UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL & APPLIED SCIENCES

REMOTE HUMAN SENSING AND CONDITIONING OF DATA

M. Sc. THESIS
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

BY
TALİP ESKİKALE
AUGUST 2011

Remote Human Sensing and Conditioning of Data

M.Sc. Thesis
in
Electrical and Electronics Engineering
University of Gaziantep

Supervisor:
Asst. Prof. Dr. Tolgay KARA

by

Talip ESKİKALE
August 2011

T.C.
UNIVERSITY OF GAZİANTEP
GRADUATE SCHOOL OF
NATURAL&APPLIED SCIENCES
ELECTRICAL&ELECTRONICS ENGINEERING

Name of the thesis : Remote Human Sensing and Conditioning of Data.

Name of the student: Talip ESKİKALE

Exam date : 11.08.2011

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Ramazan KOÇ
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Savaş UÇKUN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst.Prof.Dr. Tolgay KARA
Supervisor

Examining Committee Members Signature

1. Prof. Dr. L. Canan DÜLGER..
2. Prof. Dr. Arif NACAROĞLU..
3. Asst.Prof.Dr. Tolgay KARA..

ABSTRACT

REMOTE HUMAN SENSING and CONDITIONING OF DATA

Talip ESKİKALE

M.Sc. in Electrical & Electronics Eng.

Supervisor: Asst. Prof. Dr. Tolgay KARA

August 2011, (pages 80)

A disturbance is transmitted in a medium from one point to another by particle motion. The particles of the medium vibrate to transmit this energy from one particle to another. Persons moving over ground can be detected from vibrations induced to soil in the form of seismic waves, which are measured by geophones or accelerometers. Walking styles (standard, soft, or stealthy) and the background noise floor limit the detection range of footsteps. Walking style changes the dynamic footstep force on the ground and influences the footstep detection range.

Seismic sensors are capable of measuring pedestrian activity and are often employed for this task for a number of reasons. Such sensor systems are inexpensive, passive (do not dissipate energy), and potentially easily installed. In this thesis a system has been developed to alarm only when a pedestrian is present. Processing of the seismic signals should make it possible for the system to discriminate footsteps from other seismic sources such as animals, railroads, operating machinery, which is a challenging problem.

In this study, an algorithm has been developed to detect footsteps for security applications.

Key words: footstep detection, seismic signals, intrusion detection, perimeter security

ÖZET

UZAKTAN İNSAN ALGILAMA VE VERİLERİN İŞLENMESİ

Talip ESKİKALE
Yüksek Lisans Tezi, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Yr.Doç. Dr. Tolgay KARA
Ağustos 2011 (80 sayfa)

Sismik etkiler bir ortamdan diğerine parçacık hareketi ile iletilir. Ortama ait bu parçacıklar enerjiyi birbirine iletmek için titreşirler. Zemin üzerinde yürüyen kişiler jeofon ve ivme ölçerler ile sismik dalga olarak ürettikleri bu titreşimlerden tespit edilebilirler. Yürüme biçimleri (standart, yavaş ve gizli) ve dış etkenler adımların algılanmasını sınırlar. Yürüme biçimleri zemin üzerindeki dinamik adım kuvvetlerini değiştirir ve algılamayı zorlaştırır.

Sismik sensörler yaya hareketlerini ölçebilir. Bu tip sensörler ucuz, pasif (enerji tüketmezler) ve kolay kurulurlar. Bu tezde belirli bir alanda herhangi bir yaya aktivitesi tespit edildiğinde sistemin alarm vermesi amaçlanmaktadır. Fakat bu sistem hayvanlar, demir yolları ve iş makinelerinin yaydığı sismik aktivitelerden insan adımlarını ayırt edebilmelidir.

Bu çalışmada insan adımlarını tespit için bir algoritma geliştirilmiştir. Geliştirilen bu algoritma güvenlik uygulamalarında kullanılacaktır.

Anahtar Kelimeler: Adım tespiti, sismik işaretler, izinsiz giriş tespiti, çevre güvenliği

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor, Asst.Prof. Dr. Tolgay KARA for his advice and guidance in the preparation of this thesis.

Also, I would like to thank my wife Sibel for her grand support and patience.

Finally, I want to thank my parents.

TABLE OF CONTENTS

CHAPTER 4
HUMAN DETECTION SYSTEM DESIGN AND CONSTRUCTION

CHAPTER 5

EXPERIMENTS AND TEST RESULTS

CHAPTER 6

APPENDICES

APPENDIX A

APPENDIX B

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

The following nomenclature defines the principal symbols used in the thesis.

Symbols	**Description**
FFT	Fast Fourier Transform
STFT	Short Time Fourier Transform
FT	Fourier Transform
DFT	Discrete Fourier Transform
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
DAQ	Data Acquisition
TDMS	Technical Data Management - Streaming
MAT	Binary data container format used by MATLAB

To my wife Sibel, my son Bahadır and my parents,

CHAPTER 1

INTRODUCTION

1.1 Introduction

A footstep signature is caused by the impact on the ground. The ground is an elastic half space that supports waves that travel away from the point of impact. Each footstep has a characteristic shape that can be used to distinguish it from other noise.
One way to detect footsteps is to look for the periodic impact. The most striking feature of the footstep when comparing time series data for footsteps to other seismic signatures is the series of sharp "spikes" generated by each impact. This differs from the random noise.

The seismic background noise floor is much higher in urban areas and in buildings than in rural areas, dramatically influencing detection range. The average frequency of footstep impacts from a person walking or running is about 2 Hz [1]. The dynamic forces from footsteps that are normal to the ground/floor are the primary cause of the low-frequency component in these signals.

The aim of this thesis is to design and construct a system to detect human activity in some area. Probably the most obvious application is the detection of intruders in a secure region. It is often the case that a system capable of detecting pedestrians is desired. Any system to accomplish this task obviously needs a way to sense the pedestrian. Seismic sensors are capable of doing this task.

1.2 History of Footstep Detection

Methods of human detection utilizing low-frequency seismic signals (typically below a few hundred Hertz) from footsteps are well known in the literature and in practice. It is used for security systems designed to guard against intrusion of unauthorized personnel into a protected area.

One widely accepted method of footstep detection is based on the computation of kurtosis by Succi, which is a measure of statistical distribution used to detect extreme deviations from the mean (such seismic impulses due to footsteps)[2]. The kurtosis value is much higher in the presence of impulsive events than the presence of Gaussian or sinusoidal signatures. So it is much more sensitive to the signal generated by person footsteps than other signals generated by vehicles, noise,etc. Kurtosis values are calculated for various distributions in Figure1.1, Figure1.2, Figure 1.3 and Figure 1.4. If there are more and more x-values far from the mean, Kurtosis increases. Kurtosis is then a measure of how big the “tails” are.

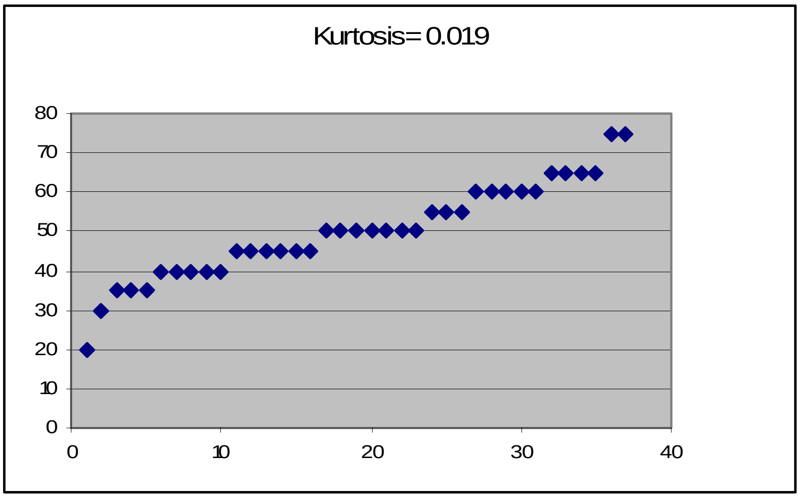

Figure 1.1 Kurtosis value is 0.019 for the samples of distribution

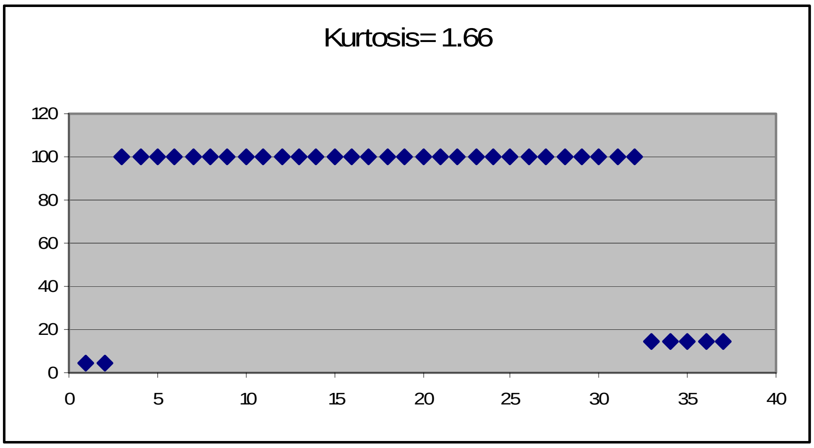

Figure 1.2 Kurtosis value is 1.66 for the samples of distribution

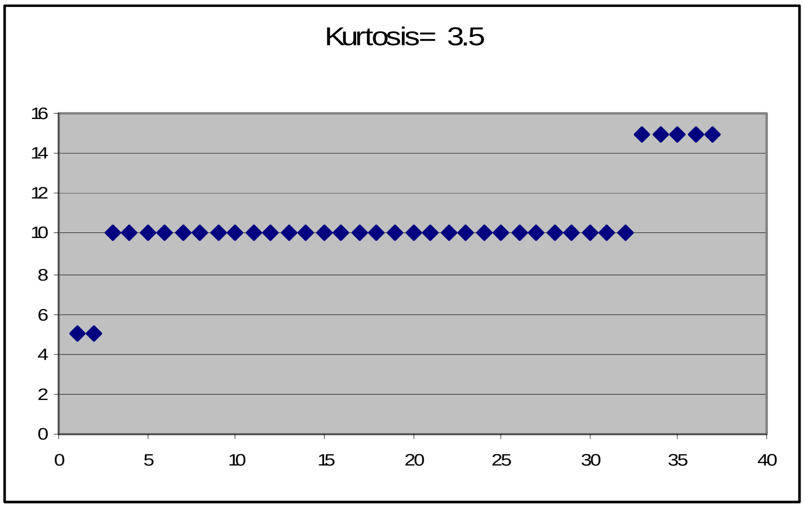

Figure 1.3 Kurtosis value is 3.5 for the samples of distribution

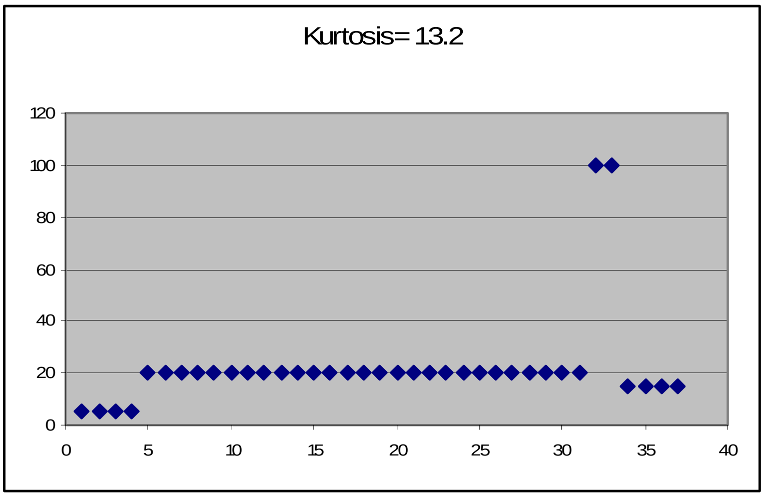

Figure 1.4 Kurtosis value is 13.2 for the samples of distribution

Another approach used by some investigators is to detect footsteps in the frequency domain by looking for the regular cadence of a typical human gait by Houston. Spectrum analysis approach is used in this approach[3]. Both of these methods are block-wise processing algorithms that look for a collection of footsteps prior to issuing an alert.

The baseline of the signal and the adaptive threshold levels above noise is defined by Mazarakis to detect footsteps[4]. In this approach, the adaptive and fixed threshold methods are used. Dibazar and Park proposed a dynamic synapse neural network on Footstep and Vehicle Recognition[5].

Both adaptive digital filtering and adaptive Kalman filtering methods are developed for Seismic Detection of Personnel by Chen. This study indicates that the self-adaptive noise-cancelling nonrecursive digital filter appears to enhance the impulsive nature of the footstep signals[6].

1.3 Organization of The Thesis

The rest of the thesis is organized as follows:

In chapter two, The fundamental information about seismic waves, sensors and signal processing is given. In chapter three, Footstep detection methods are explained. In chapter four, Programming of footstep detection is given with emphasis on the written program.. In chapter five, Experiments and test results are given. In the final chapter of the thesis, Conclusions of the study are presented.

CHAPTER 2

SEISMIC WAVES, SENSORS AND TOOLS

2.1 Seismic Waves

Seismic wave generated under the action of the short pulse is a complex wave that consists of the following components [7]:

- Longitudinal - compressive P-wave,
- Transverse S-wave,
- Rayleigh - Surface R-wave

Longitudinal P-waves and transverse S-waves are known as the body waves. Body waves are propagating through the medium by means of the hemispherical wavefront (Fig. 2.1).

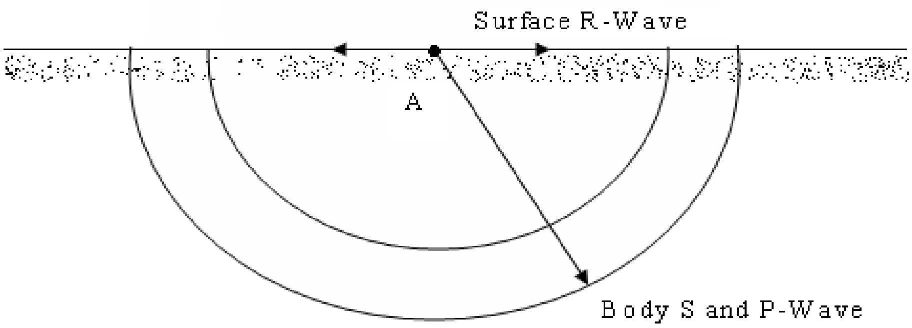

Figure 2.1 Propagation of the surface and body waves

The type of the component being considered depends on the source of vibrations. Rayleigh wave, which is propagated radially and has the cylinder-like wavefront, appears simultaneously with the body waves. Displacement of the ground in the vertical direction at the certain distance from the excitation point is illustrated in Figure 2.2.

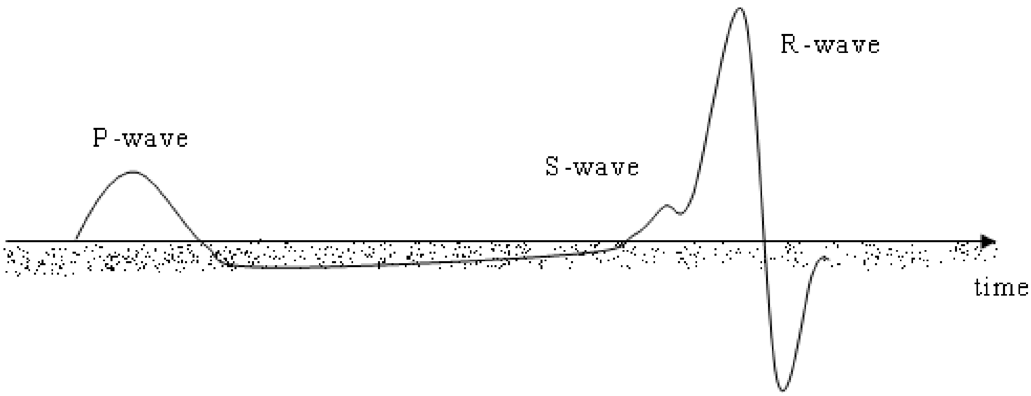

Figure 2.2 Vertical displacement of the medium particles on the surface

During the propagation through an elastic medium, components of the complex seismic wave have different velocities. Since the P-waves are faster than the other types of seismic waves, they can at first be detected by the sensors. P-waves are followed by the S-waves and Rayleigh - waves, respectively. As illustrated in Fig. 2.2, the vertical distance of the ground caused by the Rayleigh waves is greater than the distance caused by the remaining P and S-waves. The amplitude of the waves is considerably reduced with the increase of the distance from the source. The energy of the body waves is distributed through the medium according to the following expression[7]:

$$E \approx \frac{1}{r^2} \tag{2.1}$$

where E is the energy of surface density, and r is the radius of the sphere. The amplitude of the seismic wave is proportional to the square root of the energy surface density:

$$Amplitude \approx \sqrt{E} \tag{2.2}$$

$$Amplitude \approx \frac{1}{r} \tag{2.3}$$

Since the body waves are propagated through the semi-sphere only, the amplitude of the body waves is proportional to:

$$Amplitude \approx \frac{1}{r^2} \quad (2.4)$$

The amplitude of the Rayleigh waves is proportional to:

$$Amplitude \approx \frac{1}{\sqrt{r}} \quad (2.5)$$

where r is the radius of the cylinder. The attenuation of the Rayleigh waves is significantly less than that of the body waves. Rayleigh wave appears in the case of two adjacent elastic media with different elastic properties. This wave is similar to the wave generated by a stone thrown into the water. The velocity of the surface Rayleigh waves is given by[7]:

$$V_R = 0.9V_S \quad (2.6)$$

where Vs is the velocity of transverse waves in the same medium. Longitudinal waves propagate more slowly then direct transverse S-waves along the same trace, and even more slowly than direct longitudinal P-waves, so the following relation holds:

$$V_P > V_S > V_R \quad (2.7)$$

By the pulse excitation in the point A(x,y), Fig. 2.1, a surface seismic wave, moving at a constant velocity in the form of concentric circles, is generated. Geophones placed at different distances from the source point induce the presence of the wavefront. The medium vibrations take place in all three dimensions, but only the vibration of the medium particles in vertical direction is used for measurement of the seismic wave velocity. By detecting the vertical displacement of the medium particles in time, the vibration curves are obtained[7].

Vehicles and personnels can be detected using a three-component seismic velocity transducer. Persons or vehicles moving on the ground generate a succession of impacts; these soil disturbances propagate away from the source as seismic waves. Because the soil is an elastic medium, both vertical and longitudinal waves propagate, diminishing in intensity as R^{-2}. Furthermore, because the surface of the soil is the boundary of an elastic space, a Rayleigh surface wave is also generated, diminishing in intensity as R^{-1}. This surface wave is a vector wave that can be used to track the source.

2.2 Geophone

A geophone is a single axis seismometer that measures motion in the direction of its cylindrical axis[8]. In typical near-surface deployments, a geophone is packaged with a conical spike and buried a few inches underground to ensure good coupling to the motion of the Earth. Ground motion causes the hollow cylinder of a geophone to move with respect to the geophone housing. The motion of this cylinder inside the geophone is described by Equation 2.8. It is the transfer function of a second-order mechanical system. Equation 2.8 expresses the relative position of the proof mass, Xr, for the acceleration applied to a geophone, $\ddot{X}_h$, as a function of frequency, with mass m[kg], spring constant k[N/m], and damping constant b[N/(m/s)].

$$X_r = \left(\frac{-1}{s^2 + \frac{b}{m}s + \frac{k}{m}} \right) . \ddot{X}_h \tag{2.8}$$

The cylinder's motion is measured by the interaction of the coil on the cylinder with the magnetic field of the permanent magnet inside the geophone. Faraday's Law, expressed in Equation 2.9 in time and frequency domains, states that the voltage across a coil is equal to the change in flux through the coil with respect to time. In the case of a geophone, the change in flux through the coil versus coil displacement, $\frac{\partial \phi}{\partial x}$, is constant for small displacements. Therefore, the voltage across the coil is directly proportional to the velocity of the coil. Geophone manufacturers typically report the constant of proportionality, G[V/(m/s) = N/A], known as the transduction constant or generator constant[8]. It is shown that G varies by less than 0.05% as a function of position for displacements on the order of 10% of the maximum displacement[8].

$$V_o = -\frac{\partial \phi}{\partial t} = -\frac{\partial \phi}{\partial x}\frac{\partial x}{\partial t} = -G\dot{X}_r = -GsX_r \tag{2.9}$$

The transfer function relating output voltage to input acceleration, given in Equation 2.10, can be determined by combining Equation 2.8 and Equation 2,9.

$$V_o = \left(\frac{Gs}{s^2 + \frac{b}{m}s + \frac{k}{m}} \right) . \dot{X}_h \qquad (2.10)$$

In the present work, GS-20DX Geophone is used for collecting signals from intruders. One of the important characteristics of this geophone is bandwidth (8-1500 Hz). Its characteristic information is given in Appendix A.

2.3 Windowing Technique

FFT based measurements are subject to errors from an effect known as leakage. This effect occurs when the FFT is computed from a block of data, which is not periodic. To correct this problem appropriate windowing functions must be applied. The user must choose the appropriate window function for the specific application. When windowing is not applied correctly, then errors may be introduced in the FFT amplitude, frequency or overall shape of the spectrum[9].

The FFT computation assumes that a signal is periodic in each data block, that is, it repeats over and over again and it is identical every time. Figure 2.3 illustrates the FFT of a periodic signal. The matlab codes are given in Appendix C. When the FFT of a non-periodic signal is computed, the resulting frequency spectrum suffers from leakage. Leakage results in the signal energy smearing out over a wide frequency range in the FFT when it should be in a narrow frequency range. Figure 2.4 illustrates the effect of leakage. If the frequency spectrums of the two sample waves are compared, they will be differed greatly. The extra energy around 15 Hz is refered to the spectral leakage.

Figure 2.3 shows a 15 Hz sine wave with amplitude 4 that is periodic in the time frame. The resulting FFT (bottom) shows a narrow peak at 15 Hz in the frequency axis with a height of 2 as expected. Note the dB scale is used to highlight the shape of the FFT at low levels. Figure 2.4 shows a sine wave that is not periodic in the time frame resulting in leakage in the FFT (bottom). The amplitude is less than the expected 2 value and the signal energy is more dispersed. The dispersed shape of the FFT makes it more difficult to identify the frequency content of the measured signal.

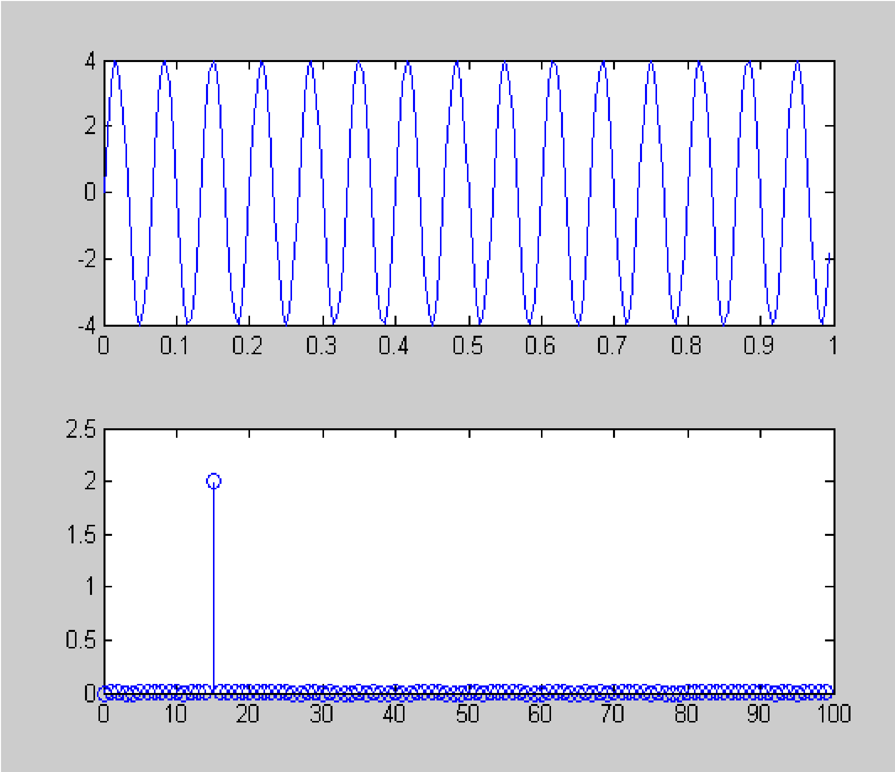

Figure 2.3 FFT of a periodic signal

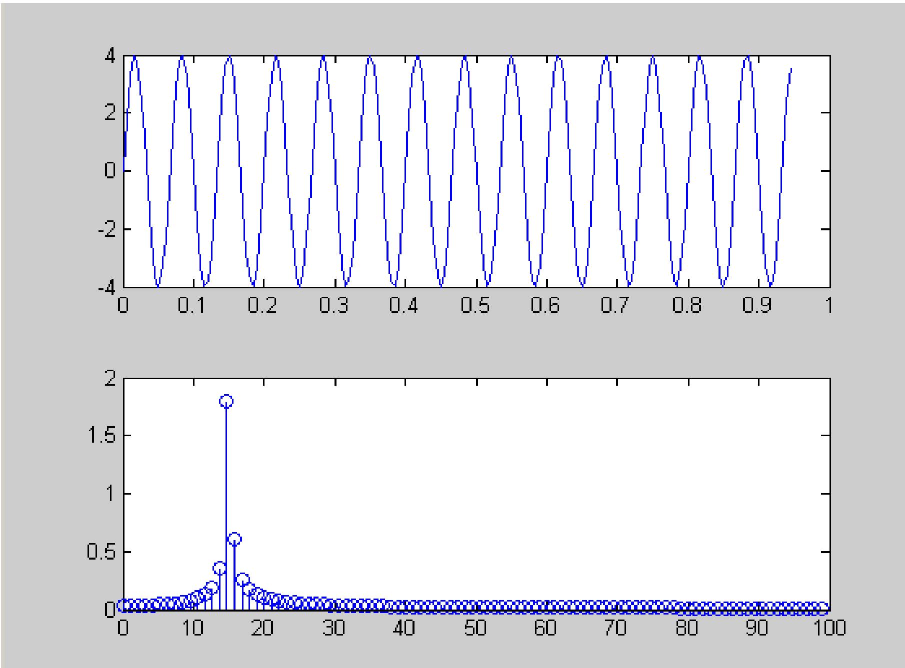

Figure 2.4 FFT of a non-periodic signal

Since most signals are not periodic in the predefined data block time periods, a window must be applied to correct the leakage. A window is shaped so that it is exactly zero at the beginning and end of the data block and has some special shape. This function is then multiplied with the time data block forcing the signal to be periodic. A Hanning window in Figure 2.5 can be used for this purpose. A special weighting factor must also be applied so that the correct FFT signal amplitude level is recovered after the windowing.

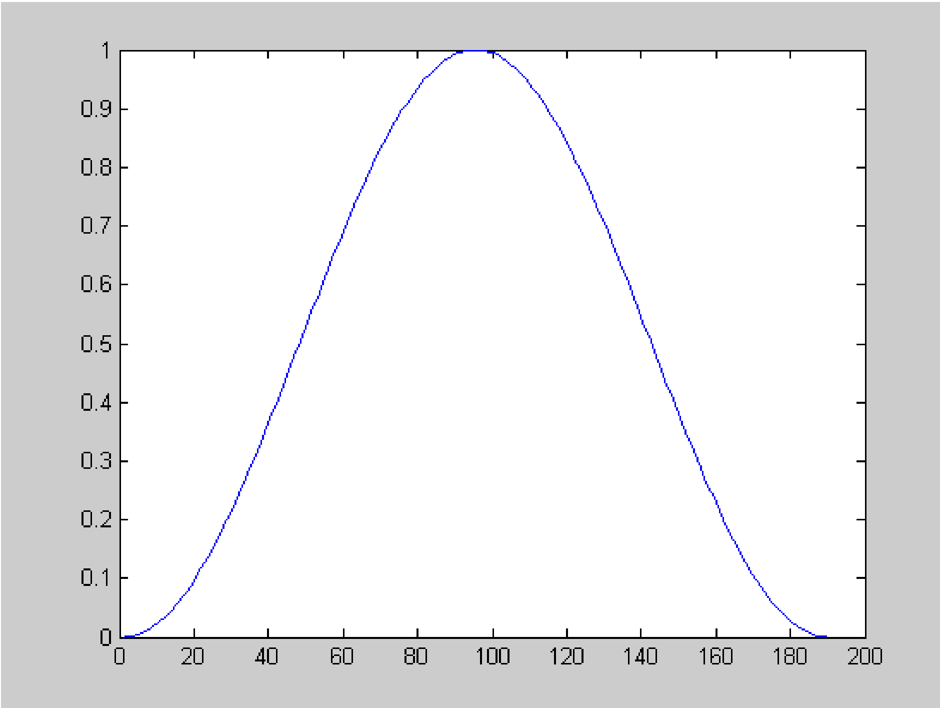

Figure 2.5 A Hanning Window

The 15 Hz sine signal in Figure 2.3 is multiplied by a Hanning window, then the result signal is represented in Figure 2.6. Frequency spectrums of windowed and non-windowed sine signals are represented in Figure 2.7. The Matlab codes are given in Appendix C.

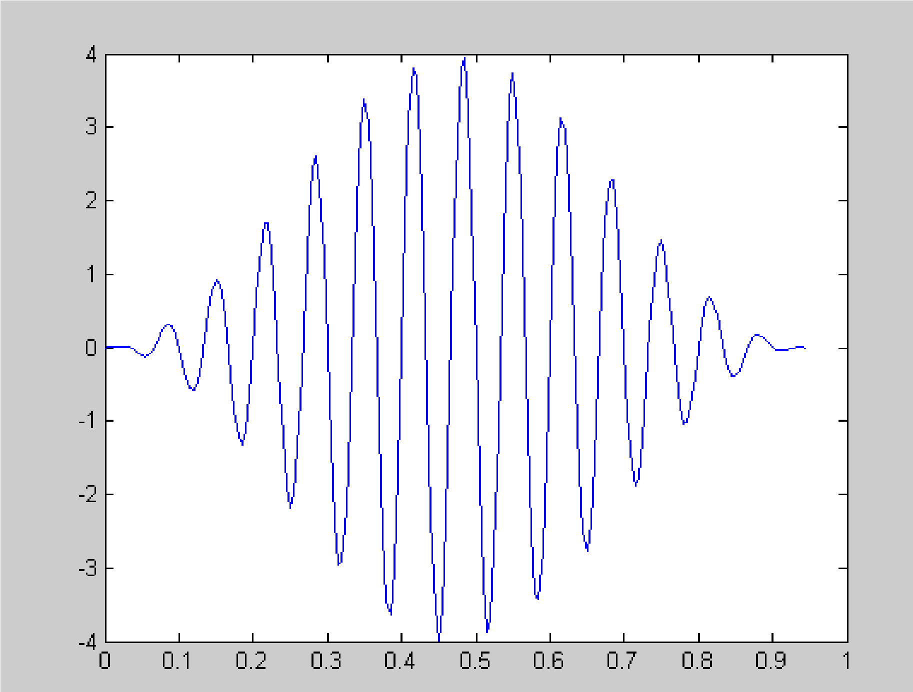

Figure 2.6 Windowed Sine wave

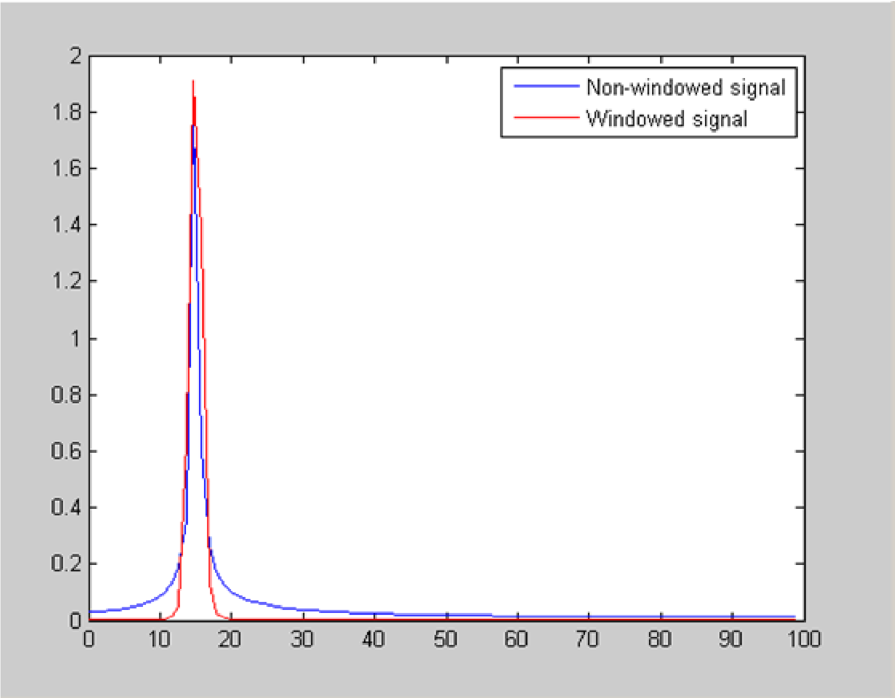

Figure 2.7 Frequency spectrum

2.4 Short Time Fourier Transform

The Fourier transforms (FT,DFT, etc) do not clearly indicate how the frequency content of a signal changes over time [10]. That information is hidden in the phase. It is not revealed by the plot of the magnitude of the spectrum. To see how the frequency content of a signal changes over time, we can cut the signal into blocks and compute the spectrum of each block.

To improve the results,

- blocks are overlapped
- each block is multiplied by a window that is tapered at its endpoints.

Several parameters must be chosen:

- Block length, R
- The type of window
- Amount of overlap between blocks.

Figure 2.8 shows no overlap between blocks. Figure 2.9 shows R/4 overlap between blocks and finally Figure 2.10 shows R/2 overlap between blocks.

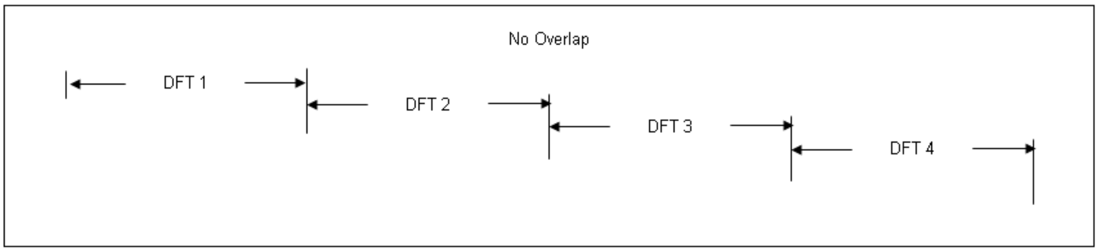

Figure 2.8 No Overlap between blocks

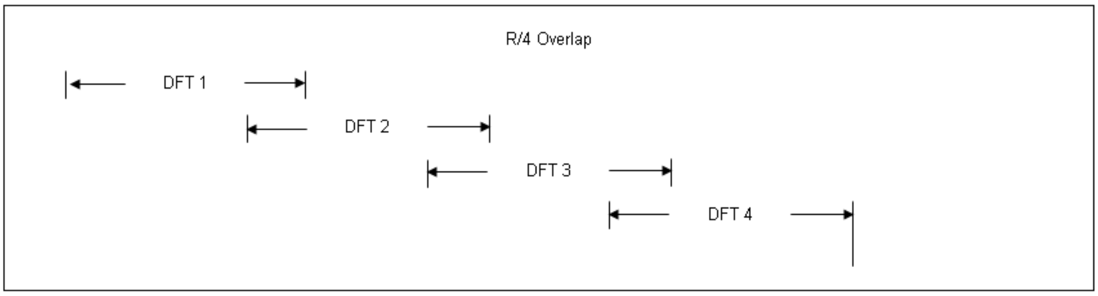

Figure 2.9 R/4 Overlap between blocks

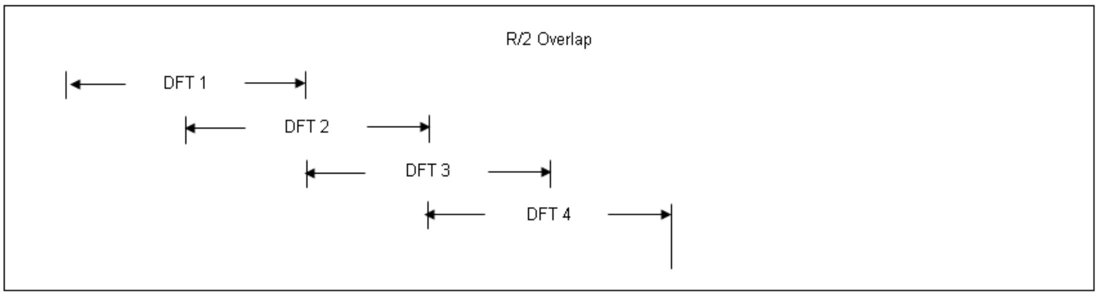

Figure 2.10 R/2 Overlap between blocks

L is the number of samples between adjacent blocks. It can be seen from Figure 2.11. Also L does not affect the time resolution or the frequency resolution.

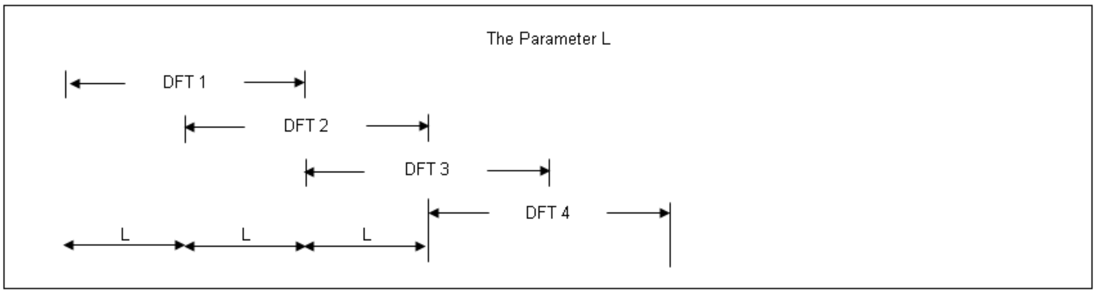

Figure 2.11 Parameter L

The short time Fourier transform is defined as

$$
\begin{aligned}
X(w,m) = (STFT(x(n)) &:= DTFT(x(n-m)w(n) \\
&= \sum_{n=-\infty}^{\infty}\left(x(n-m)w(n)e^{-(iwn)}\right) \\
&= \sum_{n=0}^{R-1}\left(x(n-m)w(n)e^{-(iwn)}\right) \qquad (2.11)
\end{aligned}
$$

where w(n) is the window function of length R[10]. The STFT of a signal x(n) is a function of two variables: time and frequency. The block length is determined by the support of the window function w(n). A graphical display of the magnitude of the STFT, │X(w,m)│, is called the spectrogram of the signal. The STFT of a signal is invertible.

One can choose the block length. A long block length will provide higher frequency resolution because the main-lobe of the window function will be narrow. A short block length will provide higher time resolution because less averaging across samples is performed for each STFT value. A narrow-band spectrogram is one computed using a relatively long block length R, (long window function). A wide-band spectrogram is one computed using a relatively short block R, (short window function).

To numerically evaluate the STFT, the frequency axis w is sampled in N equally spaced samples from w=0 to w=2π.

$$\forall k,\, 0 \le k \le N-1 : \left(w_k = \frac{2\pi}{N} k \right) \tag{2.12}$$

The discrete STFT is given by,

$$\begin{aligned} \left(X^d(k,m) := X(\frac{2\pi}{N} k, m) \right) &= \sum_{n=0}^{R-1} \left(x(n-m) w(n) e^{-(iwn)} \right) \\ &= \sum_{n=0}^{R-1} \left(x(n-m) w(n) W_N^{-(kn)} \right) \\ &= DFT_N \left(x(n-m) w(n) \Big|_{n=0}^{R-1}, 0, \ldots 0 \right) \end{aligned} \tag{2.13}$$

where 0,…0 is N-R.

In this definition, the overlap between adjacent blocks is R-1. The signal is shifted along the window one sample at a time. That generates more points than is usually needed, so the STFT is also sampled along the time direction. That means

$$X^d(k, Lm)$$

is evaluated, where L is the time-skip. The relation between the time-skip, the number of overlapping samples, and the block length is

$$\text{Overlap}=\text{R-L} \tag{2.14}$$

Consequently, the Short Time Fourier transform is used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. The Short Time Fourier transform tells the order of the frequencies used in the signal.

2.5 Digital Filtering

The purpose of a filter is to accept certain type of signals and reject others. A filter can be characterized by its frequency response as low pass, high pass, band stop, and band pass. Both the FIR and IIR difference equations can be utilized to implement digital filters. This is achieved by choosing the appropriate coefficients. From implementation perspective, the IIR filter is more appropriate since the filter size is smaller compared to the FIR filter to achieve similar frequency response. Unlike the IIR, the FIR has linear phase that is necessary to prevent phase distortion.

CHAPTER 3

FOOTSTEP DETECTION METHODS

3.1 Kurtosis

Pearson introduced kurtosis in 1905 as a measure of how flat the top of a symmetric distribution is when compared to a normal distribution of the same variance. Kurtosis is more influenced by scores in the tails of the distribution than scores in the center of a distribution [11].

Kurtosis can be defined as the standardized fourth population moment about mean,

$$\beta_2 = \frac{E(X-\mu)^4}{\left(E(X-\mu)^2\right)^2} = \frac{\mu_4}{\sigma^4} \tag{3.1}$$

where E is the expectation operator, μ is the mean, μ_4 is the fourth moment about the mean, and σ is the standard deviation. The normal distribution has a kurtosis of 3, and β_2-3 is often used so that the reference normal distribution has a kurtosis of zero(β_2-3 is sometimes denoted as γ_2). A sample counterpart to β_2 can be obtained by replacing the population moments with the sample moments, which gives

$$b_2 = \frac{\sum (X_i - \overline{X})^4 / n}{\left(\sum (X_i - \overline{X})^2 / n\right)^2} \tag{3.2}$$

where b_2 is the sample kurtosis, $\overline{X}$ is the sample mean, and n is the number of observations [12]. The kurtosis value compared for a sample sequence is much higher in the presence of impulsive events than it is in the presence of Gaussian or Sinusoidal signatures. The method depends only on the shape of the signature and not the amplitude. Figure 3.1 shows positive and negative kurtosis, if β_2-3 is greater than zero then it is positive kurtosis (leptokurtic). , if β_2-3 is less than zero then it is negative kurtosis (platykurtic) [13].

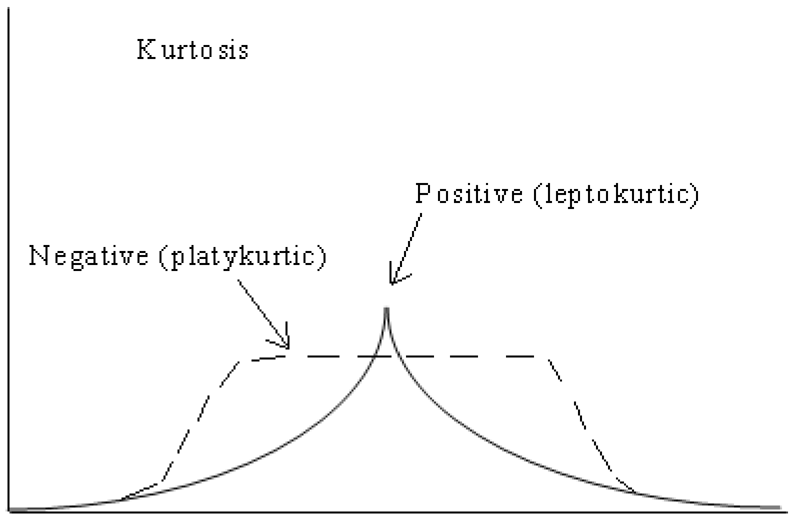

Figure 3.1 Positive and negative kurtosis

The dotted lines show normal distributions in Figure 3.2, the solid lines show distributions with positive kurtosis (left) and negative kurtosis (right). As can be seen from the Figure 3.1, Positive kurtosis has heavier tails and higher peak than the normal, whereas negative kurtosis has lighter tails and flatter peak.

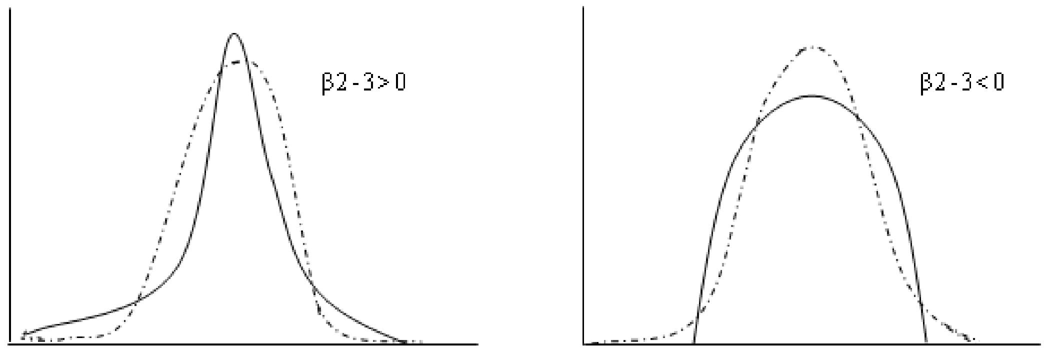

Figure 3.2 Positive and negative kurtosis with normal distributions

Kurtosis is used for footstep detection and is a statistical measure of the amplitude of the seismic signature [14].

3.2 Spectrum Analysis

A spectrogram is a time-varying spectral representation that shows how the spectral density of a signal varies with time. The purpose of the spectrogram is to analyze the signal using a sliding window. The window length is chosen so that the signal may be considered to be almost stationary inside [15]. In this case, its spectral energy density can be evaluated using the Fourier transform over each time interval obtained by shifting the sliding window. The Short time Fourier transform or its square magnitude (spectrogram) consider therefore a non-stationary signal as a concatenation of stationary signals within the sliding window. Thus the time resolution of this analysis is given by the window size, while the spectral resolution is proportional to its inverse. This means that it is not possible to increase the two resolutions simultaneously.

For highly non-stationary signals a fine time resolution is required, thus, window should be short in this case and the spectral resolution will be low. If a fine spectral resolution is required, window should be large, which reduces the time resolution. The spectrogram is used to detect the Footsteps[3].

3.3 Envelope

The envelope of a signal is the outline of the signal. Envelope detection has numerous applications in Signal Processing and Communications, including amplitude modulation (AM) detection[16].

3.3.1 Envelope of a Signal

The used method for envelope of a signal works by squaring the input signal and sending it through a low-pass filter. Squaring the signal effectively demodulates the input by using itself as the carrier wave. This means that half the energy of the signal is pushed up to higher frequencies and half is shifted towards DC. The envelope can then be extracted by keeping all the DC low-frequency energy and eliminating the high-frequency energy.

3.3.2 FIR Decimation

The FIR Decimation block resamples the discrete-time input at a rate K times slower than the input sample rate, where the integer K is specified by the Decimation factor parameter[17]. This process consists of two steps:

- The block filters the input data using a direct-form FIR filter.
- The block downsamples the filtered data to a lower rate by discarding K-1 consecutive samples following every sample retained.

The FIR Decimation block implements the above FIR filtering and downsampling steps together using a polyphase filter structure, which is more efficient than straightforward filter-then-decimate algorithms. In Figure 3.3, Seismic data is imported to simulink from Matlab workspace. It is filtered, then applied to Envelope detector by squaring the signal and low pass filtering. Finally output signal exported to Matlab workspace.

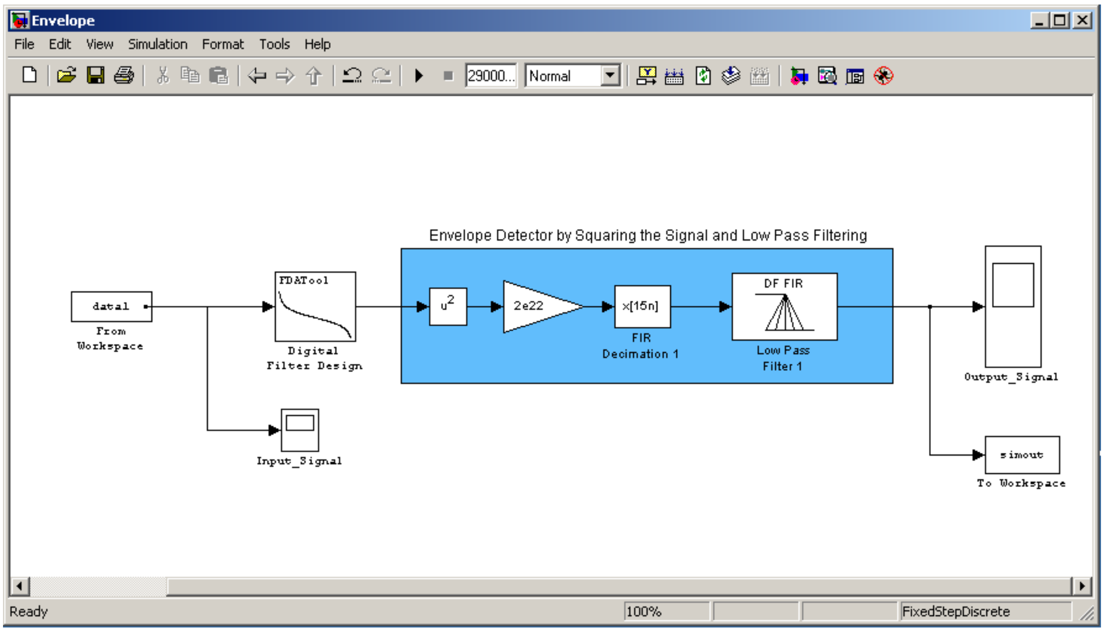

Figure 3.3 Simulink model of the system for envelope detection

3.3.3 Block Diagram of The System

The designed footstep detection system consists of two parts, hardware and software The hardware of the system is made up of geophone, data acquisition card and cabling. The software of the system is data acquisition, data conversion, filtering, envelope of the signal and the detection algorithm.

The block diagram of the designed system can be seen in Figure 3.4.

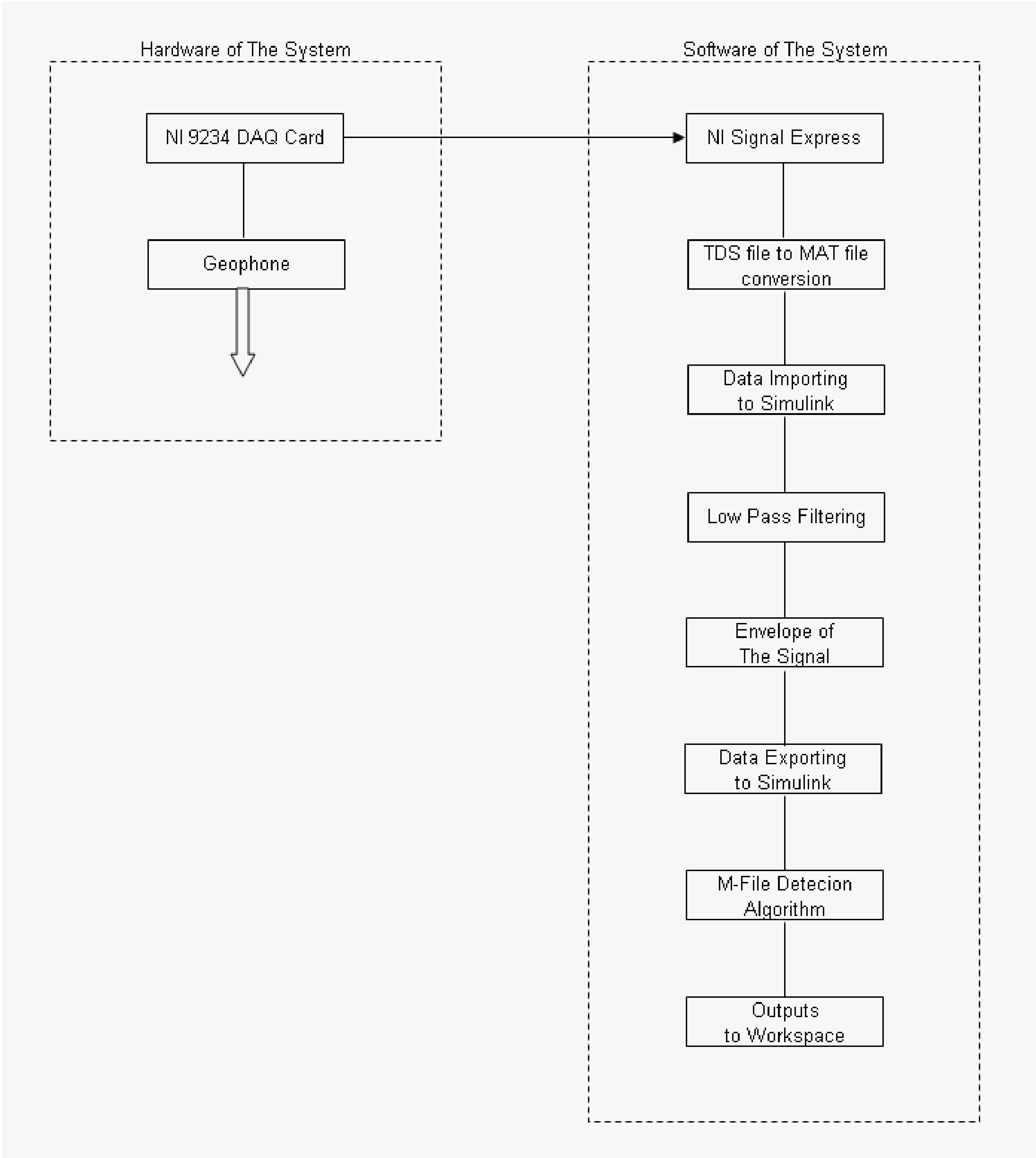

Figure 3.4 Block diagram of the system

CHAPTER 4

HUMAN DETECTION SYSTEM DESIGN AND CONSTRUCTION

4.1 System Built up

Vertical axis Geospace GS-20DX geophone is buried in the ground. It is connected to the National Instruments NI-9234 data acquisition card. This card has USB connection to a PC. After hardware installation, as a person walked along the path, seismic data was recorded via PC from geophone sensor. The system is shown in Figure 4.1

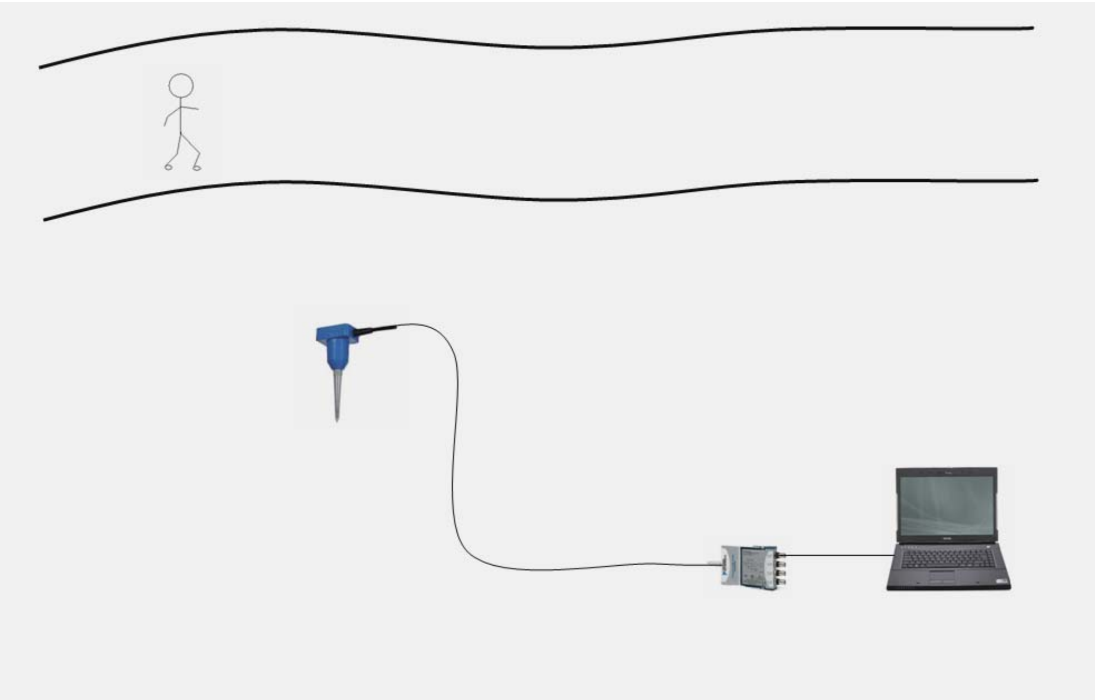

Figure 4.1 The test layout used to collect Human Seismic Data

4.2 Seismic Data Acquisition

NI Signal express program is used for data acquisition. It saves the collected data in the format of a tdms file extension. In this thesis, Matlab program is used for all calculations. Normally, a tdms file is not recognized by Matlab. A special m-file is used for converting a tdms file to mat file, which is given in Appendix D

4.2.1. Data Acquisition with NI Signal Express

Before data acquisition, setup settings must be adjusted. First of all, an "add step" is created, then the configuration parameters of NI-9234 DAQ card is adjusted in the opened window in Figure 4.2.

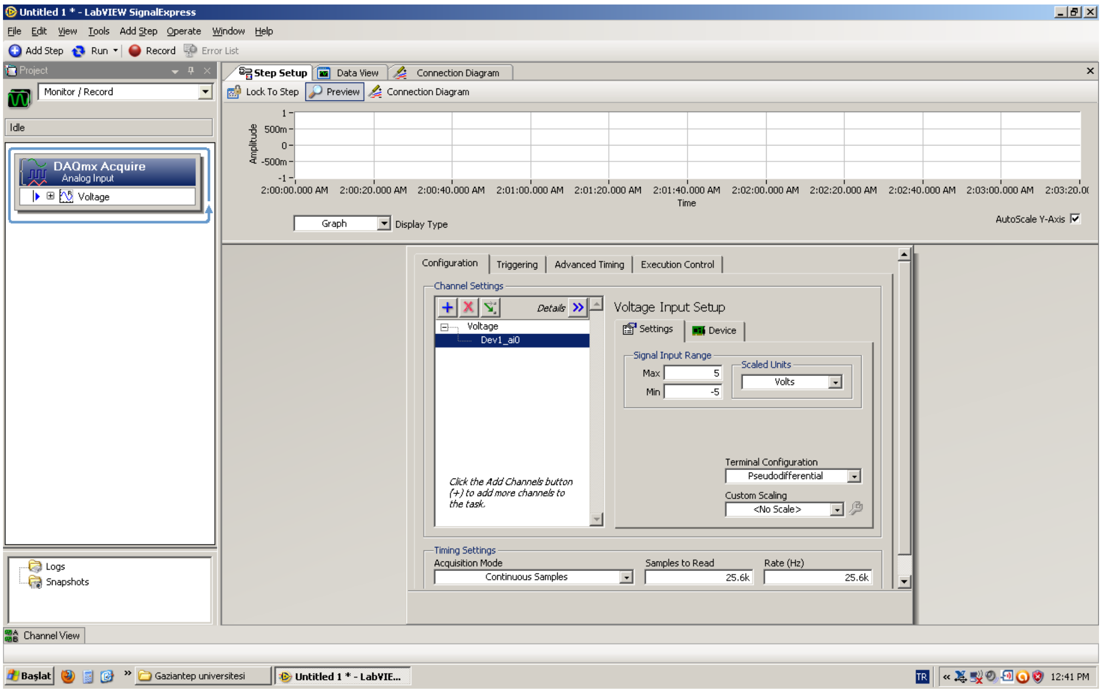

Figure 4.2 NI Signal Express signal setup

Adjusted parameters are

- Channel is selected Analog input 0, (ai0)
- Signal range is "-5 volt to +5 volt" .
- Acquisition mode is "continuous samples" and
- Sample rate is adjusted 25.6 kS/sec.

After the configuration is completed, the seismic data is collected from geophone. Now, the run button is used to start acquisition, In Figure 4.3, the seismic signal that generated by geophone can be seen. Each peaks are footsteps of a person. When a person comes close to the geophone, the peak amplitude is increased.

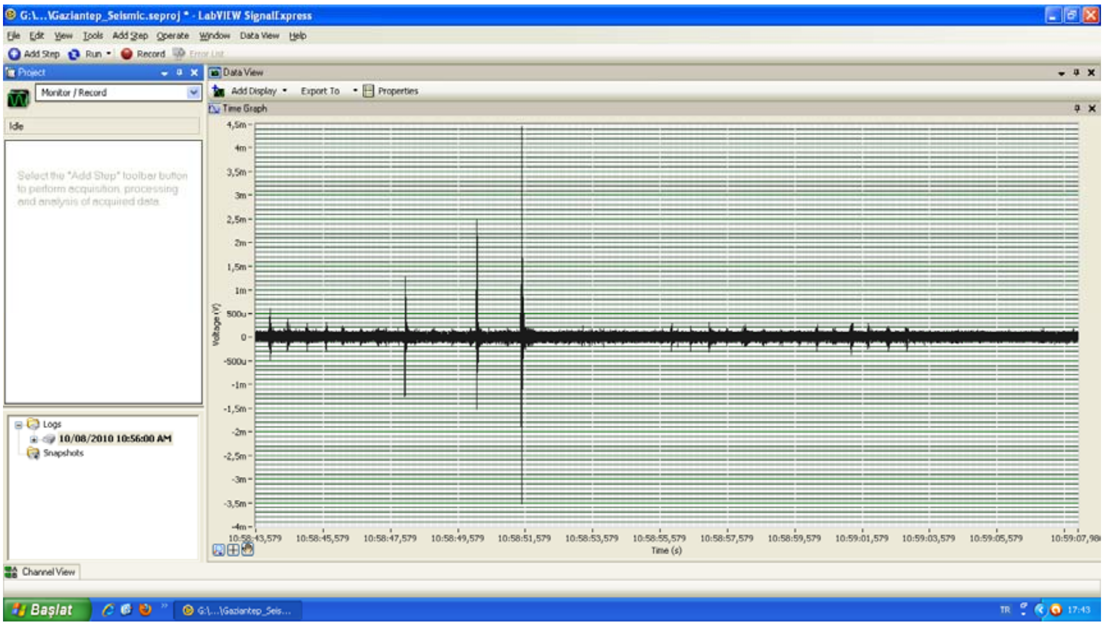

Figure 4.3 Seismic Data

4.2.2. Transformation of Data into Matlab Platform

In this thesis, Matlab R2006a with data acquisition toolbox 2.8.1 is used. It is not compatible with NI-9234 card. This DAQ card is compatible with Matlab 2009a with Data acquisition toolbox 2.14. Therefore a convertion is necessary from tdms-file to mat-file.

To convert a tdms-file, convertTDMS.m file is copied to the folder C:\Program Files\MATLAB\R2006a\work. Then the command “convertTDMS(0)” in the Matlab command window is used as can be seen from the Figure 4.4

```
>> convertTDMS(0)
```

After applied “convertTDMS(0)” command, tdms file is converted to mat-file by Matlab. A variable that named as ans is created in the structure format. It is represented in Figure 4.5

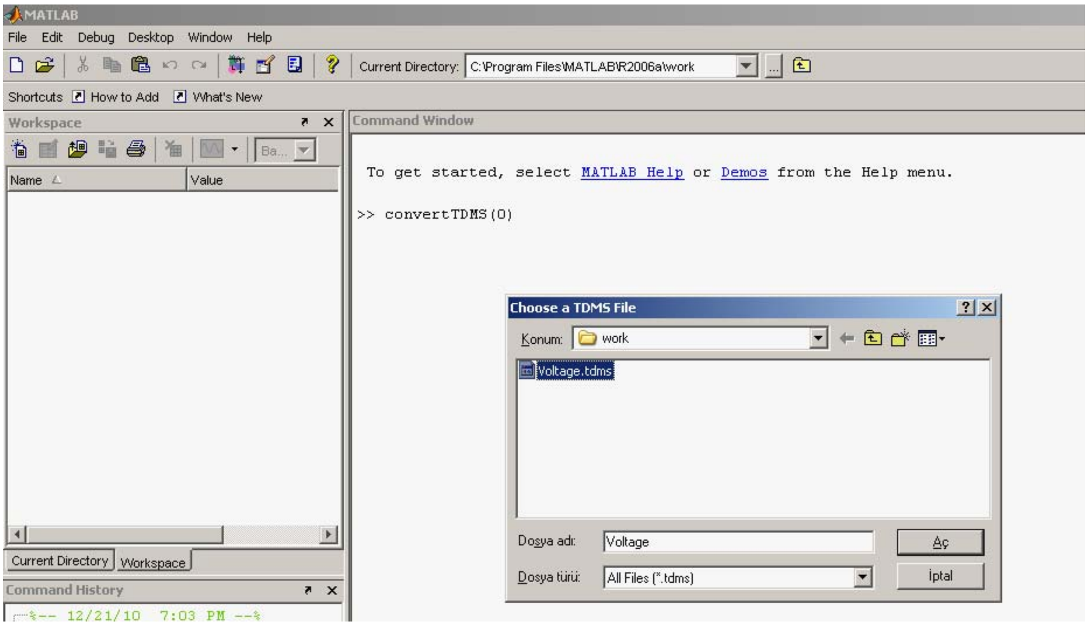

Figure 4.4 The selection of data to be converted

Finally The codes in Table 4.1 are used to obtain the data as a structure.

Table 4.1 Algorithm for obtaining a structure type data

```
B=ans.Data.MeasuredData(1,1);
sis=B.Data(:,1);
data1.signals.values=sis;
data1.time=[];
```

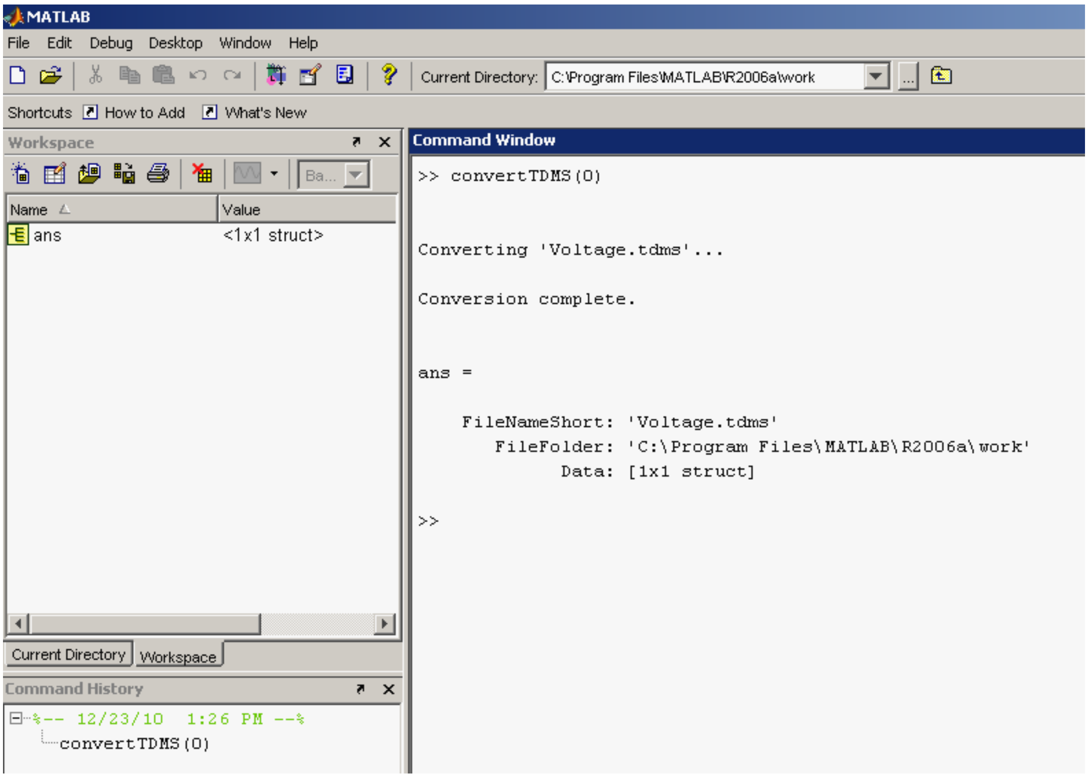

Figure 4.5 Data Conversion tdms-file to mat-file

4.2.3. Generation of Disturbance Signals

The sources of disturbances in seismic sensors are external noises and measurement noises. To simulate noise, sine functions that have different frequencies and different time values are created.

data1 is raw seismic signal at previous section, here disturbances as data2, data3 and data4 are created. data2 is 500 Hz sinusoid signal. The codes in Table 4.2 are given for obtaining a 500 Hz sinusoid signal.

Table 4.2 Algorithm for obtaining a 500 Hz sinusoid signal.

```
Fs1=25600;
t1 =0:1/Fs1:16-1/Fs1;
x1=6e-4*sin(2*pi*500*t1);
x1= x1';
data2.signals.values=x1;
data2.time=[];
```

data3 is 1000 Hz sinusoid signal. The codes are given in Table 4.3

Table 4.3 Algorithm for obtaining a 1000 Hz sinusoid signal.

```
Fs2=25600;
t2 =0:1/Fs2:16-1/Fs2;
x2=6e-4*sin(2*pi*1000*t2);
x2= x2';
data3.signals.values=x2;
data3.time=[];
```

data4 is 2000 Hz sinusoid signal for first 4 seconds and 4000 Hz sinusoid signal for remaining 12 seconds. The codes are given in Table 4.4

Table 4.4 Algorithm for obtaining a 2000 Hz and 4000 Hz sinusoid signal.

```
Fs3=25600; %102400=4 sec
Fs4=25600; %307200=12 sec
t3 =0:1/Fs3:4-1/Fs3;
t4 =0:1/Fs4:12-1/Fs4;
x3=6e-4*sin(2*pi*2000*t3);
x4=6e-4*sin(2*pi*4000*t4);
x34=[x3 x4];
x34= x34';
data4.signals.values=x34;
data4.time=[];
```

Finally, all data are combined as one signal “xTotal” and the code is

xTotal=sis+x1+x2+x34;

4.3 Simulink Model

A simulink model is designed to eliminate frequencies out of 2-100 Hz and to generate the envelope of the signal. Then output of the model is recorded and sent to the workspace. The developed model is shown in the Figure 4.6

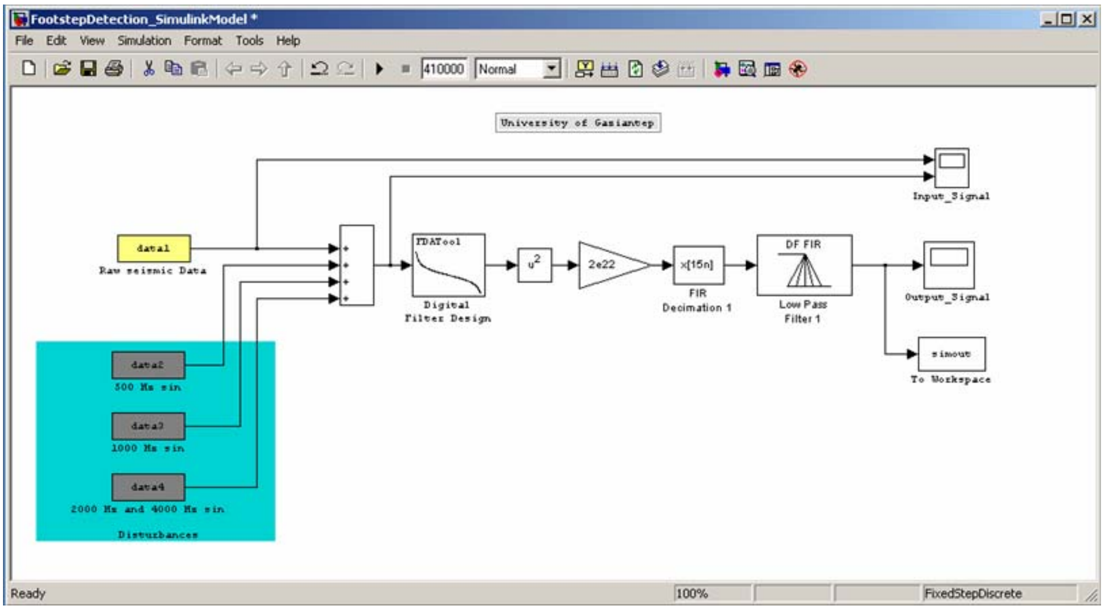

Figure 4.6 Simulink Model

In the model, first a raw data and disturbances are generated in Matlab. Then they are combined to see the performance of the model in the Simulink. It sent to a band-pass filter between 2 and 100 hz. Here low and high frequencies eliminated. Finally envelope of the signal is obtained by squaring the input signal and sending it through a low-pass filter. Squaring the signal effectively demodulates the input by using itself as the carrier wave. This means that half the energy of the signal is pushed up to higher frequencies and half is shifted towards DC. The envelope can then be extracted by keeping all the DC low-frequency energy and eliminating the high-frequency energy. A simple minimum-phase low-pass filter is used to get rid of the high-frequency energy [16].

The inputs and output of the model are shown in Figure 4.7. Highly disturbed signals are filtered and an envelope signal is generated. As can be seen from the Figure 4.7 each footstep seems as a peak and time between footsteps are nearly same (periodic).

When intruder comes close to the geophone sensor, the amplitude of the signal will be increased.

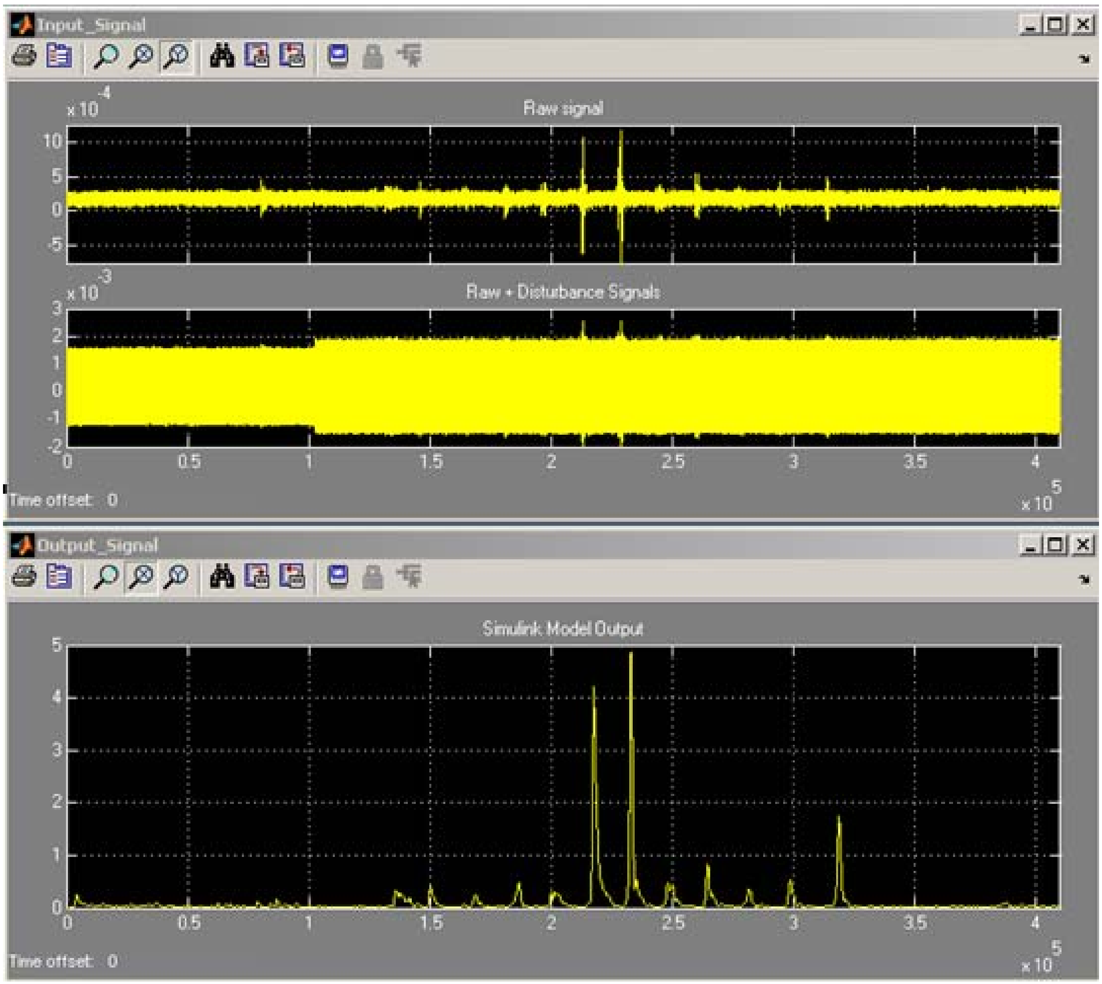

Figure 4.7 Simulink model inputs and outputs

4.4 Footstep Detection Algorithms

Two method are used to examine footsteps. First is spectrogram, other our detection algorithm based on filtering and envelope.

4.4.1 Spectrogram

A spectrogram is a time-varying spectral representation that shows how the spectral density of a signal varies with time. The spectrogram of a signal can be estimated by computing the squared magnitude of the STFT of the signal[15]. Figure 4.8 shows the spectrogram of the raw seismic signal. Here footsteps can be seen as time and frequency.

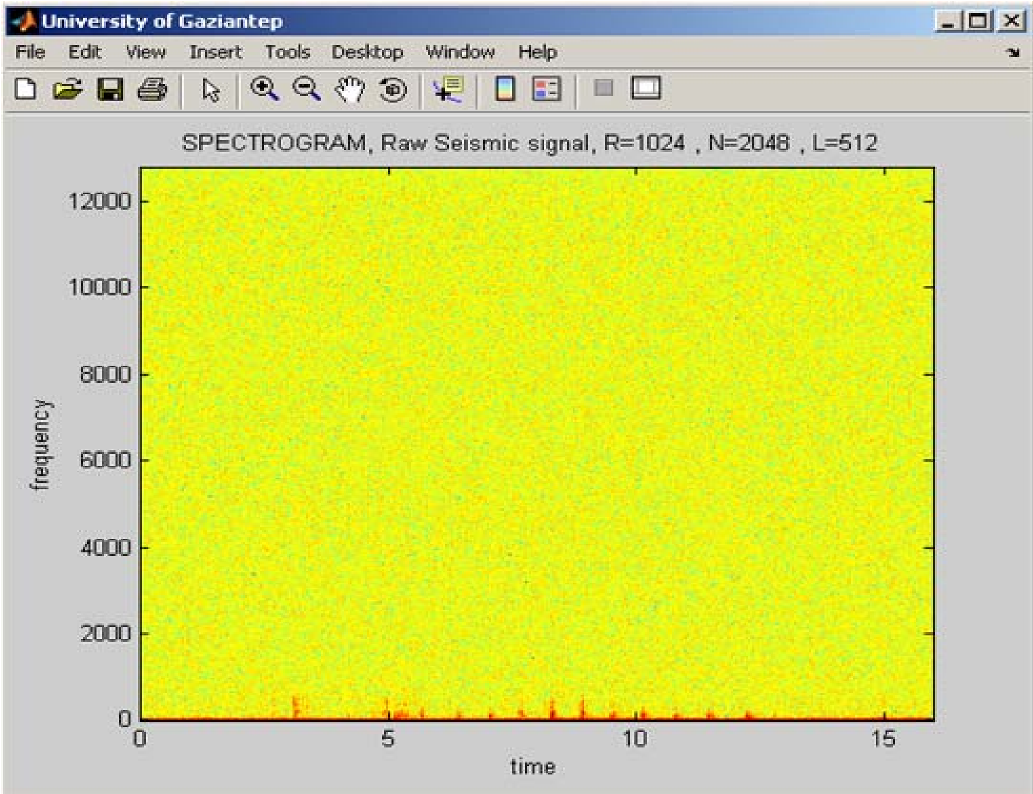

Figure 4.8 Spectrogram of the raw seismic signal

Spectrogram is applied to the raw seismic signal. Block length,R is set to 1024. Frequency discretization, N is set to 2048. Time lapse between blocks, L is 512 and Sampling frequency, fs is 25600. The Matlab codes are given in Table 4.5.

Table 4.5 Algorithm for obtaining the spectrogram of the footstep signals

```
R = 1024;
window = hamming(R);
N =2048;
L = 512;
fs = 25600;
overlap = R - L;
[B,f,t] = specgram(sis,N,fs,window,overlap);
% MAKE PLOT
figure('Name','University of Gaziantep','NumberTitle','off'), clf
imagesc(t,f,log10(abs(B)));
colormap('jet')
axis xy
xlabel('time')
ylabel('frequency')
title('SPECTROGRAM, Raw Seismic signal, R=1024 , N=2048 , L=512 ')
```

Now lets apply some noise to the raw seismic signal and see how spectrogram will change. Disturbance signals that generated before are used. Raw signal and those disturbance signals are combined as one signal. Then Spectrogram is applied to the signal. Block length,R is set to 1024. Frequency discretization,N is set to 2048. Time lapse between blocks,L is 512 and Sampling frequency, fs is 25600. The Matlab codes are given in Table 4.6.

Table 4.6 Algorithm for obtaining the spectrogram of the combined signals

```
R = 1024;
window = hamming(R);
N =2048;
L = 512;
fs = 25600;
overlap = R - L;
[B,f,t] = specgram(xTotal,N,fs,window,overlap);
% MAKE PLOT
figure('Name','University of Gaziantep','NumberTitle','off'), clf
imagesc(t,f,log10(abs(B)));
colormap('jet')
axis xy
xlabel('time')
ylabel('frequency')
title('SPECTROGRAM, Total signal=Raw+500Hz+1000+2000Hz+4000Hz sine ')
```

Figure 4.9 shows the spectrogram of the raw seismic and disturbance signal. Footsteps and disturbances can be seen as time and frequency.

Spectrogram is very useful for examining the frequency contents of the signal as time. From the spectrogram, Footsteps have low frequency (10-50 Hz) and periodic peaks. These peaks in amplitudes change as distance between the sensor and the pedestrian.

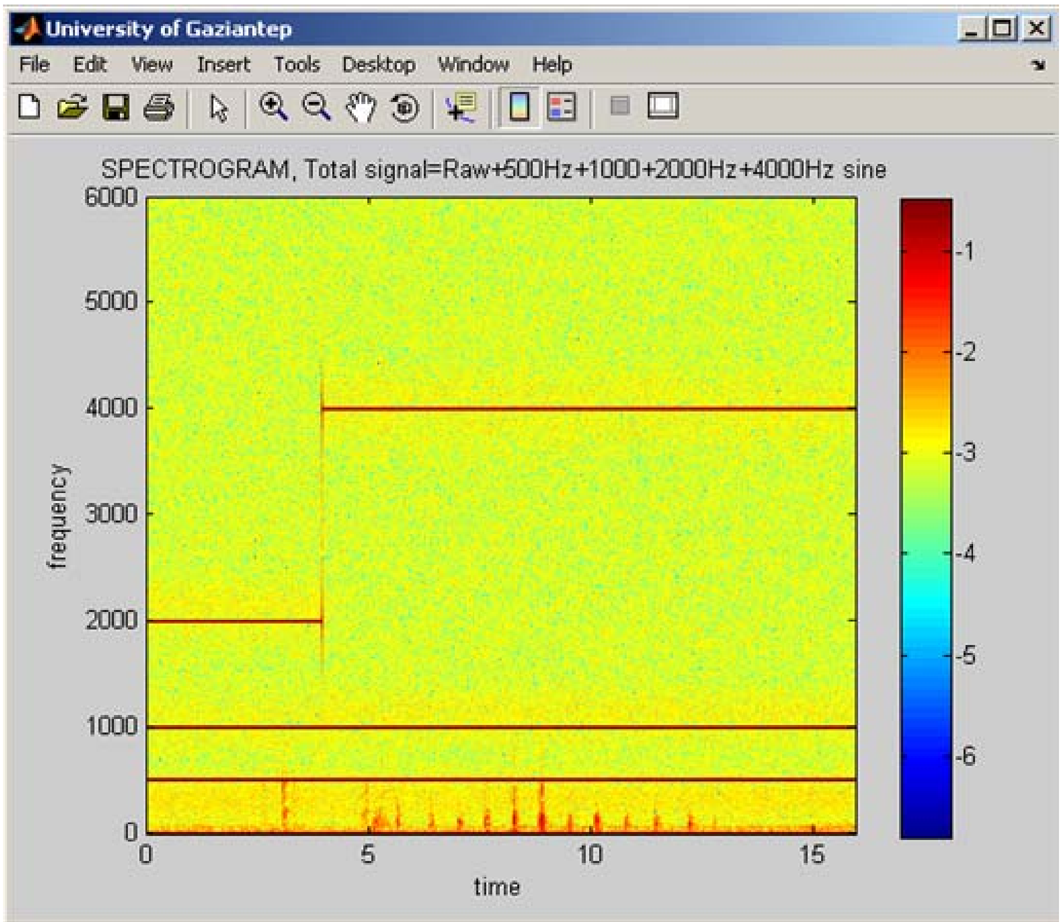

Figure 4.9 Spectrogram of the raw seismic signal with disturbances

4.4.2 Proposed Detection Algorithm

In this section, an algorithm is proposed for footstep detection. The function FSdetection(N,A,T,NRaw) is written to detect the footsteps. It is a Matlab m-file, where N is the number of simout samples, A is the amplitude values of simout variable, T is the time values of simout variable and NRaw is the number of samples of the raw data. FSdetection m-file code is given in Appendix E.

Remember that the outputs of the model is saved as a variable “simout” on the matlab workspace in section 4.3. Now The FSdetection m-file is applied for the simout variable. The function has five stages. The first stage calculates the peaks in the defined limits. The second stage selects the footstep peaks at the defined width. The third stage calculates the footsteps peaks and real time. The forth stage finds the footstep. The final stage is the graphical representation.

The block diagram of the proposed detection algorithm is below;

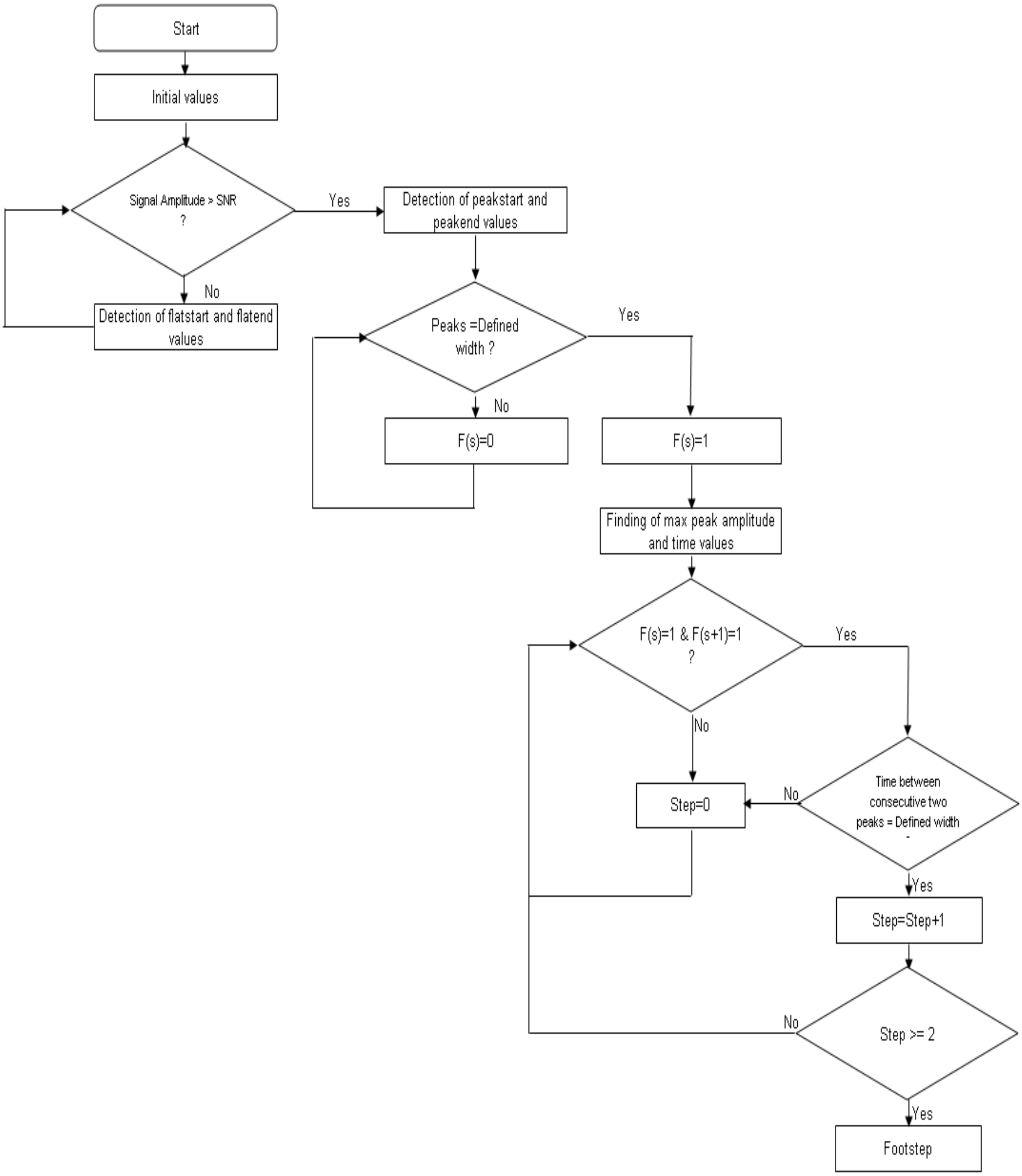

Figure 4.10 Flowchart of the detection algorithm

Variables that used in the algorithm are represented in Figure 4.11. FSdetection function produces a graphical representation to show footsteps in the real-time. In Figure 4.12, Top graph shows peaks as sample points. Middle is peaks as real time. Bottom graph is detected footsteps.

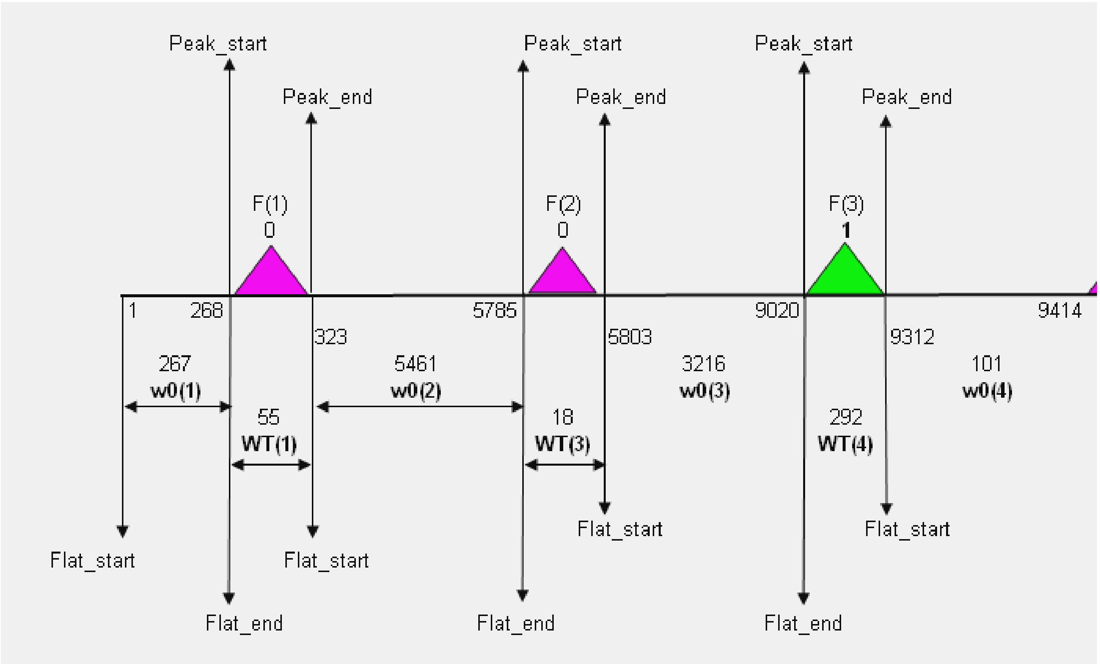

Figure 4.11 Variables of algorithm

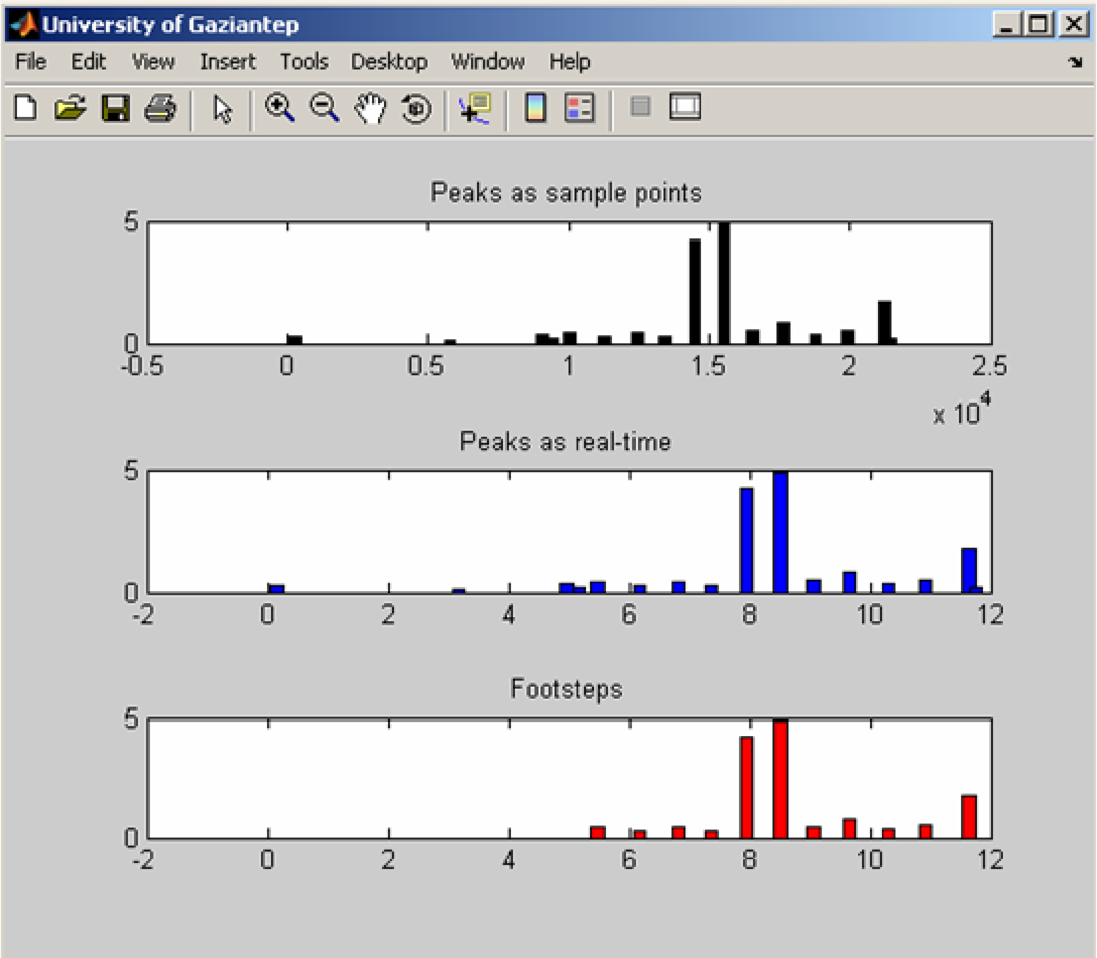

Figure 4.12 Outputs of the FSdetection algorithm

CHAPTER 5

EXPERIMENTS AND TEST RESULTS

5.1 Experiment Design

Seismic signals from human footsteps were observed experimentally at the campus of the University of Gaziantep. Footsteps were detected at ranges up to 10 m. Seismic footstep signal levels were good at short distances from the sensor. However the level of the signals decreased rapidly with increasing distance between a moving person and the sensor. Intruder trials at the campus were conducted along a gravel road in an open area. Weather was sunny.

Test equipments are below,

Computer :Fujitsu-Siemens Esprimo Mobile V5535
Intel Pentium Dual CPU T2370 1.73 GHz, 3MB Ram, Windows-XP

DAQ :NI-9234 USB

Sensor :Geospace GS20-DX

Software :NI Labview Signal Express 2009, Matlab R2006a

5.2 Experiments

Many real-time measurements were done to test the algorithm. In these tests, raw data is firstly plotted, it is located in the first figure. Then the peaks that considered in calculations are plotted, it is located at the top of second figure. These peaks can be a footstep or not. They will be tested by algorithm in the next step. Finally the peaks that are footsteps are plotted, it is located at the bottom of the second figure. If peaks are not footsteps, these peaks will be rejected by the algorithm.

5.2.1 Application 1

Collected raw seismic data is plotted in Figure 5.1. As can be seen from the Figure 5.1, first there are two peaks, then five peaks and finally two peaks.

The number of total raw samples = 793600

The number of output samples of the Simulink model = 52907

Sample Rate = 25600

X-axis Detected Peak Locations= [9.08 14.73 15.43 16.01 16.74 21.44]

Y-axis Detected Peak Values = [0.23 0.35 2.34 5.42 0.18 0.13]

FS_LocationRealX = [14.73 15.43 16.01 16.74]

FS_LocationRealY = [0.35 2.34 5.42 0.18]

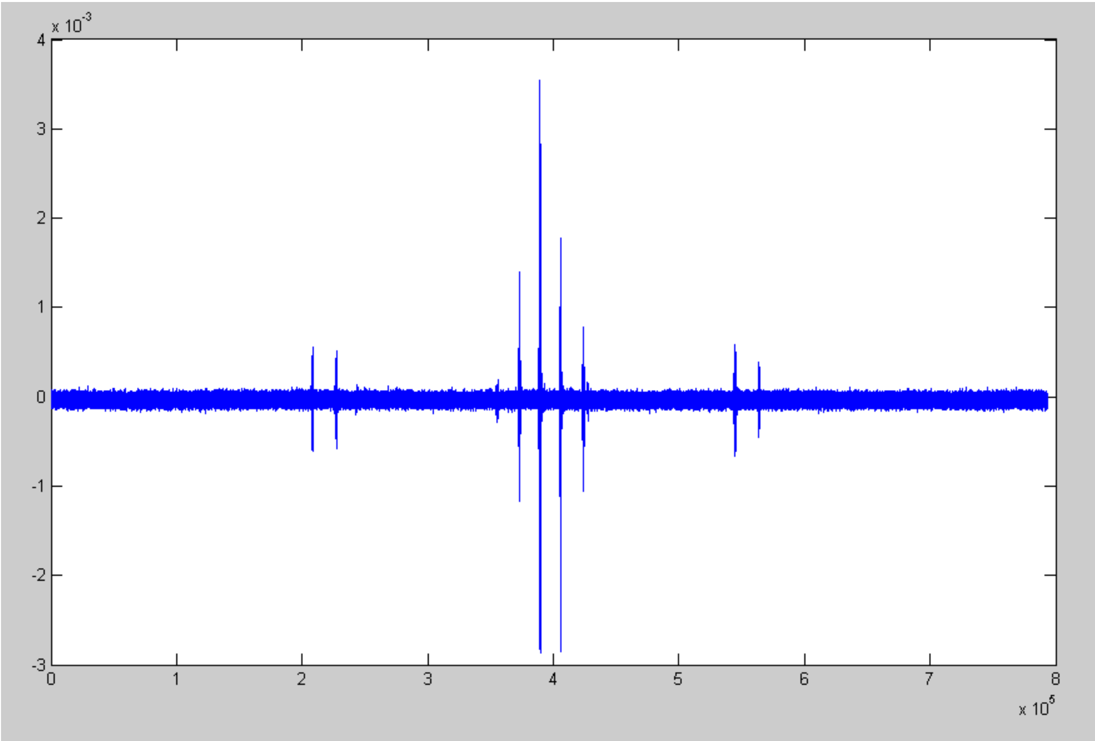

Figure 5.1 Raw seismic data.

After the raw signal is obtained, then detection algorithm is applied on it. As can be seen from the Figure 5.2, first peak are not a footstep, because three or more peaks are necessary for footstep. So next four footsteps are detected, others are rejected.

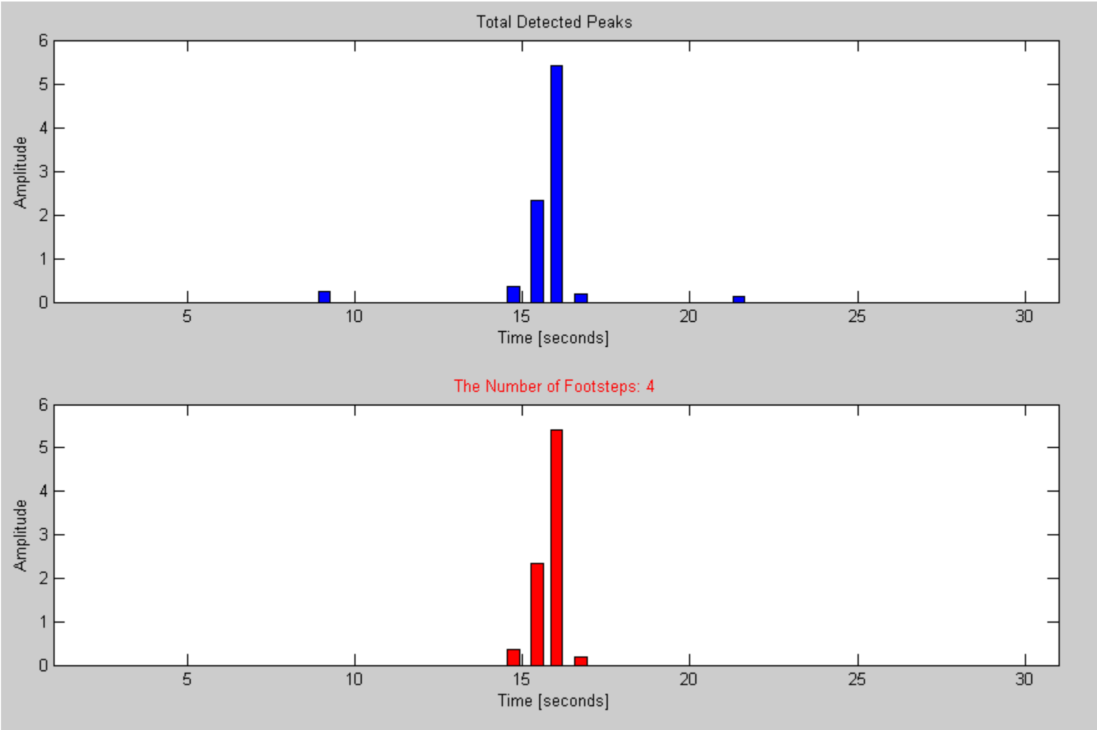

Figure 5.2 Application of the algorithm on the real seismic data.

5.2.2 Application 2

Raw seismic data is plotted in Figure 5.3.

The number of total raw samples = 742400

The number of output samples of the Simulink model = 49494

Sample Rate = 25600

X-axis Detected Peak Locations= [12.55 13.20 13.85 14.51 17.98 23.29]

Y-axis Detected Peak Values = [0.09 0.15 0.24 0.08 0.20 0.11]

FS_LocationRealX = [13.20 13.85 14.51]

FS_LocationRealY = [0.15 0.24 0.08]

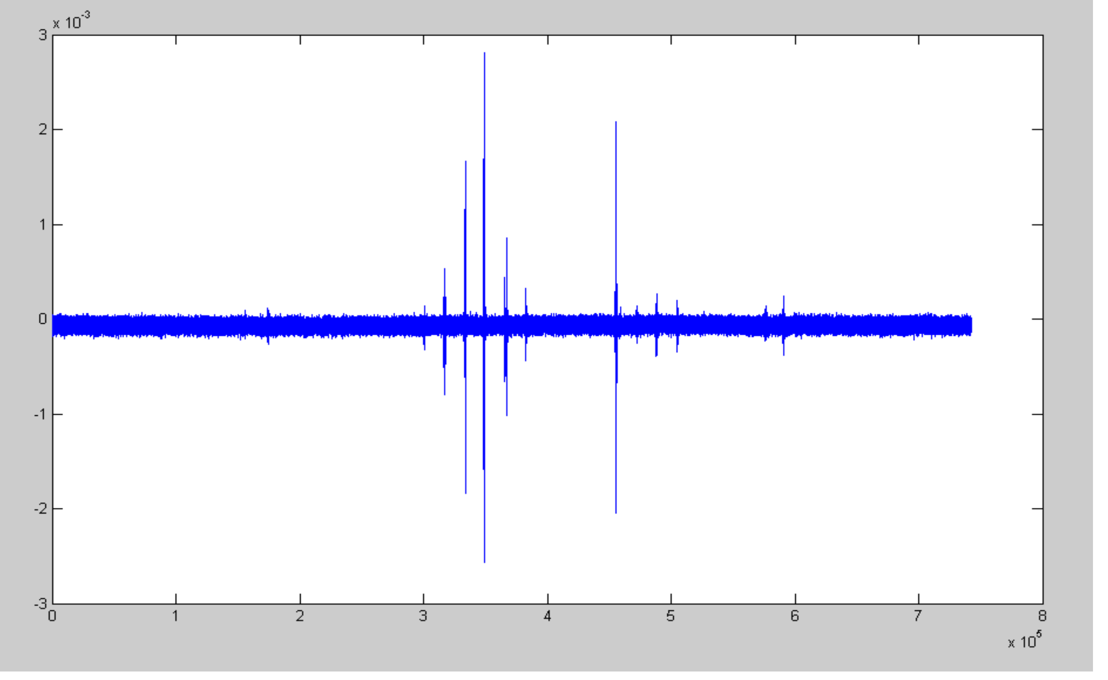

Figure 5.3 Raw seismic data.

After the raw signal is obtained, then detection algorithm is applied on it. As can be seen from the Figure 5.4, three footsteps are detected, others are rejected.

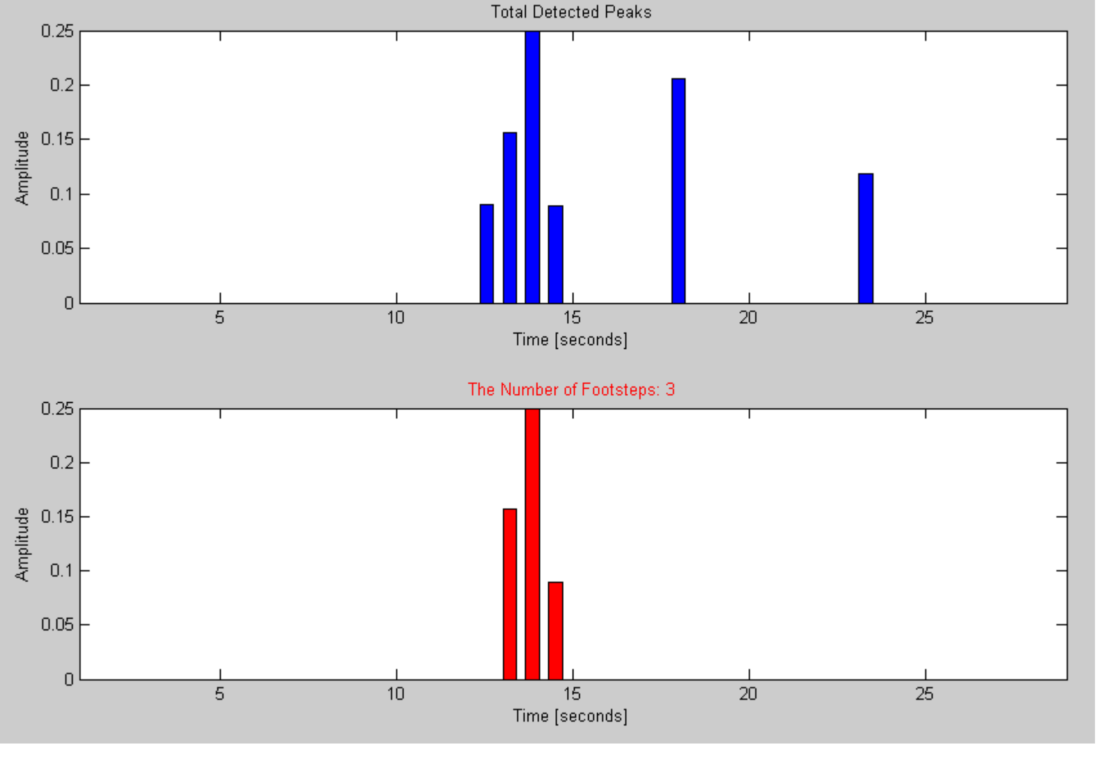

Figure 5.4 Application of the algorithm on the real seismic data.

5.2.3 Application 3

Raw seismic data is plotted in Figure 5.5.

The number of total raw samples = 972800

The number of output samples of the Simulink model = 64854

Sample Rate = 25600

X-axis Detected Peak Locations= [8.71 14.01 18.02 23.45 29.39 30.00 30.71 31.25]

Y-axis Detected Peak Values = [0.08 1.52 0.60 1.19 0.75 7.76 0.96 0.07]

FS_LocationRealX = [29.39 30.00 30.71]

FS_LocationRealY = [0.75 7.76 0.96]

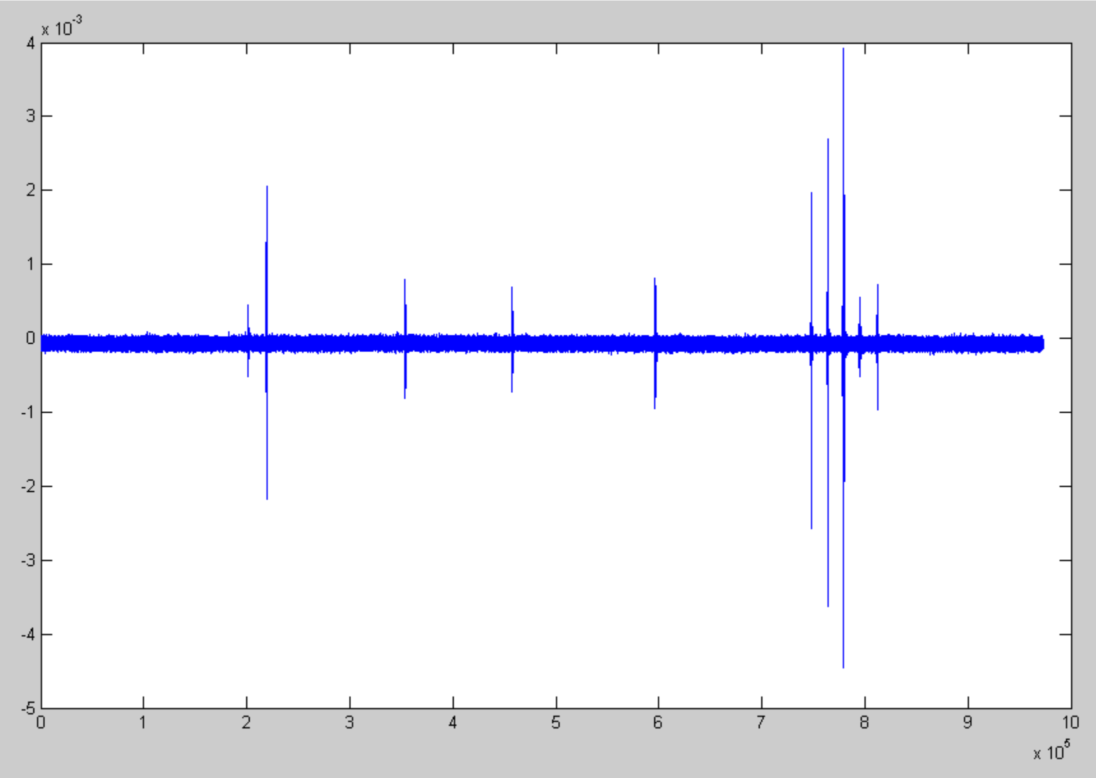

Figure 5.5 Raw seismic data.

Three footsteps are detected, others are rejected in Figure 5.6

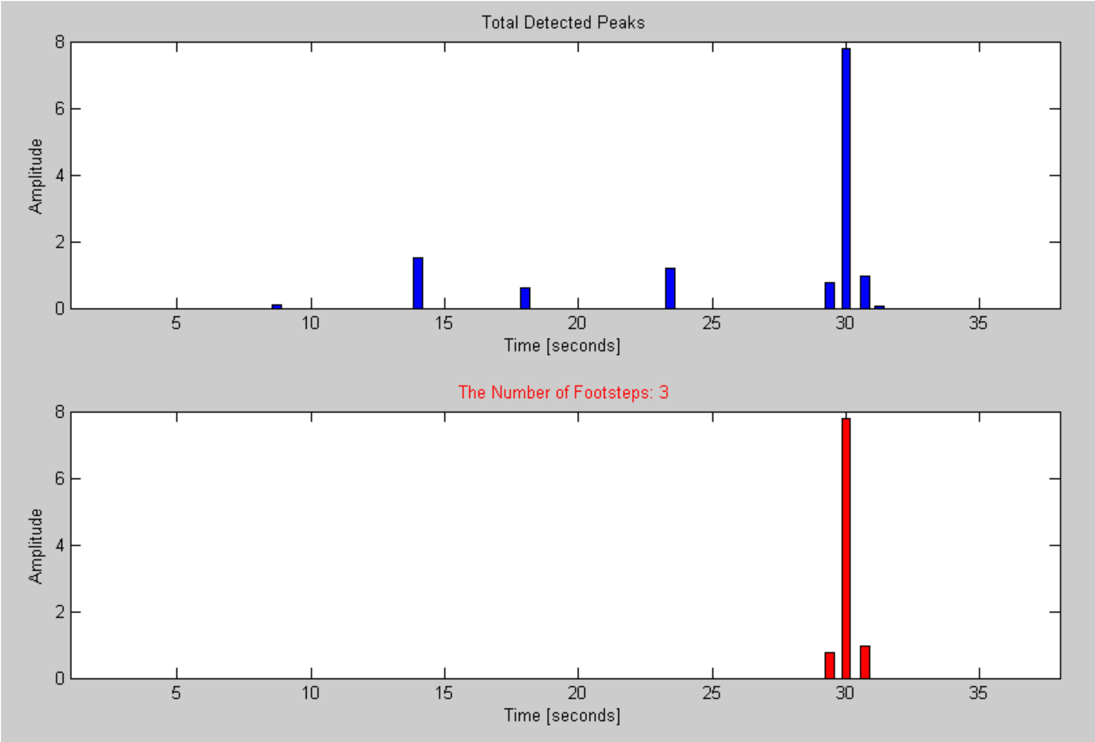

Figure 5.6 Application of the algorithm on the real seismic data.

5.2.4 Application 4

Raw seismic data is plotted in Figure 5.7.

The number of total raw samples = 409600

The number of output samples of the Simulink model = 27307

Sample Rate = 25600

X-axis Detected Peak Locations= [5.79 8.15 8.69 9.31 9.84 9.93 12.33]

Y-axis Detected Peak Values = [0.09 0.15 3.27 0.47 0.07 0.17 0.10]

FS_LocationRealX = [8.15 8.69 9.31]

FS_LocationRealY = [0.15 3.27 0.47]

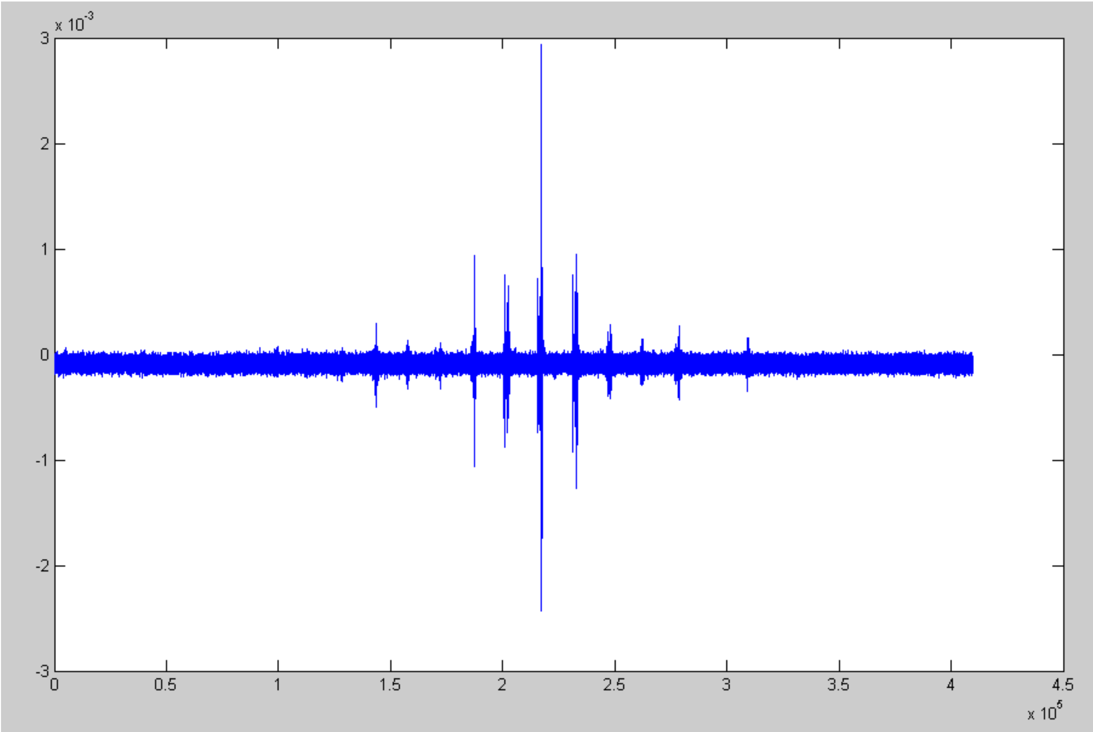

Figure 5.7 Raw seismic data.

Three footsteps are detected, others are rejected in Figure 5.8

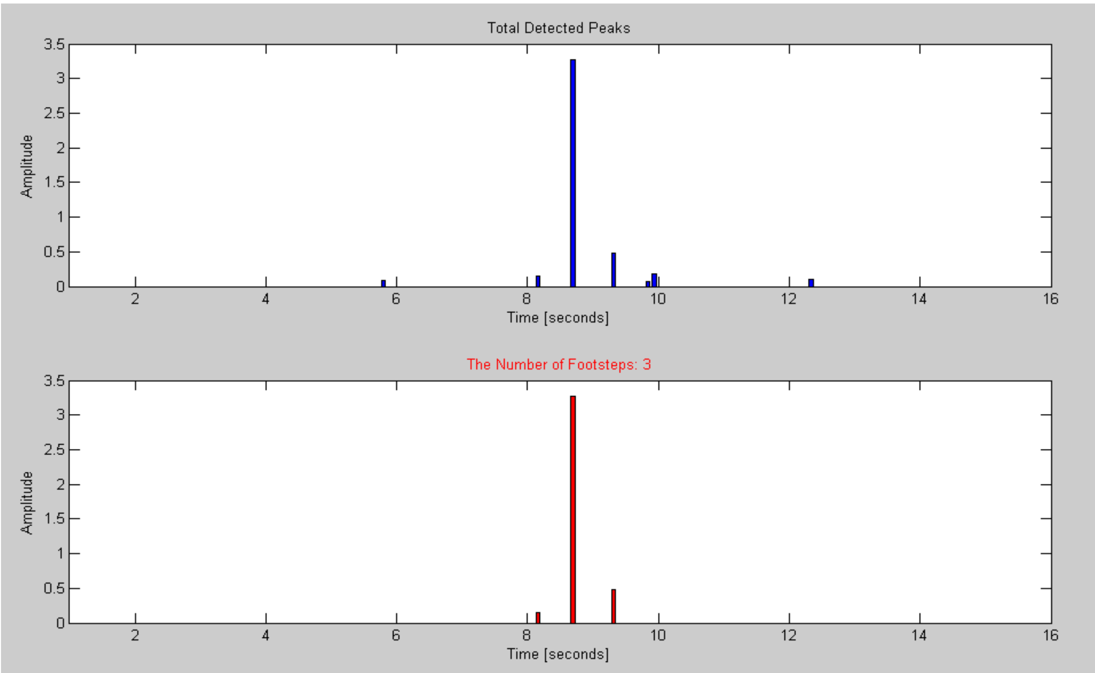

Figure 5.8 Fourth application results

5.2.5 Application 5

Raw seismic data is plotted in Figure 5.9.

The number of total raw samples = 844800

The number of output samples of the Simulink model = 56321

Sample Rate = 25600

X-axis Detected Peak Locations= [0 5.75 12.08 12.69 13.28 13.85 15.01 15.54 16.09 16.35 16.63 17.18 18.28 18.64 18.80 19.40 20.48 21.03]

Y-axis Detected Peak Values = [0 0.53 0.15 0.30 0.38 0.42 0.58 0.26 3.51 0.08 12.02 42.78 1.41 0.08 0.65 0.19 0.32 0.15]

FS_LocationRealX = [0 5.75 12.08 12.69 13.28 13.85 15.01 15.54 16.09 16.35 16.63 17.18 18.28 18.64 18.80 19.40 20.48 21.03]

FS_LocationRealY = [0.15 0.30 0.38 0.42 0.58 0.26 3.51 0.65 0.19 0.32 0.15]

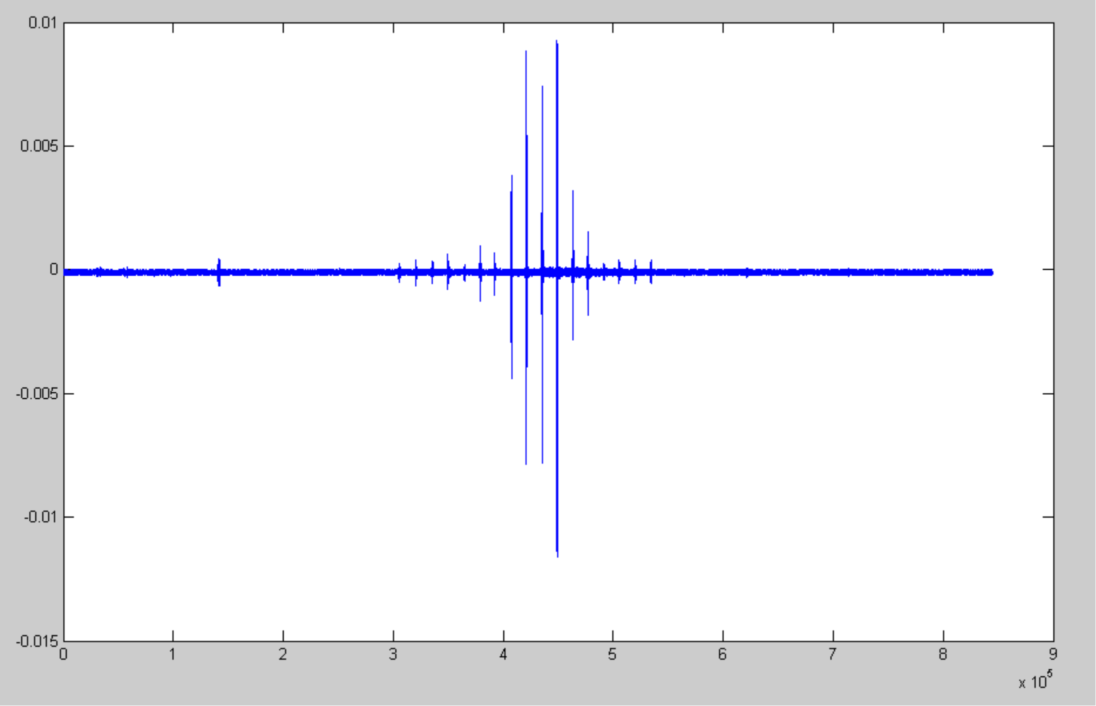

Figure 5.9 Raw seismic data.

Seven footsteps are detected, others are rejected in Figure 5.10

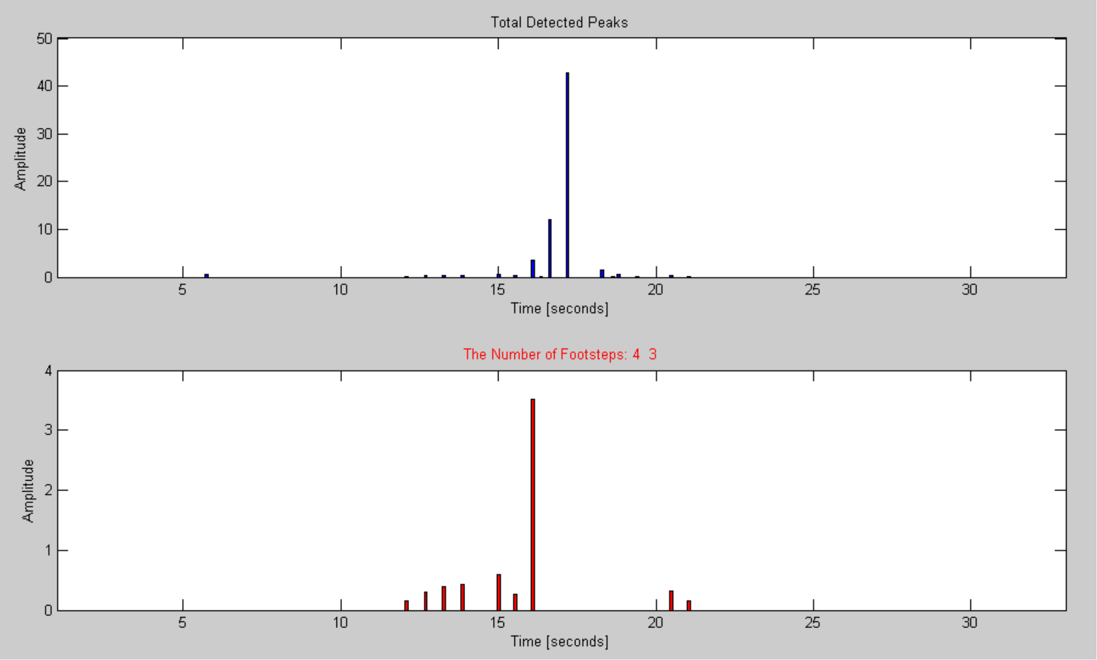

Figure 5.10 Fifth application results

5.3 Results

It is known from the applications that the distance between intruder and sensor is very important. If distance is increased, detection will be decreased. Also noise will be effective for detection. The system has always a noise that comes from sensors, daq card and other equipments. Detection is decreased by increased threshold level of noise. Detection algorithm has five stages. Peaks in defined limits are determined at the first stage. Width of each peaks is controlled at second stage. Magnitude and location of each peaks are determined in third stage. The time between two adjacent peaks is calculated and if it is in limits and if number of adjacent peaks is three or more then they will be defined as footstep in forth stage. Graphical representation is done in the fifth stage.

In application one, time between first two peaks and other four peaks is out of limits. Limits between peaks is defined as footsteps of a walking person.

In application two, four footsteps are detected instead of five footsteps. Because other peaks have low amplitude levels.

In application three, first four peaks are not footstep, because time between them is too big. Footsteps of a walking person is less than these peaks.

In application four, three footsteps are detected, because other peaks have low amplitude levels.

In application five, seven footsteps are detected, because other peaks have low amplitude levels.

CHAPTER 6

CONCLUSION

There are many methods to detect the footsteps. Footstep detection with seismic sensors is considered in this thesis. A new algorithm is developed for detection. In this algorithm, first a raw signal is filtered, then envelope method is applied. Also time between two adjacent peaks are considered, because it is known that footsteps are periodic. Finally footstep peaks are obtained and other peaks are rejected.

Proposed method is applied to real footstep signals with disturbance effects via real time experiments conducted in outdoor conditions. The results obtained reveal the performance of the method in detecting human existence in restricted areas. However the results also show that human detection success significantly depends on external uncontrollable effects such as distance from the sensor and ground characteristics.

In the present work, only one seismic geophone sensor is used. It is not enough to get all seismic activities around the protected area. At least three or more seismic geophone sensors are needed for this. High frequency sine signals 500, 1000, 2000 and 4000 Hz respectively are generated to simulate noise as disturbance.

Also there are some limitations for detection. First of all, distance effects results. When the pedestrian is far from the sensor, the signal level is very low. The second is ground type. If ground is soft, seismic waves will not propagate properly. Therefore the geophone sensor does not detect intruder activities.

In the future works, the use of a three-component geophone will be investigated to detect and track persons and vehicles. The method depends on the analysis of Rayleigh surface waves. Rayleigh surface waves diminish in intensity as R^{-1}. This surface wave is a vector wave that can be used to track the source.

REFERENCES

[1] Richman M.S., Deadrick D.S. (2001) Personnel Tracking Using Seismic Sensors, *Proceedings of SPIE*, **Vol.4393**, pp. 14-21

[2] Succi G., Clapp D., Gambert R.(2001) Footstep Detection and Tracking, *Proceedings of SPIE*, **Vol.4393**, pp. 22-29

[3] Houston K.M., McGaffigan D.P.(2003) Spectrum Analysis Techniques for Personnel Detection Using Seismic Sensors, *Proceedings of SPIE*, **Vol.5090**, pp. 162-173

[4] Mazarakis G.P., Avaritsiotis J.N.(February 2005) A Prototype Sensor Node for Footstep Detection, *Proceedings of the Second European Workshop on Wireless Sensor Networks, IEEE* pp.415-418

[5] Dibazar A.A., Park H.O., Berger T.W., (August 2007) The Application of Dynamic Synapse Neural Networks on Footstep and Vehicle Recognition, *IEEE International Joint Conference on Neural Networks,* pp.1842-1846

[6] Chen C.H., (March 1982) Adaptive and Learning Algorithms for Seismic Detection of Personnel, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **Vol. 4** No.2, pp.129-132

[7] Pavlovic V.D., Velickovic Z.S. (1998) Measurement of The Seismic Waves Propagation Velocity in The Real Medium, *Facta Universitatis The Scientific Journal of Series Physics, Chemistry and Technology* **Vol.1,** No. 5, pp.63-73

[8] Barzilai A., Vanzandt T., Kenny T. Improving The Performance of a Geophone through Capacitive Position Sensing and Feedback. *ASME International Congress Winter*, 1998

[9] Antoniou A. (2006) Digital Signal Processing, (2nd ed.), McGraw Hill Companies Inc.

[10] http://cnx.org/content/m10570/latest/

[11] http://core.ecu.edu/psyc/wuenschk/docs30/Skew-Kurt.doc

[12] Decarlo L.T. On The Meaning and Use of Kurtosis, (1997) *Psychological Methods,* **Vol.2**, No.3, pp.292-307

[13] Press W.H., Teukolsky S.A. (1992) *Numerical Recipes in C*, (2nd ed.), Cambridge University Press

[14] Pakhomov A., Sicignano A., Sandy M., Goldburt T. (2003) Seismic Footstep Signal Characterization, *Proceedings of SPIE*, **Vol.5071**, pp. 297-305

[15] Quinquis A. (2008) Digital Signal Processing using Matlab, (2nd ed.), ISTE Ltd and John Wiley & Sons Inc.

[16] http://www.mathworks.com/help/toolbox/dspblks/ref/firdecimation.html

[17] http://www.mathworks.com/products/sigprocblockset/demos.html?file=/products/ demos/shipping/ dspblks/dspenvdet.html#1

APPENDIX A

SEISMIC SENSOR GEOPHONE GS-20DX

Table A.1 Geophone GS-20DX Characteristics

No	Characteristics	Values
1	Natural Frequency	10 Hz
2	Frequency Tolerance	± 5 %
3	Maintains Fn Specifications to Tilt Angle of	20^0
4	Typical Spurious Frequency	> 250 Hz
5	Harmonic Distortion with Driving Velocity of 0.7 in/sec (1.8 cm/sec)P-P	< .2 %
6	Distortion Measured at	12 Hz
7	Open Circuit Damping (Bo) (for all coil resistances)	0.30 ± 10%
8	Standard Coil Resistance (Rc) ± 5%	395 Ohms
9	Intrinsic Voltage Sensitivity ± 10% ,V/cm/sec	0.28
10	Damping Constant at Fn (Rt Bc Fn)	5500 for 395 Ohms
11	Normalized Transduction Constant	0.0138 (sq.root Rc)V/cm/sec
12	Moving Mass ± .5g	11g
13	Case to Coil Motion P-P	Min =0,8mm, Max=1.5mm
14	Operating Temperature Range (°C)	-45° to +100°
15	Dimensions (less terminals*)	Height=3.3 cm, Diameter=2,54cm, Weight=87.3 g

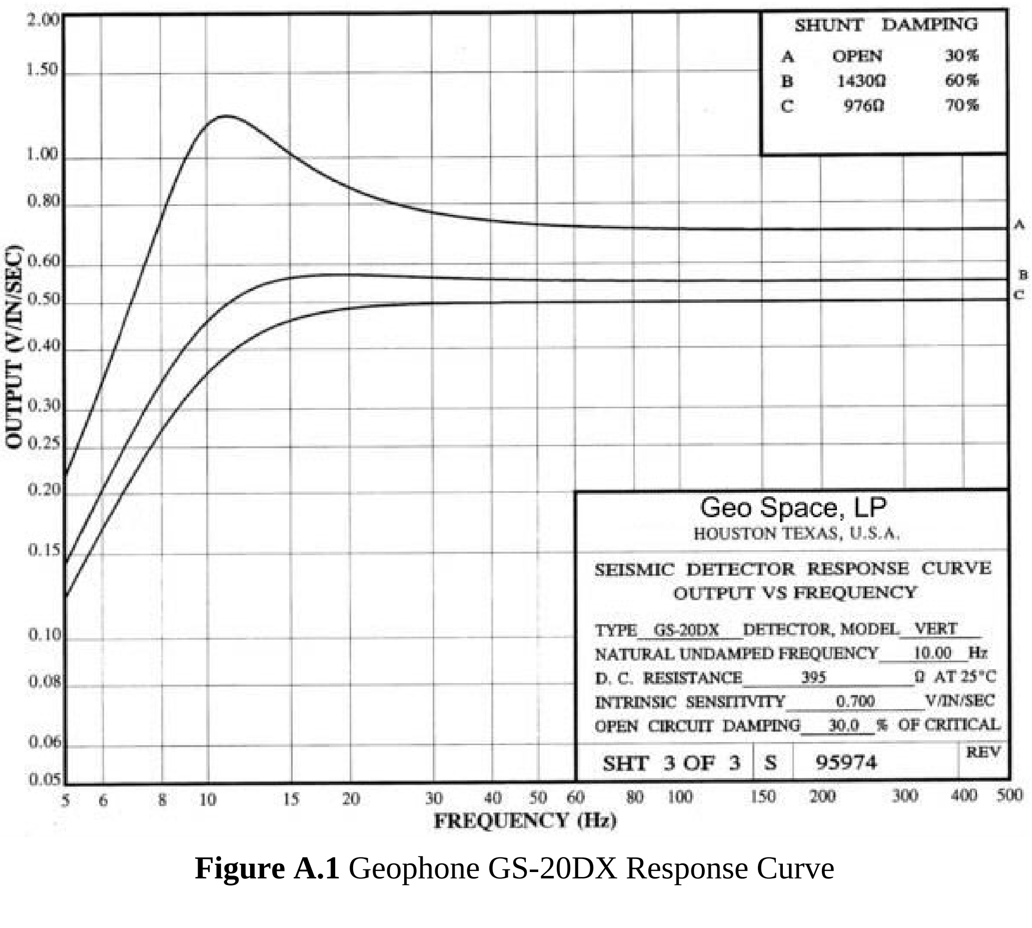

Figure A.1 Geophone GS-20DX Response Curve

APPENDIX B

DAQ CARD NI-9234

Table B.1 NI-9234 DAQ Card Characteristics

No	Characteristics	Values	
1	PC Bağlantı Portu	USB	
2	ADC Resolution (bits)	24	
3	Type of ADC	Delta-sigma with analog prefiltering	
4	Dynamic Range (dB)	102	
5	Sampling rate per channel	51.2 kS/s	
6	Analog inputs	4	
7	Input range	± 5 volt	
8	TEDS Support	Yes, IEEE 1451.4 TEDS Class I	
9	Master timebase (internal frequency)	13.1 Mhz	
10	Master timebase (accuracy)	± 50 ppm max	
11	Input coupling	Software selectable AC/DC	
12	AC cutoff frequency	-3 dB	0.5 Hz typ
12	AC cutoff frequency	-0.1 dB	4.6 Hz max
13	AC voltage full scale range	Typical	5.1 Vpk
13	AC voltage full scale range	Minimum	5 Vpk
13	AC voltage full scale range	Maximum	5.2 Vpk
14	Common mode voltage	Al- to earth ground	± 2V
15	IEPE excitation current	Minimum	2 mA
15	IEPE excitation current	Typical	2.1 mA
16	IEPE compliance voltage	19 V max	
17	Overvoltage protection (with respect to chassis ground)	For an IEPE sensor connected to AI+ and AI-	± 30 V
17	Overvoltage protection (with respect to chassis ground)	For a low-impedance source connected to AI+ and AI-	-6 to 30 V
18	Internal master timebase (fM)	Frequency	13.1072 MHz
18	Internal master timebase (fM)	Accuracy	±50 ppm max
19	Data rate range (fs) using internal master timebase	Minimum	1.652 kS/s
19	Data rate range (fs) using internal master timebase	Maximum	51.2 kS/s
20	Power-on glitch	90 µA for 10 µs	
21	Input delay	38.4/ fs + 3.2 µs	
22	Gain drift	Typical	0.14 mdB/°C (16 ppm/°C)
22	Gain drift	Maximum	0.45 mdB/°C (52 ppm/°C)

No	Characteristics	Values			
23	Offset drift	Typical	19.2 µV/°C		
		Maximum	118 µV/°C		
24	Channel-to-channel matching	Gain	Typical	0.01 dB	
			Maximum	0.04 dB	
		Phase (f_{in} in kHz)	f_{in} · 0.045° + 0.04 max		
25	Passband	Frequency	0.45 · f_s		
		Flatness (f_s = 51.2 kS/s)	±40 mdB (pk-to-pk max)		
26	Phase nonlinearity, (f_s = 51.2 kS/s)	±0.45° max			
27	Stopband	Frequency	0.55 · f_s		
		Rejection	100 dB		
28	Alias-free bandwidth	0.45 · f_s			
29	Oversample rate	64 · f_s			
30	Crosstalk (1 kHz)	-110 dB			
31	CMRR (f_{in} δ 1 kHz)	Minimum	40 dB		
		Typical	47 dB		
32	SFDR (f_{in} = 1 kHz, –60 dBFS)	120 dB			
33	Input impedance	Differential	305 kΩ		
		AI– (shield) to chassis ground	50 Ω		
34	Safety Standards	IEC 61010-1, EN 61010-1 UL 61010-1, CSA 61010-1			
35	Electromagnetic Compatibility	EN 61326 (IEC 61326): Class A emissions; Basic immunity EN 55011 (CISPR 11): Group 1, Class A emissions AS/NZS CISPR 11: Group 1, Class A emissions FCC 47 CFR Part 15B: Class A emissions ICES-001: Class A emissions			
36	Operating vibration	Random (IEC 60068-2-64).........5 g_{rms}, 10 to 500 Hz Sinusoidal (IEC 60068-2-6) ....5 g, 10 to 500 Hz			
37	Idle channel noise and noise density	**Idle Channel**	**51.2 kS/s**	**25.6 kS/s**	**2.048 kS/s**
		Noise	97 dBFS	99 dBFS	103 dBFS
			50 µVrms	40 µVrms	25 µVrms
		Noise density	310 nV/√Hz	350 nV/√Hz	780 nV/√Hz
38	Accuracy	**Measurement Conditions**	**Percent of Reading (Gain Error)**	**Percent of Range (5.1 Vpk) (Offset Error)**	
		Calibrated max (–40 to 70 °C)	0.34%, ±0.03 dB	±0.14%, 7.1 mV	
		Uncalibrated max (–40 to 70 °C)	1.9%, ±0.16 dB	±0.27%, 13.9 mV	

APPENDIX C

MATLAB CODES USED IN CHAPTER-2

The Function "positiveFFT"

```
function [X,freq]=positiveFFT(x,Fs)
N=length(x);            %get the number of points
k=0:N-1;                %create a vector from 0 to N-1
T=N/Fs;                 %get the frequency interval
freq=k/T;               %create the frequency range
X=fft(x)/N;             % normalize the data
%only want the first half of the FFT, since it is redundant
cutOff = ceil(N/2);
%take only the first half of the spectrum
X = X(1:cutOff);
freq = freq(1:cutOff);
```

Figure 2.3

```
fo = 15; %frequency of the sine wave
Fs = 200; %sampling rate
Ts = 1/Fs; %sampling period
t1 = 0:Ts:1 - Ts; %time vector
n1 = length(t1); %number of samples
y1 = 4*sin(2*pi*fo*t1);
%plot the curve in the time domain
subplot(2,1,1)
plot(t1,y1) %plot the sine wave
 [Y1,freq1] = positiveFFT(y1,Fs); %compute the frequency spectrum
%positiveFFT is a custom function that is included in the source file
subplot(2,1,2);
stem(freq1,abs(Y1)) %plot the frequency spectrum
```

Figure 2.4

```
fo = 15; %frequency of the sine wave
Fs = 200; %sampling rate
Ts = 1/Fs; %sampling periodl
t2 = 0:Ts:0.95 -Ts; %time vector(notice the difference here!)
n2 = length(t2); %number of samples
y2 =4*sin(2*pi*fo*t2);
subplot(2,1,1) %plot the curve in the time domain
plot(t2,y2)
[Y2,freq2] = positiveFFT(y2,Fs); %compute the frequency spectrum
subplot(2,1,2);
stem(freq2,abs(Y2)) %plot the frequency spectrum
```

Figure 2.5

```
windowHanning = window(@hann,n2).'; %create a hanning window vector, n2=190
hanningWindowFigure = figure;
plot(windowHanning); %plot the hanning window
```

Figure 2.6

```
windowedSignal = windowHanning.*y2; %multiply the inputsignal with this window
windowedSignalPlot = figure;
plot(t2,windowedSignal) %plot the windowed signal
```

Figure 2.7

```
[a,b] = positiveFFT(y2,Fs); %calculate positive fft for non-windowed signal
[c,d] = positiveFFT(windowedSignal,Fs); %calculate positive fft for windowed signal
c = c * 2; %multiply by the coherence factor
fftWindowedSignalLinear = figure;
plot(b,abs(a),d,abs(c),'r' )
legend('Non-windowed signal' ,'Windowed signal' )
%plot the windowed signal in log scale
fftWindowedSignalLog = figure; plot(b,20*log10(abs(a)),d,20*log10(abs(c)),'r')
```

APPENDIX D

MATLAB CODES of ConvertTDMS.m FUNCTION

```
function [ConvertedData,ConvertVer]=convertTDMS(SaveConvertedFile,filename)

%Function to load LabView TDMS data file(s) into variables in the MATLAB
workspace.
%An *.MAT file can also be created.  If called with one input, the user
selects
%a data file.  This function was submitted to MATLAB Central's File
Exchange by
%Robert Seltzer on 1 SEP 10.
%
%   TDMS format is based on information provided by National Instruments
%   at:    http://zone.ni.com/devzone/cda/tut/p/id/5696
%
% [ConvertedData,Index,ConvertVer]=convertTDMS(SaveConvertedFile,filename);
%
%       Inputs:
%               SaveConvertedFile (required) - Logical flag (true/false)
that
%                 determines whether a MAT file is created.  The MAT file's
name
%                 is the same as 'filename' except that the 'TDMS' file
extension is
%                 replaced with 'MAT'.  The MAT file is saved in the same
folder
%                 and will overwrite an existing file without warning.  The
%                 MAT file contains all the output variables.
%
%               filename (optional) - Filename (fully defined) to be
converted.
%                 If not supplied, the user is provided dialog box to open
file.
%                 Can be a cell array of files for bulk conversion.
%
%       Outputs:
%               ConvertedData (required) - Structure with all of the data
objects.
%               ConvertVer (required) - the version number of this
function.
%

%---------------------------------------------
%Brad Humphreys - v1.0 2008-04-23
%ZIN Technologies
%---------------------------------------------

%---------------------------------------------
%Brad Humphreys - v1.1 2008-07-03
%ZIN Technologies
%-Added abilty for timestamp to be a raw data type, not just meta data.
%-Addressed an issue with having a default nsmaples entry for new objects.
%-Added Error trap if file name not found.
%-Corrected significant problem where it was assumed that once an object
%    existsed, it would in in every subsequent segement.  This is not true.
%---------------------------------------------

%---------------------------------------------
%Grant Lohsen - v1.2 2009-11-15
%Georgia Tech Research Institute
%-Converts TDMS v2 files
%Folks, it's not pretty but I don't have time to make it pretty. Enjoy.
%---------------------------------------------
```

```
%-------------------------------------------
%Jeff Sitterle - v1.3 2010-01-10
%Georgia Tech Research Institute
%Modified to return all information stored in the TDMS file to inlcude
%name, start time, start time offset, samples per read, total samples, unit
%description, and unit string.  Also provides event time and event
%description in text form
%Vast speed improvement as save was the previous longest task
%-------------------------------------------

%-------------------------------------------
%Grant Lohsen - v1.4 2009-04-15
%Georgia Tech Research Institute
%Reads file header info and stores in the Root Structure.
%-------------------------------------------

%-------------------------------------------
%Robert Seltzer - v1.5 2010-07-14
%BorgWarner Morse TEC
%-Tested in MATLAB 2007b and 2010a.
%-APPEARS to now be compatible with TDMS version 1.1 (a.k.a 4712) files;
%   although, this has not been extensively tested.  For some unknown
%   reason, the version 1.2 (4713) files process noticeably faster. I think
%   that it may be related to the 'TDSm' tag.
%-"Time Stamp" data type was not tested.
%-"Waveform" fields was not tested.
%-Fixed an error in the 'LV2MatlabDataType' function where LabView data
type
%   'tdsTypeSingleFloat' was defined as MATLAB data type 'float64' .
Changed
%   to 'float32'.
%-Added error trapping.
%-Added feature to count the number of segments for pre-allocation as
%   opposed to estimating the number of segments.
%-Added option to save the data in a MAT file.
%-Fixed "invalid field name" error caused by excessive string lengths.
%-------------------------------------------

%-------------------------------------------
%Robert Seltzer - v1.6 2010-09-01
%BorgWarner Morse TEC
%-Tested in MATLAB 2010a.
%-Fixed the "Coversion to cell from char is not possible" error found
%  by Francisco Botero in version 1.5.
%-Added capability to process both fragmented or defragmented data.
%-Fixed the "field" error found by Lawrence.
%-------------------------------------------

%-------------------------------------------
%Christian Buxel - V1.7 2010-09-17
%RWTH Aachen
%-Tested in Matlab2007b.
%-Added support for german umlauts (Ä,ä,Ö,ö,Ü,ü,ß) in 'propsName'
%-------------------------------------------

%-------------------------------------------
%André Rüegg - V1.7 2010-09-29
%Supercomputing Systems AG
%-Tested in MATLAB 2006a & 2010b
```

```
%-Make sure that data can be loaded correctly independently of character
% encoding set in matlab.
%-Fixed error if object consists of several segments with identical segment
% information (if rawdataindex==0, not all segments were loaded)
%----------------------------------------------

%Initialize outputs
ConvertVer='1.7';    %Version number of this conversion function
ConvertedData=[];

switch nargin
    case 0
        e=errordlg('The function requires at least 1 input
argument','Insufficient Input Arguments');
        uiwait(e)
        return

    case 1

        if ~islogical(SaveConvertedFile)
            if ~ismember(SaveConvertedFile,[0,1])
                e=errordlg('The function''s input argument must be ''True''
or ''False''','Invalid Input Argument');
                uiwait(e)
                return
            end
        end

        %Prompt the user for the file
        [filename,pathname,filterindex]=uigetfile({'*.tdms','All Files
(*.tdms)'},'Choose a TDMS File');
        if filename==0
            return
        end
        filename=fullfile(pathname,filename);
        infilename=cellstr(filename);

    case 2

        if ~islogical(SaveConvertedFile)
            if ~ismember(SaveConvertedFile,[0,1])
                e=errordlg('The function''s first input argument must be
''True'' or ''False''','Invalid Input Argument');
                uiwait(e)
                return
            end
        end

        if ~ischar(filename) && ~iscell(filename)
            e=errordlg(['The function''s second input argument (file list)
must be either a character string for 1 file '...
                'or a cell array of 1 or more files'],'Invalid Input
Argument');
            uiwait(e)
            return
        end

        if iscell(filename)
            %For a list of files
            infilename=filename;
```

```
        else
            infilename=cellstr(filename);
        end

    otherwise
        e=errordlg('The function requires 1 or 2 input arguments','Too Many
Input Arguments');
        uiwait(e)
        return

end

for fnum=1:numel(infilename)

    if ~exist(infilename{fnum},'file')
        e=errordlg(sprintf('File ''%s'' not found.',infilename{fnum}),'File
Not Found');
        uiwait(e)
        return
    end

    FileNameLong=infilename{fnum};
    [pathstr,name,ext]=fileparts(FileNameLong);
    FileNameShort=sprintf('%s%s',name,ext);
    FileNameNoExt=name;
    FileFolder=pathstr;

    if fnum==1
        fprintf('\n\n')
    end
    fprintf('Converting ''%s''...',FileNameShort)

    fid=fopen(FileNameLong);

    if fid==-1
        e=errordlg(sprintf('Could not open ''%s''.',FileNameLong),'File
Cannot Be Opened');
        uiwait(e)
        fprintf('\n\n')
        return
    end


%*********************************************************************
*******************************************
    %Count the number of segments.  While doing the count, also include
error trapping.
    %Find the end of the file
    fseek(fid,0,'eof');
    eoff=ftell(fid);
    frewind(fid);

    segCnt=0;
    CurrPosn=0;
    LeadInByteCount=28; %From the National Instruments web page
(http://zone.ni.com/devzone/cda/tut/p/id/5696) under
    %the 'Lead In' description on page 2: Counted the bytes shown in the
table.
    while (ftell(fid) ~= eoff)
```

```
        Ttag=fread(fid,1,'uint8');
        Dtag=fread(fid,1,'uint8');
        Stag=fread(fid,1,'uint8');
        mtag=fread(fid,1,'uint8');

        if Ttag==84 && Dtag==68 && Stag==83 && mtag==109
            %Apparently, this sequence of numbers identifies the start of a
new segment.

            segCnt=segCnt+1;

            if segCnt==1
                StartPosn=0;
            else
                StartPosn=CurrPosn;
            end

            %ToC Field
            ToC=fread(fid,1,'uint32');
            kTocMetaData=bitget(ToC,2);
            kTocNewObject=bitget(ToC,3);
            kTocRawData=bitget(ToC,4);
            kTocInterleavedData=bitget(ToC,6);
            kTocBigEndian=bitget(ToC,7);

            if kTocInterleavedData
                e=errordlg(sprintf(['Seqment %.0f within ''%s'' has
interleaved data which is not supported with this '...
                    'function
(%s.m).'],segCnt,TDMSFileNameShort,mfilename),'Interleaved Data Not
Supported');
                fclose(fid);
                uiwait(e)
                uiwait
            end

            if kTocBigEndian
                e=errordlg(sprintf(['Seqment %.0f within ''%s'' uses the
big-endian data format which is not supported '...
                    'with this function
(%s.m).'],segCnt,TDMSFileNameShort,mfilename),'Big-Endian Data Format Not
Supported');
                fclose(fid);
                uiwait(e)
                uiwait
            end

            %TDMS format version number
            vernum=fread(fid,1,'uint32');
            if ~ismember(vernum,[4712,4713])
                e=errordlg(sprintf(['Seqment %.0f within ''%s'' used
LabView TDMS file format version %.0f which is not '...
                    'supported with this function
(%s.m).'],segCnt,TDMSFileNameShort,vernum,mfilename),...
                    'TDMS File Format Not Supported');
                fclose(fid);
                uiwait(e)
                uiwait
            end
```

```
            %From the National Instruments web page
(http://zone.ni.com/devzone/cda/tut/p/id/5696) under the
            %'Lead In' description on page 2:
            %The next eight bytes (64-bit unsigned integer) describe the
length of the remaining segment (overall length
            %of the segment minus length of the lead in). If further
segments are appended to the file, this number can be
            %used to locate the starting point of the following segment. If
an application encountered a severe problem
            %while writing to a TDMS file (crash, power outage), all bytes
of this integer can be 0xFF. This can only
            %happen to the last segment in a file.
            segLength=fread(fid,1,'uint64');
            metaLength=fread(fid,1,'uint64');
            TotalLength=segLength+LeadInByteCount;
            CurrPosn=CurrPosn+TotalLength;

            SegInfo(segCnt).SegStartPosn=StartPosn;
            SegInfo(segCnt).MetaStartPosn=StartPosn+LeadInByteCount;

SegInfo(segCnt).DataStartPosn=SegInfo(segCnt).MetaStartPosn+metaLength;

            fseek(fid,CurrPosn,'bof');      %Move to the beginning position
of the next segment
        end

    end
    NumOfSeg=segCnt;

%***************************************************************************
********************************************

    %Initialize variables for the file conversion
    ob=[];
    lastIndex=[];
    for segCnt=1:NumOfSeg

        fseek(fid,SegInfo(segCnt).SegStartPosn,'bof');

        Ttag=fread(fid,1,'uint8');
        Dtag=fread(fid,1,'uint8');
        Stag=fread(fid,1,'uint8');
        mtag=fread(fid,1,'uint8');

        %ToC Field
        ToC=fread(fid,1,'uint32');
        kTocMetaData=bitget(ToC,2);
        kTocNewObject=bitget(ToC,3);
        kTocRawData=bitget(ToC,4);
        kTocInterleavedData=bitget(ToC,6);
        kTocBigEndian=bitget(ToC,7);

        vernum=fread(fid,1,'uint32');                           %TDMS
format version number

        segLength=fread(fid,1,'uint64');

        metaLength=fread(fid,1,'uint64');
```

```
        %Process Meta Data
        if kTocMetaData
            clear index

            numObjInSeg=fread(fid,1,'uint32');

            for q=1:numObjInSeg

                obLength=fread(fid,1,'uint32');                 %Get the
length of the objects name
                obname=convertToText(fread(fid,obLength,'uint8'))'; %Get
the objects name

                %Fix Object Name
                if strcmp(obname,'/')
                    obname='Root';
                else

[obname,TruncFieldName,ValidFieldName]=fixcharformatlab(obname);

                    if ~ValidFieldName
                        e=errordlg(sprintf('A valid field name could not be
created for ''%s''.',obname),...
                            'Cannot Create Valid Field Name');
                        uiwait(e)
                        fclose(fid);
                        fprintf('\n\n')
                        return
                    end

                    NameUsed=false;
                    if exist('index','var')
                        if any(strcmpi({index.name},obname))
                            NameUsed=true;
                        end
                    end

                    if NameUsed
                        %The name has already been used.  Add numbers to
the end until the name is unique.
                        MaxNameLen=namelengthmax;
                        if TruncFieldName
                            BaseName=obname(1:MaxNameLen);
                        else
                            BaseName=obname;
                        end
                        HaveValidName=false;
                        NameCount=1;
                        while ~HaveValidName

                            CountStr=sprintf('_%.0f',NameCount);

                            if TruncFieldName
                                NewName=sprintf('%s%s',BaseName(1:(end-
numel(CountStr))),CountStr);
                            else
                                NewName=sprintf('%s%s',BaseName,CountStr);
                            end
```

```
                            if numel(NewName)>MaxNameLen
                                e=errordlg(sprintf('A unique, valid field 
name could not be created for ''%s''.',...
                                    obname),'Cannot Create Valid Field 
Name');
                                uiwait(e)
                                fclose(fid);
                                fprintf('\n\n')
                                return
                            end

                            if all(~strcmpi({index.name},NewName))
                                HaveValidName=true;
                                if TruncFieldName
                                    fprintf('\n\n\tField name ''%s'' is too 
long and\n\t\thas been truncated to ''%s''.\n',...
                                        obname,NewName)
                                else
                                    fprintf('\n\n\tField name ''%s'' 
already exits so\n\t\tit has been changed to ''%s''.\n',...
                                        obname,NewName)
                                end
                                obname=NewName;
                            else
                                NameCount=NameCount+1;
                            end
                        end
                    end
                end

                %Create the 'index' structure
                if exist('index','var')
                    index(end+1).name=obname;
                else
                    index.name=obname;
                end

                %Validate the object
                if isfield(ob,obname)
                    index(end).newob=false;
                else
                    ob.(obname)=[];     %Create a blank version of the 
object
                    index(end).newob=true;
                end

                %Get the raw data Index
                rawdataindex=fread(fid,1,'uint32');
                if rawdataindex==0
                    % Use index information of the last segement of this 
object
                    fields=fieldnames(lastIndex.(obname));
                    for i=1:numel(fields)
                        
index(end).(fields{i})=lastIndex.(obname).(fields{i});
                    end
                elseif rawdataindex+1==2^32
                    %Objects raw data index matches previous index - no 
changes.  The root object will always have an
```

```
                    %FFFFFFFF entry
                    if strcmpi(index(end).name,'Root')
                        index(end).rawdataindex=0;
                        index(end).rawDataInThisSeg=false;
                    else
                        %Need to account for the case where an object
(besides the 'root') is added that has no data but
                        %reports using previous.
                        if index(end).newob
                            index(end).rawdataindex=0;
                            index(end).rawDataInThisSeg=false;
                        else
                            if kTocRawData
                                index(end).rawdataindex=index(end-
1).rawdataindex;

                                index(end).rawDataInThisSeg=true;
                            else
                                index(end).rawdataindex=0;
                                index(end).rawDataInThisSeg=false;
                            end
                        end
                    end
                else
                    %Get new object information
                    index(end).rawdataindex=rawdataindex;
                    index(end).dataType=fread(fid,1,'uint32');
                    index(end).arrayDim=fread(fid,1,'uint32');
                    index(end).nValues=fread(fid,1,'uint64');
                    if index(end).dataType==32
                        %Datatype is a string
                        index(end).byteSize=fread(fid,1,'uint64');
                    else
                        index(end).byteSize=0;
                    end
                    index(end).rawDataInThisSeg=true;
                end

                %Save index information of this segment of this object
                lastIndex.(obname)=index(end);

                %Get the properties
                index(end).numProps=fread(fid,1,'uint32');
                for p=1:index(end).numProps
                    propNameLength=fread(fid,1,'uint32');
                    propsName=fread(fid,propNameLength,'*uint8')';
                    propsName=native2unicode(propsName,'UTF-8');
                    propsName=fixcharformatlab(propsName);
                    propsDataType=fread(fid,1,'uint32');
                    propExists=isfield(ob.(obname),propsName);
                    dataExists=isfield(ob.(obname),'data');

                    if dataExists
                        %Get number of data samples for the object in this
segment

                        nsamps=ob.(obname).nsamples+1;
                    else
                        nsamps=0;
                    end

                    if propsDataType==32
                        %String data type
```

```
                        propsValueLength=fread(fid,1,'uint32');

propsValue=convertToText(fread(fid,propsValueLength,'uint8=>char'))';
                        if propExists
                            if isfield(ob.(obname).(propsName),'cnt')
                                cnt=ob.(obname).(propsName).cnt+1;
                            else
                                cnt=1;
                            end
                            ob.(obname).(propsName).cnt=cnt;
                            ob.(obname).(propsName).value{cnt}=propsValue;
                            ob.(obname).(propsName).samples(cnt)=nsamps;
                        else
                            if strcmp(obname,'Root')
                                %Header data
                                ob.(obname).(propsName)=propsValue;
                            else
                                ob.(obname).(propsName).cnt=1;

ob.(obname).(propsName).value=cell(nsamps,1);       %Pre-allocation

ob.(obname).(propsName).samples=zeros(nsamps,1);    %Pre-allocation
                                if iscell(propsValue)

ob.(obname).(propsName).value(1)=propsValue;
                                else

ob.(obname).(propsName).value(1)={propsValue};
                                end
                                ob.(obname).(propsName).samples(1)=nsamps;
                            end
                        end
                    else
                        %Numeric data type
                        if propsDataType==68
                            %Timestamp data type

tsec=fread(fid,1,'uint64')/2^64+fread(fid,1,'uint64');  %time since Jan-1-
1904 in seconds
                            propsValue=tsec/86400+695422-5/24;  %/864000
convert to days; +695422 days from Jan-0-0000 to Jan-1-1904
                        else
                            matType=LV2MatlabDataType(propsDataType);
                            if strcmp(matType,'Undefined')
                                e=errordlg(sprintf('No MATLAB data type
defined for a ''Property Data Type'' value of ''%.0f''.',...
                                    propsDataType),'Undefined Property Data
Type');
                                uiwait(e)
                                fclose(fid);
                                return
                            end
                            if strcmp(matType,'uint8=>char')

propsValue=convertToText(fread(fid,1,'uint8'));
                            else
                                propsValue=fread(fid,1,matType);
                            end
                        end
                        if propExists
                            cnt=ob.(obname).(propsName).cnt+1;
```

```
                            ob.(obname).(propsName).cnt=cnt;
                            ob.(obname).(propsName).value(cnt)=propsValue;
                            ob.(obname).(propsName).samples(cnt)=nsamps;
                        else
                            ob.(obname).(propsName).cnt=1;
                            ob.(obname).(propsName).value=NaN(nsamps,1);
%Pre-allocation

ob.(obname).(propsName).samples=zeros(nsamps,1);        %Pre-allocation
                            ob.(obname).(propsName).value(1)=propsValue;
                            ob.(obname).(propsName).samples(1)=nsamps;
                        end
                    end

                end %'end' for the 'Property' loop
            end %'end' for the 'Objects' loop

        end

        %Process Raw Data
        if kTocRawData

            %Loop through each of the groups/channels and read the raw data
            fseek(fid,SegInfo(segCnt).DataStartPosn,'bof');
            for r=1:numel(index)

                cname=index(r).name;

                if index(r).newob && index(r).rawDataInThisSeg
                    index(r).newob=false;
                    ob.(cname).nsamples=0;
                end

                if index(r).rawDataInThisSeg

                    nvals=index(r).nValues;

                    if nvals>0

                        switch index(r).dataType

                            case 32     %String
                                %From the National Instruments web page
(http://zone.ni.com/devzone/cda/tut/p/id/5696) under the
                                %'Raw Data' description on page 4:
                                %String type channels are preprocessed for
fast random access. All strings are concatenated to a
                                %contiguous piece of memory. The offset of
the first character of each string in this contiguous
                                %piece of memory is stored to an array of
unsigned 32-bit integers. This array of offset values is
                                %stored first, followed by the concatenated
string values. This layout allows client applications to
                                %access any string value from anywhere in
the file by repositioning the file pointer a maximum of
                                %three times and without reading any data
that is not needed by the client.

                                StrOffsetArray=fread(fid,nvals,'uint32');
```

```
                                data=cell(1,nvals); %Pre-allocation
                                for dcnt=1:nvals
                                    if dcnt==1
                                        StrLength=StrOffsetArray(dcnt);
                                    else
                                        StrLength=StrOffsetArray(dcnt)-
StrOffsetArray(dcnt-1);
                                    end

data{1,dcnt}=char(convertToText(fread(fid,StrLength,'uint8=>char'))');
                                end
                                cnt=nvals;

                            case 68     %Timestamp
                                data=NaN(1,nvals);  %Pre-allocation
                                for dcnt=1:nvals

tsec=fread(fid,1,'uint64')/2^64+fread(fid,1,'uint64');   %time since Jan-1-
1904 in seconds
                                    data(1,dcnt)=tsec/86400+695422-5/24;
%/864000 convert to days; +695422 days from Jan-0-0000 to Jan-1-1904
                                end
                                cnt=nvals;

                            otherwise   %Numeric

matType=LV2MatlabDataType(index(r).dataType);
                                if strcmp(matType,'Undefined')
                                    e=errordlg(sprintf('No MATLAB data type
defined for a ''Raw Data Type'' value of ''%.0f''.',...
                                        index.dataType(r)),'Undefined Raw
Data Type');
                                    uiwait(e)
                                    fclose(fid);
                                    return
                                end

                                if strcmp(matType,'uint8=>char')
                                    [data,cnt]=fread(fid,nvals,'uint8');
                                    data=convertToText(data);
                                else
                                    [data,cnt]=fread(fid,nvals,matType);
                                end
                        end

                        if isfield(ob.(cname),'nsamples')
                            ssamples=ob.(cname).nsamples;
                        else
                            ssamples=0;
                        end

                        ob.(cname).data(ssamples+1:ssamples+cnt,1)=data;
                        ob.(cname).nsamples=ssamples+cnt;
                    end
                end

            end %'end' for the 'index' loop

        end
```

```
        %% Clean up preallocated arrays   (preallocation required for
speed)
        for y=1:numel(index)

            cname=index(y).name;

            if isfield(ob.(cname),'nsamples')

                nsamples=ob.(cname).nsamples;
                %Remove any excess from preallocation of data
                if nsamples>0
                    if numel(ob.(cname).data)>nsamples
                        ob.(cname).data(nsamples+1:end)=[];
                    end

                    %Remove any excess from preallocation of properties
                    proplist=fieldnames(ob.(cname));
                    for isaac=1:numel(proplist)
                        if isfield(ob.(cname).(proplist{isaac}),'cnt')
                            cnt=ob.(cname).(proplist{isaac}).cnt;
                            if
numel(ob.(cname).(proplist{isaac}).value)>cnt

ob.(cname).(proplist{isaac}).value(cnt+1:end)=[];

ob.(cname).(proplist{isaac}).samples(cnt+1:end)=[];

ob.(cname).(proplist{isaac})=rmfield(ob.(cname).(proplist{isaac}),'cnt');
                            end
                        end
                    end

                end
            end
        end %'end' for the 'groups/channels' loop

    end %'end' for the 'Segment' loop

    fclose(fid);

    %% Assign the outputs
    ConvertedData(fnum).FileNameShort=FileNameShort;
    ConvertedData(fnum).FileFolder=FileFolder;
    ConvertedData(fnum).Data=postProcess(ob);

    Index(fnum).FileNameShort=FileNameShort;
    Index(fnum).FileFolder=FileFolder;
    Index(fnum).Data=index;

    %% Save the MAT file
    if SaveConvertedFile
        MATFileNameShort=sprintf('%s.mat',FileNameNoExt);
        MATFileNameLong=fullfile(FileFolder,MATFileNameShort);
        try
            save(MATFileNameLong,'ConvertedData','Index','ConvertVer')
            fprintf('\n\nConversion complete (saved in
''%s'').\n\n',MATFileNameShort)
        catch exception
```

```
            fprintf('\n\nConversion complete (could not save
''%s'').\n\t%s: %s\n\n',MATFileNameShort,exception.identifier,...
                exception.message)
        end
    else
        fprintf('\n\nConversion complete.\n\n')
    end

end %'end' for the 'Number of Files' loop

end


function DataStructure=postProcess(ob)

%Modified to return all information stored in the TDMS file to include
name, start time, start time offset, samples
%per read, total samples, unit description, and unit string.  Also provides
event time and event description in
%text form

DataStructure.Root=[];
DataStructure.MeasuredData.Name=[];
DataStructure.MeasuredData.Data=[];
DataStructure.Events.Name=[];
DataStructure.Events.Data=[];

varNameMask='';
cntData=1;
cntEvent=1;

GroupNames=fieldnames(ob);

for i=1:numel(GroupNames)
    cname=GroupNames{i};
    if strcmp(cname, 'Root')
        DataStructure.Root=ob.(cname);
    end
    if isfield(ob.(cname),'data')
        if strcmp(varNameMask,'Events')
            DataStructure.Events(cntEvent).Name=cname;

            if strcmp(DataStructure.Events(cntEvent).Name,'Description')
                event_string=char(ob.(cname).data');
                seperator=event_string(1:4);
                locations=findstr(seperator, event_string);
                num_events=max(size(locations));
                for j=1:num_events
                    if j<num_events

DataStructure.Events(cntEvent).Data(j,:)=cellstr(event_string(locations(j)+
4:locations(j+1)-1));
                    else

DataStructure.Events(cntEvent).Data(j,:)=cellstr(event_string(locations(j)+
4:max(size(event_string))));
                    end
                end
            else
```

```
                DataStructure.Events(cntEvent).Data=ob.(cname).data;
            end
            cntEvent=cntEvent+1;

        else
            DataStructure.MeasuredData(cntData).Name=cname;
            DataStructure.MeasuredData(cntData).Data=ob.(cname).data;

DataStructure.MeasuredData(cntData).Total_Samples=ob.(cname).nsamples;
            if isfield(ob.(cname),'wf_start_time')

DataStructure.MeasuredData(cntData).Start_Time=ob.(cname).wf_start_time.val
ue;

DataStructure.MeasuredData(cntData).Start_Time_Offset=ob.(cname).wf_start_o
ffset.value;

DataStructure.MeasuredData(cntData).Sample_Rate=ob.(cname).wf_increment.val
ue;

DataStructure.MeasuredData(cntData).Samples_Per_Read=ob.(cname).wf_samples.
value;
            end
            if isfield(ob.(cname),'NI_UnitDescription')

DataStructure.MeasuredData(cntData).Units_Decription=char(ob.(cname).NI_Uni
tDescription.value)';
            else

DataStructure.MeasuredData(cntData).Units_Decription='Unknown';
            end
            if isfield(ob.(cname),'unit_string')

DataStructure.MeasuredData(cntData).Unit_String=char(ob.(cname).unit_string
.value)';
            else
                DataStructure.MeasuredData(cntData).Unit_String='Unknown';
            end
            cntData = cntData + 1;
        end
    end

end %'end' for the 'groups/channels' loop

end



function
[FixedText,TruncFieldName,ValidFieldName]=fixcharformatlab(textin)
    %Private Function to remove all text that is not MATLAB variable name
compatible

    OrigText=textin;

    %First character cannot be a space.  If it is, replace with 'x'.
    if isspace(textin(1))
        textin(1)='x';
    end

    textin=strrep(textin,'_0''/''','_0_');
```

```
    textin=strrep(textin,'''','');
    textin=strrep(textin,'\','');
    textin=strrep(textin,'/Untitled/','');
    textin=strrep(textin,'/','.');
    textin=strrep(textin,'-','');
    textin=strrep(textin,'?','');
    textin=strrep(textin,' ','_');
    textin=strrep(textin,'.','');
    textin=strrep(textin,'[','_');
    textin=strrep(textin,']','');
    textin=strrep(textin,'%','');
    textin=strrep(textin,'#','');
    textin=strrep(textin,'(','');
    textin=strrep(textin,')','');
    textin=strrep(textin,':','');
    textin=strrep(textin,'^','_');
    textin=strrep(textin,'Ä','Ae');
    textin=strrep(textin,'ä','ae');
    textin=strrep(textin,'Ö','Oe');
    textin=strrep(textin,'ö','oe');
    textin=strrep(textin,'Ü','Ue');
    textin=strrep(textin,'ü','ue');
    textin=strrep(textin,'ß','ss');
    textin=strrep(textin,'é','e');
    textin=strrep(textin,'°','deg');

    %Check for a case that is not explicitly listed above
    InvalidCharIndices=regexp(textin,'[^A-Za-z_0-9]');      %NOT A thru Z,
a thru z, underscore or 0 thru 9
    if ~isempty(InvalidCharIndices)
        fprintf('\n')

        InvalidChar=unique(cellstr(textin(InvalidCharIndices)'));

        if numel(InvalidChar)==1
            fprintf(['\nA valid replacement character has not been defined
in the ''fixcharformatlab'' private function\n\t'...
                '(within %s.m) for the invalid character ''%s'' contained
within the\n\t''%s'' Group/Channel name.\n\t'...
                'It has been replaced with an
''_''.\n'],mfilename,char(InvalidChar),OrigText)
        else
            for i=1:numel(InvalidChar)
                switch i
                    case 1
                        MyString=sprintf('''%s''',InvalidChar{i});
                    case numel(InvalidChar)
                        MyString=sprintf('%s &
''%s''',MyString,InvalidChar{i});
                    otherwise
                        MyString=sprintf('%s,
''%s''',MyString,InvalidChar{i});
                end
            end
            fprintf(['\nValid replacement characters have not been defined
in the ''fixcharformatlab'' private function\n\t'...
                '(within %s.m) for the invalid characters %s contained
within the\n\t''%s'' Group/Channel name.\n\t'...
                'They have been replaced with an
''_''.\n'],mfilename,MyString,OrigText)
        end
```

```
        textin(InvalidCharIndices)='_';
    end

    %Ensure that the name isn't too long
    maxid=namelengthmax;
    if numel(textin)<=maxid
        FixedText=textin;
        TruncFieldName=false;
    else
        FixedText=textin(1:maxid);
        TruncFieldName=true;
    end

    %Check for a valid fieldname
    ValidFieldName=isvarname(FixedText);
    if ~ValidFieldName
        %Check to see if maybe the issue is the first character is not a
letter.  If it is, then add an 'a' to the front
        %of the string.
        if ~isletter(FixedText(1))
            if TruncFieldName || numel(FixedText)>=(maxid-1)
                FixedText=sprintf('a%s',FixedText(1:end-1));
            else
                FixedText=sprintf('a%s',FixedText);
            end
        end
    end
    %Confirm whether or not the issue has been fixed.
    ValidFieldName=isvarname(FixedText);

end


function matType=LV2MatlabDataType(LVType)
%Cross Refernce Labview TDMS Data type to MATLAB

    switch LVType
        case 0   %tdsTypeVoid
            matType='';
        case 1   %tdsTypeI8
            matType='int8';
        case 2   %tdsTypeI16
            matType='int32';
        case 3   %tdsTypeI32
            matType='int32';
        case 4   %tdsTypeI64
            matType='int64';
        case 5   %tdsTypeU8
            matType='uint8';
        case 6   %tdsTypeU16
            matType='uint16';
        case 7   %tdsTypeU32
            matType='uint32';
        case 8   %tdsTypeU64
            matType='uint64';
        case 9  %tdsTypeSingleFloat
            matType='float32';
        case 10  %tdsTypeDoubleFloat
            matType='float64';
```

```
        case 11  %tdsTypeExtendedFloat
            matType='';
        case 32  %tdsTypeString
            matType='uint8=>char';
        case 33  %tdsTypeBoolean
            matType='bit1';
        case 68  %tdsTypeTimeStamp
            matType='bit224';
        otherwise
            matType='Undefined';
    end

end

function text=convertToText(bytes)
%Convert numeric bytes to the character encoding localy set in MATLAB (TDMS
uses UTF-8)

    text=native2unicode(bytes,'UTF-8');
end
```

APPENDIX E

MATLAB CODES of FSdetection.m FUNCTION

```
function[FS_Location,F,FS_LocationRealY,PeakY,RealTime,Number_of_Footsteps]=FS
detection(N,A,T,NRaw,FS,AA)
% ////////////////////////////////////////////
% --------University of Gaziantep------------
%       Control and Command Systems
%              Master Thesis
%       Footstep Detection Algorithm
% ///////////////////////////////////////////
%
%
% N is the number of simout samples
% A is the y-axis of simout samples
% T is the x-axis of simout samples
% NRaw is the number of samples of the Raw data
%
p=0;
m=1;
n=1;
k=0;
SNR=0.07;
FSwidth_min=70;
FSwidth_max=800;
c1=100; %min distance between two peaks
c2=1500; %max distance between two peaks
TotalNumbers_of_Peak=0;
Peak_start=0;
Peak_end=0;
Flat_start=0;
Flat_end=0;
j=0;

for i=1:N %First stage: This loop calculates peaks as the defined limits.
    if A(i)>SNR
        j=j+1;
        p=p+1;
        k=0;
        if p==1
            m=m+1;
            n=i;
            PA(j)=A(i);
            PT(j)=T(i);
            WT(m)=1;

        elseif p==2
            PA(j)=A(i);
            PT(j)=T(i);
            WT(m)=WT(m)+1;
            Peak_start(m)=n;
            Peak_end(m)=i;
            p=1;

        else
        end
    else
        p=0;
        j=i;
```

```
        PA(j)=0;
        PT(j)=T(i);
        k=k+1;
        w0(m)=k;
        Flat_start(m)=i-k+1;
        Flat_end(m)=i
    end
end
%
% Second stage: This loop calculates the Footpeaks
% Above amplitude is okey, here width of peak considered. Both of them is
% ok,then you can use them at detection calculations as footstep peak in
% stage fourth.
%
s=1;
for s=1:m
     if WT(s)>FSwidth_min
        if WT(s)<FSwidth_max
            F(s)=1;
            TotalNumbers_of_Peak=TotalNumbers_of_Peak+1;
        else
            F(s)=0;
        end
    else
        F(s)=0;
    end
end
%
% Third stage: This loop calculates the max peak values for each footstep
%
fy=1;
for fy=2:m
        P1=Peak_start(fy);
        P2=Peak_end(fy);
        PeakY(fy)=max(PA(P1:P2)); % Y-axis (Amplitude)
        for fx=P1:P2
            ValX=fx;
            if PA(ValX)==PeakY(fy)
                PeakX(fy)=ValX; % X-axis (Time)
                RealTime(fy)=(PeakX(fy)/FS)*(NRaw/N);
            end
        end


end
%
% Forth stage: This loop calculates the Footsteps,
% Here we look the time between two peaks.CadencyFrequency 1.5-2.5 Hz
%
b=0;
for b=1:m
    FS_Location(b)=0;
end
ADIM=0;
g=0;
z=0;
```

```
for f=1:m-1
   if (F(f)==1)&(F(f+1)==1)
            if w0(f)>c1
                    if w0(f)<c2
                        ADIM=ADIM+1;
                        FS_Location(f)=f;
                        FS_Location(f+1)=f+1;
                            if ADIM==1
                                z=z+1;
                            else
                                    if ADIM>=2
                                        Number_of_Footsteps(z)=ADIM+1;
                                    end
                            end
                    else
                     if ADIM==1
                         FS_Location(f)=0;
                         FS_Location(f-1)=0;
                         ADIM=0;
                     else
                         ADIM=0;
                     end

                    end

            else
                if ADIM==1
                    FS_Location(f)=0;
                    FS_Location(f-1)=0;
                    ADIM=0;
                else
                    ADIM=0;
                end
            end
   else
        if ADIM==1
           FS_Location(f)=0;
           FS_Location(f-1)=0;
           ADIM=0;
        else
           ADIM=0;
        end
    end
end
%
% Final stage: Graphical Representation
%
c=0;
for c=1:m
    if FS_Location(c)==0;
           FS_LocationRealX(c)=RealTime(c);
           FS_LocationRealY(c)=0;
    else
           FS_LocationRealX(c)=RealTime(c);
           FS_LocationRealY(c)=PeakY(c);
    end
end
```

```
figure('Name','University of Gaziantep','NumberTitle','off'), clf
subplot(2,1,1)
bar(RealTime,PeakY,0.6,'b')
set(gca,'xlim',[1 NRaw/FS])
title('Total Detected Peaks')
xlabel('Time [seconds]')
ylabel('Amplitude')
hold on
subplot(2,1,2)
bar(FS_LocationRealX,FS_LocationRealY,0.6,'r')
set(gca,'xlim',[1 NRaw/FS])
title(['The Number of Footsteps: ',int2str(
Number_of_Footsteps)],'Color','r')
xlabel('Time [seconds]')
ylabel('Amplitude')
end
```