1 + extendForWindow*Fs;%Datalength - 1 +
extendForWindow*Fs;
        if  nEnd > nEndIndex
            nEnd = nEndIndex;
        end

        % Read the data section and elimilate the DC component
        yRawData = wavread(strFile,[nStart nEnd]); %Data(nStart:nEnd,1);
        yRawData = yRawData(:,1);
        ymean = mean(yRawData);
        yDetrend = yRawData - ymean;

        % Anti-alias before downsample(low-band pass filter)
        yAntialis = filter(antialias, yDetrend);
        % Down sample
        yDownSample = downsample(yAntialis, nDown);
        % Band-pass filter
        yFilter = filter(bandpass, yDownSample);

        % Sampling frequnecy after down-sampling
        FsFFT = Fs/nDown;
        % Window size,hop size, and FFT point
        LengData = round(winSize*FsFFT); % data point
        HopSize = hopSize*FsFFT;
        %NFFT = (zeroPadding + 1)*round(LengData);

        % Here, we calculate the number of data sections, that we must
        % subtract 10-second data

        x1 = yFilter(overlapForBP*FsFFT + 1:end);
        k = round((length(x1) - extendForWindow*FsFFT)/(HopSize));

        % FFT analysis
        n = 0;        % iteration number

        for j =  1:k  %% for-loop 2
            nIndex = round(n*HopSize+1);
            if nIndex + LengData > length(x1)
                yData = x1(nIndex : end);
```

```
            else
                yData = x1(nIndex : nIndex + LengData - 1);
            end

            % Add window function
            if isWindow == 1
                % Blackman window
                nWin = blackman(length(yData));
                yData1 = yData.*nWin;
            else
                yData1 = yData;
            end
            lengthdata = length(yData1);
            NFFT = (zeroPadding + 1)*round(lengthdata);
            NFFT = 2^(nextpow2(NFFT));
            % Divided by real length
            Yres = fft(yData1, NFFT)/length(yData1);
            yMag = abs(Yres);
            yPhase = angle(Yres);
            Freq = FsFFT/2 * linspace(0,1,NFFT/2);
            Nf1 = round((nCenterFreq - 1)/FsFFT*NFFT);
            Nf2 = round((nCenterFreq + 1)/FsFFT*NFFT);
            % Find the maximal magnitude
            [~, index] = max(2*abs(Yres(Nf1:Nf2,:)));

%             [~, index] = max(yMag(Nf1:Nf2,1));

            xFreq1 = [Freq(index + Nf1 - 2) Freq(index + Nf1 - 1) Freq(index
+ Nf1)];
            yMag1 = [yMag(index + Nf1 - 2) yMag(index + Nf1 - 1) yMag(index +
Nf1)];
            yPhase1 = [yPhase(index + Nf1 - 2) yPhase(index + Nf1 - 1)
yPhase(index + Nf1)];

            % Interpolation based on xFreq1,yMag1 and yPhase1
            xFreq2 = (Freq(index + Nf1 - 2):0.0001:Freq(index + Nf1));
            yMag2 = interp1(xFreq1,yMag1,xFreq2,'spline');
            yPhase2 = interp1(xFreq1,yPhase1,xFreq2,'spline');
            [~, nT] = max(yMag2);

            FreqV(a) = xFreq2(nT);
            PhaseV(a) = yPhase2(nT);

            a = a+1;

            n = n + 1;

        end % end of for-loop 2

    end  % end of for-loop 1
```

```
    String = get(handles.DataLocationTag,'String');
    fl = exist(String);
    if fl == 2
        save(String,'-append','FreqV')
    else
        save(String,'FreqV')
    end

    time = [1:length(FreqV)];
    time = time./10;
    plot(time,FreqV)
    title('Estimated Frequency')
    xlabel('Time(s)')
    ylabel('Frequency(Hz)')


% --- Executes on button press in PhaseTag.
function PhaseTag_Callback(hObject, eventdata, handles)
% hObject    handle to PhaseTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    strFile = get(handles.BrowseTag,'UserData');
    isWindow = 1;       % 0: no window function(or defaulted rectangle
window); 1: window function
    nCenterFreq = 60;   % Hz
    zeroPadding = get(handles.ZeroTag,'UserData');    % Padding zero at the
end of a signal
    winSize = get(handles.WinTag,'UserData');        % Second
    hopSize = 0.1;      % Second

    % Obtain the sampling rate
    [~, Fs, ~, ~] = wavread(strFile, [1 1000]);

    % Design the filter parameters
    % All frequency values are in Hz.
    if Fs == 8000  % the sample rate is 8000 Hz
        nDown = 8;
        N    = 151;      % Order
    elseif Fs == 11025   % the sample rate is 11025 Hz
        nDown = 9;
        N    = 151;      % Order
    elseif Fs == 44100   % the sample rate is 44100 Hz
        nDown = 30;
        N    = 251;      % Order
    end

    Fc   = Fs/(2*nDown)*0.5;      % Cutoff Frequency
    flag = 'scale';  % Sampling Flag
    % Create the window vector for the design algorithm.
    win  = hamming(N+1);
```

```
    % Calculate the coefficients using the FIR1 function.
    b    = fir1(N, Fc/(Fs/2), 'low', win, flag);
    antialias = dfilt.dffir(b);

    % All frequency values are in Hz.
    FsDown = Fs/nDown;  % Sampling Frequency for bandpass filter

    Fstop1 = nCenterFreq - 1;          % First Stopband Frequency
    Fpass1 = nCenterFreq - 0.4;        % First Passband Frequency
    Fpass2 = nCenterFreq + 0.4;        % Second Passband Frequency
    Fstop2 = nCenterFreq + 1;          % Second Stopband Frequency
    Astop1 = 80;          % First Stopband Attenuation (dB)
    Apass  = 0.5;         % Passband Ripple (dB)
    Astop2 = 80;          % Second Stopband Attenuation (dB)
    match  = 'passband';  % Band to match exactly

    % Construct an FDESIGN object and call its CHEBY2 method.
    h  = fdesign.bandpass(Fstop1, Fpass1, Fpass2, Fstop2, Astop1, Apass, ...
                          Astop2, FsDown);
    bandpass = design(h, 'cheby2', 'MatchExactly', match);

    %[antialias, bandpass, nDown] = DesignFilter(Fs, nCenterFreq);

    %Get input variables
    nSection = 60; %get(handles.VarTag2,'UserData');   % Constant interval by
which data will be read from file
    nStartSeconds = get(handles.BeginTag,'UserData'); % Start Time in seconds
    nEndSeconds = get(handles.EndTag,'UserData'); % End Time in seconds
    Datalength = nSection * Fs;
    nStartIndex = nStartSeconds * Fs;
    nEndIndex = nEndSeconds * Fs;
    %dataseg = 300*Fs;
    nSize = wavread(strFile, 'size');
    % Warning if endpoint exceeds length of recording
    if nEndIndex > nSize(1)
        warndlg('End point exceeds length of recording','Bad Input')
        return
    end

    a = 1;  %Iteration Storer for results
    overlapForBP = 10;     % second
    extendForWindow = 10;  % second

    % Get the data section from wave and FFT analysis
    for i = nStartIndex : Datalength : nEndIndex  % for-loop 1

        nStart = i - overlapForBP*Fs + 1;
        if nStart < 0
            nStart = nStartIndex;
        end
```

```
        % Here we select the data which extend 10 more seconds
        nEnd = i + Datalength - 1 + extendForWindow*Fs;%Datalength - 1 +
extendForWindow*Fs;
        if  nEnd > nEndIndex
            nEnd = nEndIndex;
        end

        % Read the data section and elimilate the DC component
        yRawData = wavread(strFile,[nStart nEnd]); %Data(nStart:nEnd,1);
        yRawData = yRawData(:,1);
        ymean = mean(yRawData);
        yDetrend = yRawData - ymean;

        % Anti-alias before downsample(low-band pass filter)
        yAntialis = filter(antialias, yDetrend);
        % Down sample
        yDownSample = downsample(yAntialis, nDown);
        % Band-pass filter
        yFilter = filter(bandpass, yDownSample);

        % Sampling frequnecy after down-sampling
        FsFFT = Fs/nDown;
        % Window size,hop size, and FFT point
        LengData = round(winSize*FsFFT); % data point
        HopSize = hopSize*FsFFT;
        %NFFT = (zeroPadding + 1)*round(LengData);

        % Here, we calculate the number of data sections, that we must
        % subtract 10-second data

        x1 = yFilter(overlapForBP*FsFFT + 1:end);
        k = round((length(x1) - extendForWindow*FsFFT)/(HopSize));

        % FFT analysis
        n = 0;        % iteration number

        for j =  1:k  %% for-loop 2
            nIndex = round(n*HopSize+1);
            if nIndex + LengData > length(x1)
                yData = x1(nIndex : end);
            else
                yData = x1(nIndex : nIndex + LengData - 1);
            end

            % Add window function
            if isWindow == 1
                % Blackman window
                nWin = blackman(length(yData));
                yData1 = yData.*nWin;
```

```
            else
                yData1 = yData;
            end
            lengthdata = length(yData1);
            NFFT = (zeroPadding + 1)*round(lengthdata);
            NFFT = 2^(nextpow2(NFFT));
            yFFT = fft(yData1,NFFT);
            yMag = abs(yFFT);
            yPhase = angle(yFFT);
            Freq = (0:length(yFFT)-1)'*FsFFT/length(yFFT);

            [~,maxIndex] = max(yMag);

           %% Eric Jacobsen's method
           constantA = pi*(NFFT-1)/NFFT; % offset of phase angle
           nom = yFFT(maxIndex - 1) - yFFT(maxIndex + 1);
           denom = 2*yFFT(maxIndex) - yFFT(maxIndex - 1) - yFFT(maxIndex +
1);
           delta = real(nom/denom);

           freqEJ = (maxIndex + delta - 1)*FsFFT/length(yFFT);
           phaseEJ = yPhase(maxIndex) - constantA * delta;

           FreqV(a) = freqEJ;   % estimated frequency
           PhaseV(a) = phaseEJ;  % estimated phase angle

            a = a+1;

            n = n + 1;

        end % end of for-loop 2

    end  % end of for-loop 1

    PHASE = unwrap(PhaseV);

    String = get(handles.DataLocationTag,'String');
    fl = exist(String);
    if fl == 2
        save(String,'-append','PHASE')
    else
        save(String,'PHASE')
    end

    time = [1:length(PHASE)];
    time = time./10;
    plot(time,PHASE)
    title('Estimated Phase Angle')
    xlabel('Time(s)')
    ylabel('Angle(rad)')
```

```
function WinTag_Callback(hObject, eventdata, handles)
% hObject    handle to WinTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of WinTag as text
%        str2double(get(hObject,'String')) returns contents of WinTag as a
double

    winSize = str2double(get(hObject,'string'));
    if isnan(winSize)
        errordlg('You must enter a numeric value','Bad Input','modal')
        uicontrol(hObject)
        return
    end
    set(handles.WinTag,'UserData',winSize);

% --- Executes during object creation, after setting all properties.
function WinTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to WinTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function BeginTag_Callback(hObject, eventdata, handles)
% hObject    handle to BeginTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of BeginTag as text
%        str2double(get(hObject,'String')) returns contents of BeginTag as a
double
    begindata = str2double(get(hObject,'string'));
    if isnan(begindata)
        errordlg('You must enter a numeric value','Bad Input','modal')
        uicontrol(hObject)
        return
    end
    set(handles.BeginTag,'UserData',begindata);
```

```
% --- Executes during object creation, after setting all properties.
function BeginTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BeginTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function ZeroTag_Callback(hObject, eventdata, handles)
% hObject    handle to ZeroTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ZeroTag as text
%        str2double(get(hObject,'String')) returns contents of ZeroTag as a
double
    zeroPad = str2double(get(hObject,'string'));
    if isnan(zeroPad)
        errordlg('You must enter a numeric value','Bad Input','modal')
        uicontrol(hObject)
        return
    end
    set(handles.ZeroTag,'UserData',zeroPad);

% --- Executes during object creation, after setting all properties.
function ZeroTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ZeroTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --------------------------------------------------------------------
function uipushtool2_ClickedCallback(hObject, eventdata, handles)
% hObject    handle to uipushtool2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% --- Executes on button press in SaveTag.
function SaveTag_Callback(hObject, eventdata, handles)
% hObject    handle to SaveTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    file = uiputfile;
    set(handles.DataLocationTag,'String',file);



% --- Executes on button press in HelpTag.
function HelpTag_Callback(hObject, eventdata, handles)
% hObject    handle to HelpTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    notebook('GUI_Help_File.doc')


% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to WinTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of WinTag as text
%        str2double(get(hObject,'String')) returns contents of WinTag as a
double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to WinTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
```

```
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to BeginTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of BeginTag as text
%        str2double(get(hObject,'String')) returns contents of BeginTag as a
double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BeginTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to ZeroTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ZeroTag as text
%        str2double(get(hObject,'String')) returns contents of ZeroTag as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ZeroTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

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


% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function EndTag_Callback(hObject, eventdata, handles)
% hObject    handle to EndTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of EndTag as text
%        str2double(get(hObject,'String')) returns contents of EndTag as a
double
    enddata = str2double(get(hObject,'string'));
    if isnan(enddata)
        errordlg('You must enter a numeric value','Bad Input','modal')
        uicontrol(hObject)
        return
    end
    set(handles.EndTag,'UserData',enddata);

% --- Executes during object creation, after setting all properties.
function EndTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EndTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes during object creation, after setting all properties.
function DataLocationTag_CreateFcn(hObject, eventdata, handles)
% hObject    handle to DataLocationTag (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

GUI Instruction Manual

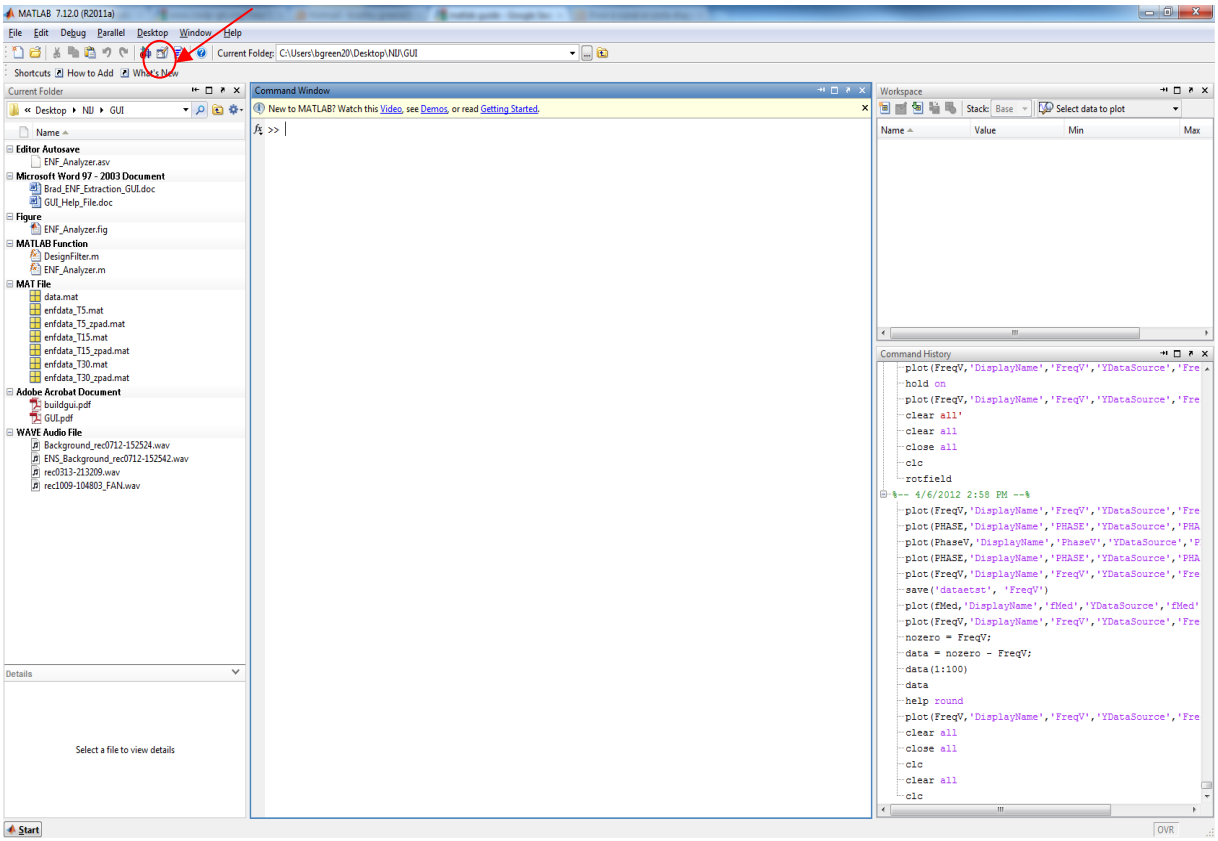
This document provides a walk-through of the steps required to perform several functions using the MATLAB GUI.

The MATLAB GUI folder should already be loaded on your desktop.

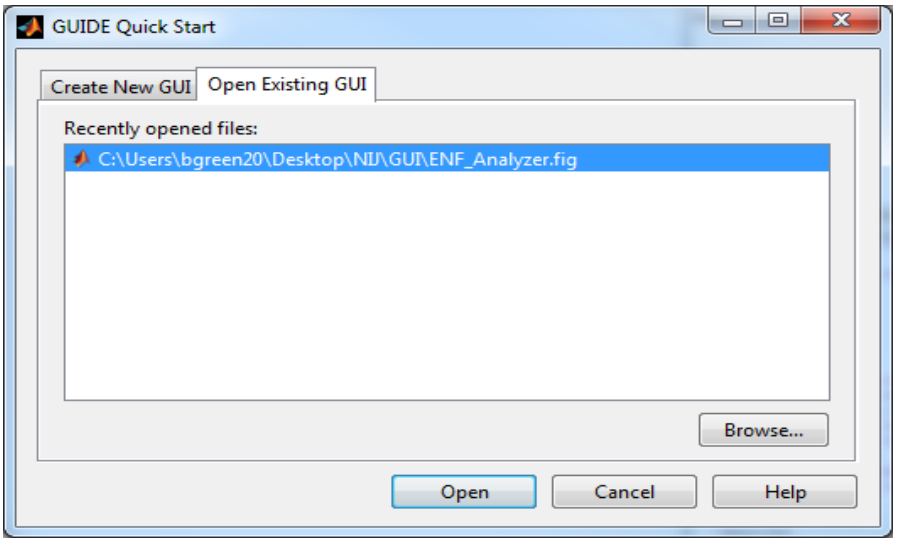
Copy all audio files you intend to analyze into this folder.

To open the ENF analyzer GUI follow these steps:

1) Open MATLAB.
2) In the MATLAB window open the GUIDE tool.

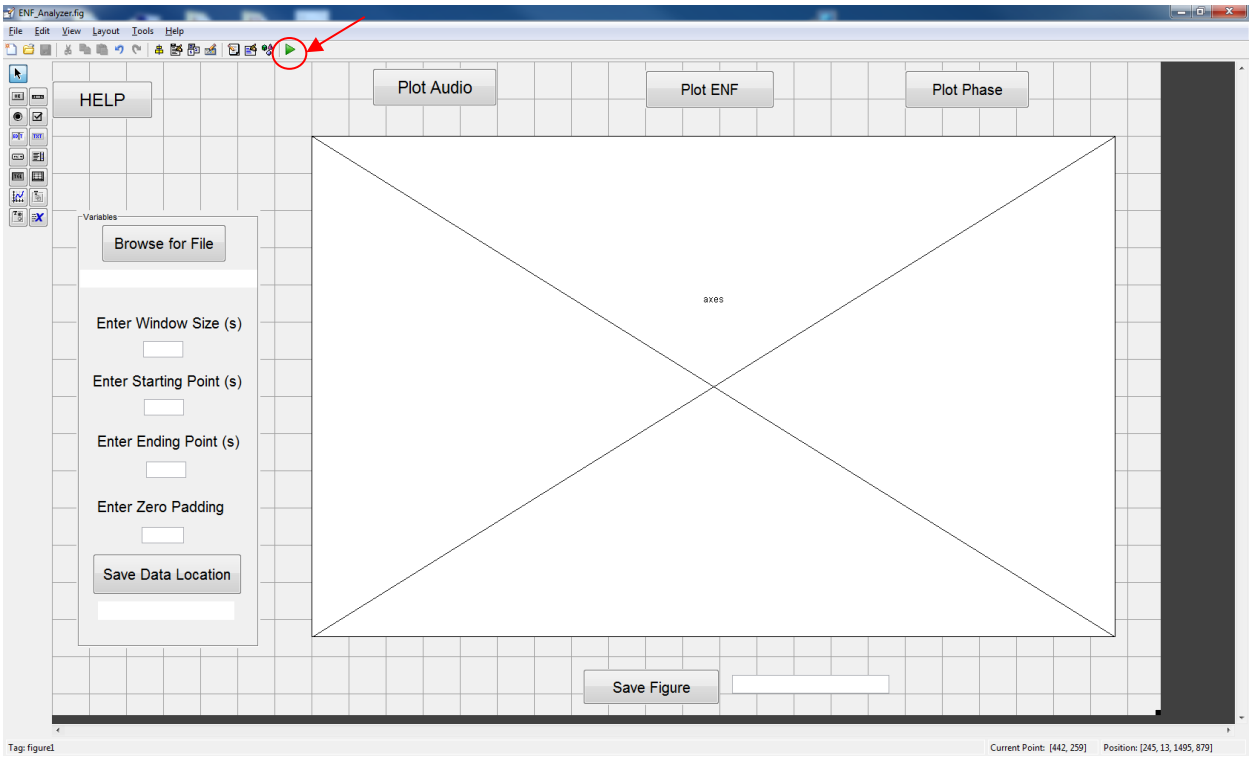


3) The screen below will appear.



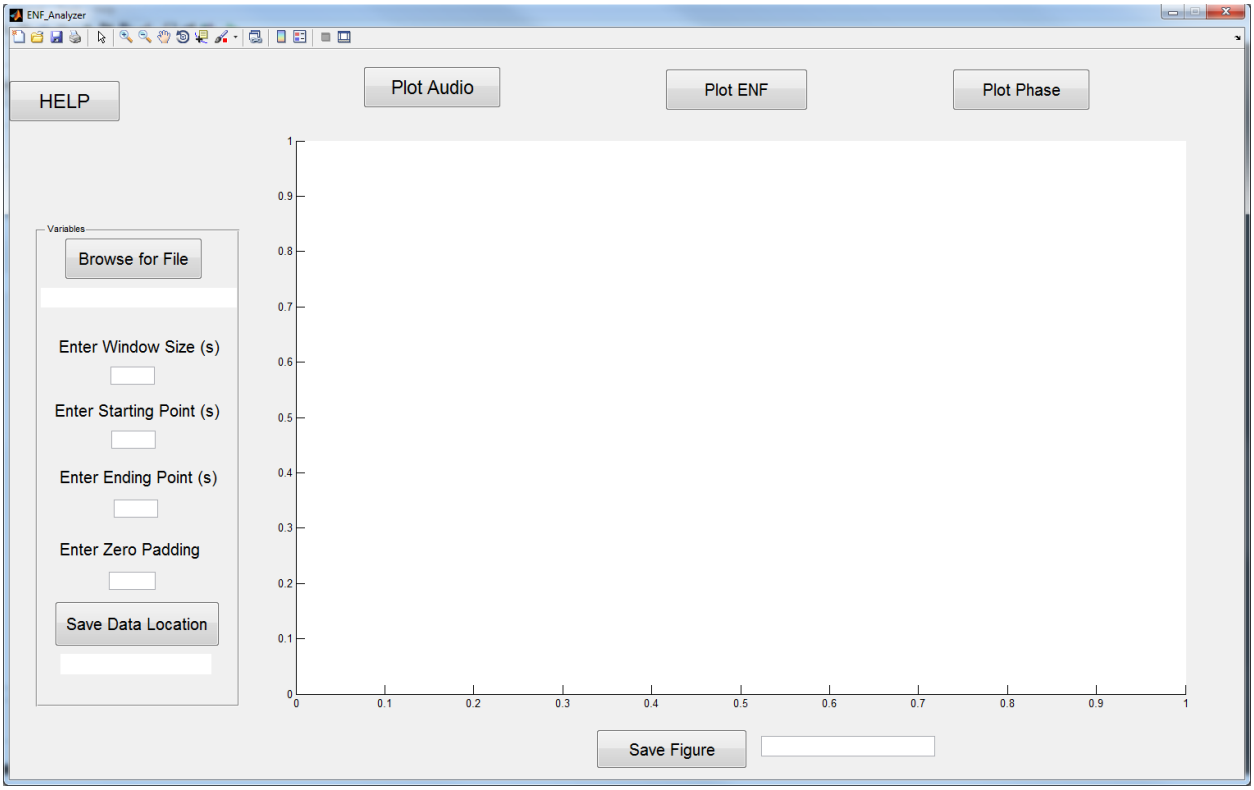
Browse to the MATLAB GUI folder and open 'ENF_Analyzer.fig'.

4) The screen below will appear.



Press the ‘Run Figure’ button shown above.

5) This will open up ENF analyzer program shown below.



You are now ready to use GUI to perform the extraction processes. If you look at the GUI you will see a box titled ‘Variables’ that includes several inputs that are required to perform the functions. ***Note: All** of these variables **must** be filled in before you proceed with the plots.* There is also the axes where the plots appear, and above this you will find the various plotting functions that perform the extractions etc. and provide the results on the axes.

To use the tool follow these steps:

1) Press the ‘Browse for File’ button, and select the recording you will like to analyze.

2) Enter the ‘Window Size’, ‘Starting Point’, ‘Ending Point’ and ‘Zero Padding Factor’ you would like to use for the analysis.

3) Press the ‘Save Data Location’ button to name the file you would like the extracted data to be saved into (i.e. Extracted **ENF** and **Phase** Data).

4) Press the plotting function button you would like to perform to analyze the recording *('Plot Audio', 'Plot ENF', 'Plot Phase').* The results of the analysis will then be displayed on the axes. This step also saves the extracted data in the previously selected file location.

5) Press the 'Save Figure' button to save the plots of the various functions.

The tool bar at the top of the GUI allows you to perform the various functions as with traditional MATLAB figures for example: zooming, printing etc.

Also the 'HELP' button opens up this word file with instructions on how to use the program.

Appendix B

A total of 9 Protection Math Variables (PMV), and 33 Protection Control Equation Variables (PSV) were used in the logic necessary for the conversion of the input voltage of the PMU, to the corresponding LTC position. The programming was performed using the AcSELerator QuickSet software and the coded logic is shown below, with the final PMV64 variable corresponding to the LTC position.

```
PMV64 := (DC1 / 9.000000) * 16.000000
PMV60 := PMV64
PMV63 := PMV64 - 0.500000
PMV62 := PMV64 + 0.500000
PSV01 := (16 > PMV63 AND 16 <= PMV62)
PSV02 := (15 > PMV63 AND 15 <= PMV62)
PSV03 := (14 > PMV63 AND 14 <= PMV62)
PSV04 := (13 > PMV63 AND 13 <= PMV62)
PSV05 := (12 > PMV63 AND 12 <= PMV62)
PSV06 := (11 > PMV63 AND 11 <= PMV62)
PSV07 := (10 > PMV63 AND 10 <= PMV62)
PSV08 := (9 > PMV63 AND 9 <= PMV62)
PSV09 := (8 > PMV63 AND 8 <= PMV62)
PSV10 := (7 > PMV63 AND 7 <= PMV62)
PSV11 := (6 > PMV63 AND 6 <= PMV62)
PSV12 := (5 > PMV63 AND 5 <= PMV62)
PSV13 := (4 > PMV63 AND 4 <= PMV62)
PSV14 := (3 > PMV63 AND 3 <= PMV62)
PSV15 := (2 > PMV63 AND 2 <= PMV62)
```

PSV16 := (1 > PMV63 AND 1 <= PMV62)

PSV17 := (0 > PMV63 AND 0 <= PMV62)

PSV18 := (PMV63 < -1 AND PMV62 >= -1)

PSV19 := (PMV63 < -2 AND PMV62 >= -2)

PSV20 := (PMV63 < -3 AND PMV62 >= -3)

PSV21 := (PMV63 < -4 AND PMV62 >= -4)

PSV22 := (PMV63 < -5 AND PMV62 >= -5)

PSV23 := (PMV63 < -6 AND PMV62 >= -6)

PSV24 := (PMV63 < -7 AND PMV62 >= -7)

PSV25 := (PMV63 < -8 AND PMV62 >= -8)

PSV26 := (PMV63 < -9 AND PMV62 >= -9)

PSV27 := (PMV63 < -10 AND PMV62 >= -10)

PSV28 := (PMV63 < -11 AND PMV62 >= -11)

PSV29 := (PMV63 < -12 AND PMV62 >= -12)

PSV30 := (PMV63 < -13 AND PMV62 >= -13)

PSV31 := (PMV63 < -14 AND PMV62 >= -14)

PSV32 := (PMV63 < -15 AND PMV62 >= -15)

PSV33 := (PMV63 < -16 AND PMV62 >= -16)

PMV01 := PSV01 * 16.000000 + PSV02 * 15.000000 + PSV03 * 14.000000 + PSV04 * 13.000000 + PSV05 * 12.000000 + PSV06 * 11.000000 + PSV07 * 10.000000

PMV02 := PSV08 * 9.000000 + PSV09 * 8.000000 + PSV10 * 7.000000 + PSV11 * 6.000000 + PSV12 * 5.000000 + PSV13 * 4.000000 + PSV14 * 3.000000

PMV03 := PSV15 * 2.000000 + PSV16 * 1.000000 + PSV17 * 0.000000 + PSV18 * -1.000000 + PSV19 * -2.000000 + PSV20 * -3.000000 + PSV21 * -4.000000

PMV04 := PSV22 * -5.000000 + PSV23 * -6.000000 + PSV24 * -7.000000 + PSV25 * -8.000000 + PSV26 * -9.000000 + PSV27 * -10.000000 + PSV28 * -11.000000

PMV05 := PSV29 * -12.000000 + PSV30 * -13.000000 + PSV31 * -14.000000 + PSV32 * -15.000000 + PSV33 * -16.000000

PMV64 := PMV01 + PMV02 + PMV03 + PMV04 + PMV05

Vita

Bradley Kerwin Greene was born in the Caribbean country of Trinidad and Tobago. He received a Bachelor of Science degree in Electrical Engineering from Morgan State University in 2011. He then began graduate studies at the University of Tennessee toward a Master of Science in Electrical Engineering degree, with a concentration in power systems. His research interests include smart grid and synchrophasor applications, power system economics, digital signal processing applications to power systems, and renewable energy technologies.