

IMPROVED SIGNAL EXTENSION METHODS
FOR WAVELET SMOOTHING

By

JONATHAN MARK HYNSON

Bachelor of Science

Oklahoma State University

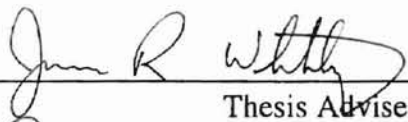
Stillwater, Oklahoma

1993

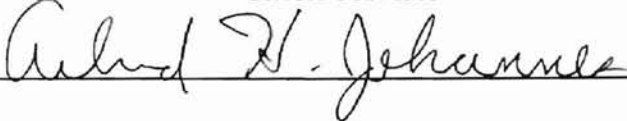
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1996

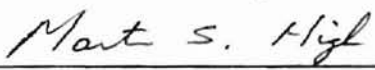
IMPROVED SIGNAL EXTENSION METHODS
FOR WAVELET SMOOTHING


Thesis Approved:



Thesis Adviser







Dean of the Graduate College

PREFACE

This research work has developed as a subset of pattern-based data analysis research. Pattern-based data analysis requires a relatively smooth trend to provide accurate and useful classification and analysis of the chemical process. Yet, most sensor signals contain both noise and measurement error which makes pattern classification difficult. Wavelet decomposition provides an excellent trend extraction technique. The fundamental sensor trend is extracted by eliminating the process noise and measurement error. The resulting smoothed sensor signal can be used for pattern-based classification and analysis.

Wavelet representation is similar to Fourier representation, however, wavelets offer several distinct advantages over Fourier representation. The implementation of wavelet decomposition causes end-distortion of the extracted trend. To overcome this end-distortion, an extension technique is used. The sensor signal is extended, then smoothed with the wavelet decomposition. The extended smoothed trend is removed to leave a smoothed version of the original sensor signal. The objective of this thesis is to develop an adaptive extension technique to eliminate the end-distortion of the

extracted trend. The adaptive extension technique can be applied to many different types of process sensors.

This thesis proposes five new adaptive extension techniques. Each of the techniques is evaluated to determine which technique performs the best. The technique that is recommend as the best extension technique is the simplest to implement and has the least amount of computational requirements.

There are several people that I would like to thank for their work and encouragement as I completed my work for a Master of Science in Chemical Engineering at Oklahoma State University. First and foremost I would like to thank my parents Larry and Kathy Hynson. They have provided more than enough love and encouragement as they raised me. Their numerous words of encouragement have provided me with the endurance to continue. I would also like to thank my siblings: Janette, Jennifer, and Jason. They are always dear to my heart.

I would like to thank Dr. Rob Whiteley. As my advisor he has unselfishly put much time and effort into my research and, specifically, this thesis. I would also like to thank Dr. AJ Johannes and Dr. Martin High for serving on my thesis committee. Additionally, I would like to thank Bruce Colgate and Jack DeVeux at Phillips Petroleum for their work on the research project.

I would also like to thank Mike Dodson and Keith Crone for their spiritual guidance and encouragement while I was a student at OSU. There are several fellow students who have offered words of encouragement and for advice which I will be eternally grateful. I would like to thank Paul and Megan Belcher for their continual encouragement to finish this thesis. I would like to thank Brian Neely and Greg Holland. As fellow graduate students, they have helped me in so many ways.

In conclusion, I will summarize my research experience with a quote from King Solomon. He stated, "Of making many books there is no end, and much study wearies the body." [Bible, Ecclesiastes 12:12b]

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
1.1 Background	1
1.2 Problem Addressed in Thesis	5
1.3 Organization of Thesis	8
II. WAVELETS BACKGROUND	9
2.1 Introduction	9
2.2 Fourier Representation	9
2.3 Wavelet Representation	13
2.4 Wavelet Decomposition	19
2.5 Trend Extraction	22
2.6 Implementation of Trend Extraction	25
2.7 Chapter Summary.....	29
III. PROPOSED EXTENSION METHODS	31
3.1 Introduction	31
3.2 Raghavan NET2 Technique	36
3.3 Dead-zone	45
3.4 Adaptive NET2 Extension Technique	50
3.5 Square Root Objective Function Extension Technique	51
3.6 First Point Past Dead-Zone Extension Technique	52
3.7 Fuzzy Tie-Point Extension Technique	52
3.8 Weighted Fuzzy Tie-Point Extension Technique	58
3.9 Chapter Summary.....	62
IV. Performance of Proposed Extension Techniques	65
4.1 Introduction	65
4.2 Ability to Minimize End-Distortion	65
4.3 Robustness Evaluation	83
4.4 Sampling Rate Evaluation	88
4.5 Evaluation of Composite Results	90

Chapter	Page
V. CONCLUSIONS	94
5.1 Conclusions	94
5.2 Recommendations for Future Work	95
LIST OF REFERENCES	97
APPENDIXES	101
APPENDIX A	101
A.1 Fuzzy Logic	101
A.2 Approximate Reasoning	104
A.3 Defuzzification	107
APPENDIX B	109
B.1 Computer Code for Adaptive NET2 Extension Technique	109
B.2 Computer Code for Square Root Objective Function Extension Technique	110
B.3 Computer Code for First Point Past Dead-Zone Extension Technique	111
B.4 Computer Code for Fuzzy Tie-Point Extension Technique	111
B.5 Computer Code for Weighted Fuzzy Extension Technique	127

LIST OF TABLES

Table	Page
3-I. Rule-base used to rate the tie-point for Fuzzy Tie-Point extension technique	55
3-II. Rule-base for rating each possible tie-point for the Weighted Fuzzy Tie-Point extension technique	60
4-I. Evaluation of extension techniques to minimize end-distortion for the flow meter sensor	74
4-II. Evaluation of extension techniques to minimize end-distortion for the temperature sensor	79
4-III. Evaluation of extension technique for robustness on flow meter	84
4-IV. Evaluation of extension technique for robustness on temperature meter	87
4-V. Sampling rate comparison of extension techniques	90

LIST OF FIGURES

Figure	Page
1-1. Process sensor noise from two different process sensors	2
1-2. Extracted sensor trends for two different process sensors	3
1-3. Process sensor pattern \bar{P} generated from smoothed trend plots	4
1-4. Process sensor pattern generated from original sensor signals. The sampled representation for the upper trend plot is clearly not representative of the true trend	5
1-5. Example of end distortion when using discrete wavelet transforms	7
2-1. Fourier decomposition of a signal and the reconstruction to time domain	12
2-2. (a) Original wavelet function, (b) translated wavelet function, and (c) dilated and translated wavelet function	15
2-3. Daubechies family wavelet and scaling function with order = 1	17
2-4. Daubechies family wavelet and scaling function with order = 3	18
2-5. Daubechies family wavelet and scaling function with order = 6	19
2-6. Decomposition pyramid for wavelet representation	23
2-7. Decreasing number of blurred coefficients used to represent a signal at different levels of decomposition	24
2-8. Number of terms used to calculate the convolution term at each index	28

Figure	Page
3-1. Original sensor signal to be smoothed	32
3-2. Extended signal prior to wavelet smoothing	33
3-3. Extended signal after wavelet smoothing	34
3-4. The original sensor signal with the extracted trend	35
3-5. The inverted symmetric extension technique. The extended signal is generated from the windowed signal	37
3-6. Inverted symmetric extension technique with high tie-point	38
3-7. Inverted symmetric extension technique with a low tie-point	39
3-8. Smoothed signal using inverted symmetric extension technique with high tie-point	40
3-9. Smoothed signal using inverted symmetric extension technique with low tie-point	41
3-10. Smoothed signal using inverted symmetric extension technique with correct tie-point	42
3-11. Sensor signal showing the periodic frequency of the sensor noise	45
3-12. Power spectrum of sensor signal	46
3-13. Extended signal with dead-zone less than dominant period of the sensor signal. Notice the extended signal is moved up	48
3-14. Extended signal with dead-zone equal to dominant period of the sensor signal. Notice the extended signal is very close to the current sensor values	49
3-15. Fuzzy Tie-Point extension technique membership function for the mean squared deviation	54
3-16. Fuzzy Tie-Point extension technique membership function for the index number	54

Figure	Page
3-17. Fuzzy Tie-Point extension technique tie-point rating membership function	56
3-18. Weighted Fuzzy Tie-Point extension technique membership function for the mean squared deviation	59
3-19. Weighted Fuzzy Tie-Point extension technique membership function for the index number	59
3-20. Weighted Fuzzy Tie-Point extension technique rating of each point	61
4-1. The test standard: Extracted trend as smoothed in the middle of the signal	66
4-2. Compared trend: Extracted trend as smoothed at the end of the signal	67
4-3. Flow meter sensor signal for the seven days. Sample period is one minute	70
4-4. Flow meter sensor power spectrum plot. 1000 sensor values used with a sample period of one minute	71
4-5. Temperature sensor signal for the seven days. Sample period is one minute	72
4-6. Temperature sensor power spectrum plot. 1000 sensor values used with a sample period of one minute	73
4-7. Worst case for smoothed flow meter sensor signal for the seven days using First Point Past Dead-Zone extension technique	77
4-8. Worst case for smoothed flow meter sensor signal for the seven days using Weighted Fuzzy Tie-Point extension technique	78
4-9. Worst case for smoothed temperature sensor signal for the seven days using First Point Past Dead-Zone extension technique	81

Figure	Page
4-10. Worst case for smoothed temperature sensor signal for the seven days using Weighted Fuzzy Tie-Point extension technique	82
4-9. Example of Raghavan NET2 extension technique	92
4-10. Example of First Point Past Dead-Zone extension technique	93
A-1. Graphical representation of fuzzy set of tall people	103
A-2. Fuzzy logic implication with one condition	104
A-3. Example of fuzzy logic implication with one condition	105
A-4. Fuzzy logic implication with one condition	105
A-5. Fuzzy logic two conditions implication rule-base	106
A-6. General form of Mamdani's implication rule for two conditions	106

NOMENCLATURE

Glossary

Dead-Zone	Minimum number of sensor points included in the calculation of the tie-point
NET2	New Extension Technique 2 developed by V. Raghavan
Tie-Point	Point where the extended signal is appended to the current sensor signal
Threshold	Maximum number of sensor points included in the calculation of the tie-point

Roman Letters

a	Fourier coefficients for cosine basis function or scaling function coefficients
b	Fourier coefficients for sine basis function or wavelet function coefficients
c_k	Scaling function coefficients
d_k	Wavelet coefficients
f	Frequency
f_s	Sampling rate of the signal
G	Highpass filter
H	Lowpass filter
i	Index number (integer)
j	Index number (integer)

k	Index number (integer)
m	Dimension of vector
M	Mean squared deviation
n	Dimension of vector
N	Integer value representing the total number of points
T	Period of frequency
\bar{X}	Vector representation
\bar{x}	Mean
\bar{x}_{TP}	Tie-point

Greek Letters

α	Threshold number
β	Dead-zone size
γ	Objective function used in square root extension technique to calculate the tie point
$\mu(x_j)$	Fuzzy membership value for element x_j
$\phi(x)$	Scaling function
$\psi(x)$	Wavelet

Chapter One: Introduction

1.1 Background

New computer applications for process monitoring and control continue to be developed. One application being pioneered at Oklahoma State University is the utilization of the computer for advanced process monitoring, specifically pattern-based sensor data analysis [Raghavan, 1995; Raghavan and Whiteley, 1993; Whiteley and Davis 1994; Whiteley and Davis 1993]. The methods developed by Whiteley *et al.* use trend data for one or more sensors to monitor a chemical process. The trend patterns for the sensors are combined to develop a fingerprint of the current operating condition of the process. The fingerprint is then compared with historical plant operating data to classify the current process conditions. This information can be made available for monitoring purposes or used for other applications [Anderson, 1993 and Sinha, 1995].

Pattern-based process monitoring can be broken up into three distinct tasks: individual sensor trend extraction, composite pattern (fingerprint) formation, and pattern classification. The three steps are performed sequentially. The first step is trend extraction. During trend extraction, the fundamental trends associated with each of the sensors which make up the process fingerprint are identified. Almost all sensors are affected by both measurement and process noise. These effects make pattern classification very difficult. The goal of trend extraction is to reduce or eliminate as

much noise and interference as possible. The following figure illustrates an example of the process noise in two different sensors.

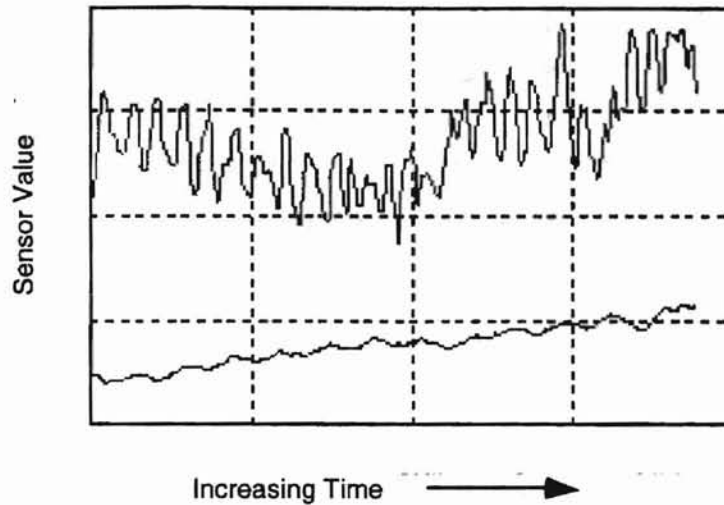


Figure 1-1: Process sensor noise from two different process sensors.

Both sensors have noise which can make the pattern classification more difficult. The noise in the upper trend plot in Figure 1-1 is characterized by a larger amplitude and a higher frequency. The lower trend plot exhibits noise with a smaller amplitude and a lower frequency. In both cases a technique is needed to extract the fundamental trend of the signal. In essence, trend extraction attempts to mimic human intuition to pick out the underlying process trend. The following figure shows the extracted trend from the process sensors.

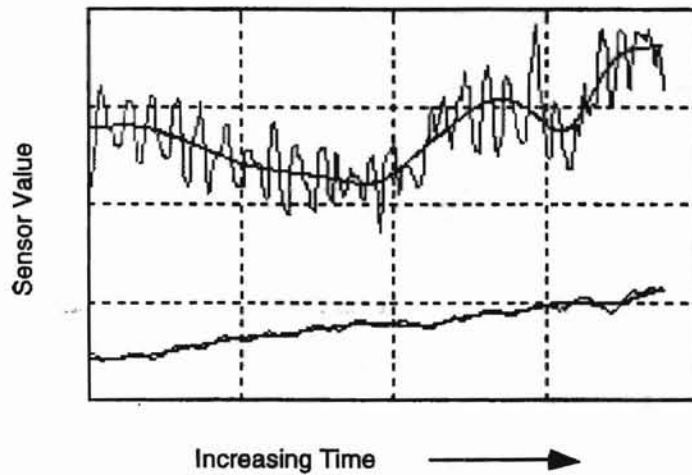


Figure 1-2: Extracted sensor trends for two different process sensors.

The extracted sensor trends are then used for the second step of pattern-based data analysis. The second step is pattern development. During this step a vector representation of the extracted trend patterns is formed to provide a fingerprint of the process signal. The first trend is sampled within a fixed length pattern window to form a vector of sampled sensor values. The next sensor trend is sampled and the values are concatenated to the first sensor trend vector. This continues until all the sensor trends have been sampled and combined to form one vector. This is the process fingerprint. This fingerprint is then used to classify the process conditions.

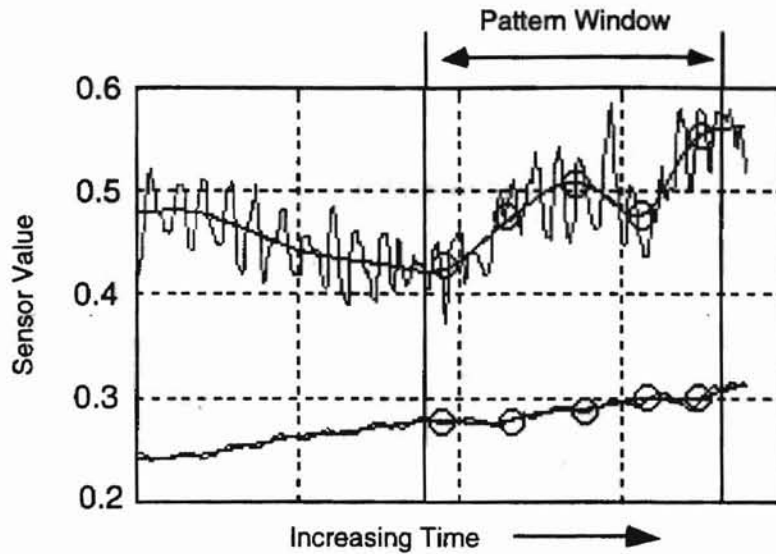


Figure 1-3: Process sensor pattern \bar{P} generated from smoothed trend plots.

For Figure 1-3, the fingerprint or process pattern would be represented as the vector $\bar{P} = [0.28 \ 0.27 \ 0.29 \ 0.30 \ 0.30 \ 0.43 \ 0.47 \ 0.52 \ 0.48 \ 0.56]^T$.

The last step is pattern classification. The fingerprint representing the process pattern can be classified using any suitable pattern recognition method [Whiteley and Davis 1994; Whiteley and Davis 1993]. The classification is based on historical data. The current sensor pattern is compared to historical patterns and a match is made.

1.2 Problem Addressed in Thesis

Trend extraction is a key step in pattern-based sensor data analysis. The extracted trend must maintain the fundamental sensor trend while eliminating noise and interference. Otherwise, the sampled pattern vector as illustrated in the following figure is not representative of the true trend. Pattern classification becomes more robust as noise effects are eliminated.

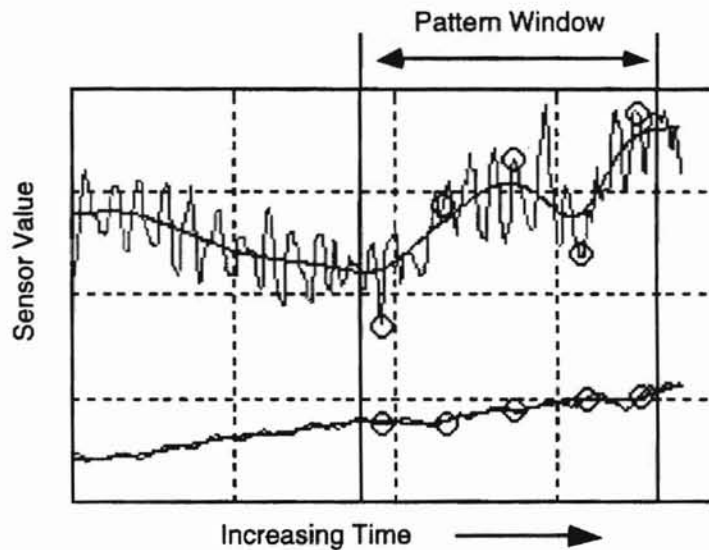


Figure 1-4: Process sensor pattern generated from original sensor signals. The sampled representation for the upper trend plot is clearly not representative of the true trend.

Discrete wavelet transforms [Mallat, 1989; Daubechies, 1990; Rioul and Duhamel, 1992; Walter, 1992; Chui, 1992a] can be used to extract the fundamental trend while eliminating process noise. The wavelet transforms provide an ability to adjust the noise filtering. When a higher level of smoothing is used, then the wavelet transforms

provide a trend with less of the high frequency content of the sensor signal. Alternatively, a lower level of smoothing can be used where more of the high frequency signal is retained in the extracted trend. This thesis does not address selection of the appropriate level of smoothing but focuses on an implementation problem which occurs after the degree of smoothing has been fixed.

Wavelet transforms have a problem when applied to real-time applications. Unless special steps are taken as described in this thesis, application of the wavelet transforms to a real-time sensor signal will cause end-distortion at each end of the signal. This end-distortion may cause the extracted trend to deviate from the true underlying trend. End-distortion is a mathematical consequence of implementing wavelet smoothing using a traditional convolution filtering approach. Figure 1-5 illustrates distortion of the extracted trend at the real-time end of the process signal. The right side of the smoothed sensor signal exhibits end-distortion caused by the wavelet transforms. The smoothed or extracted trend indicates the process measurement is declining when, in fact, it appears that the measurement is actually leveling out. Correct classification of the process state cannot be made unless the extracted trend is also correct.

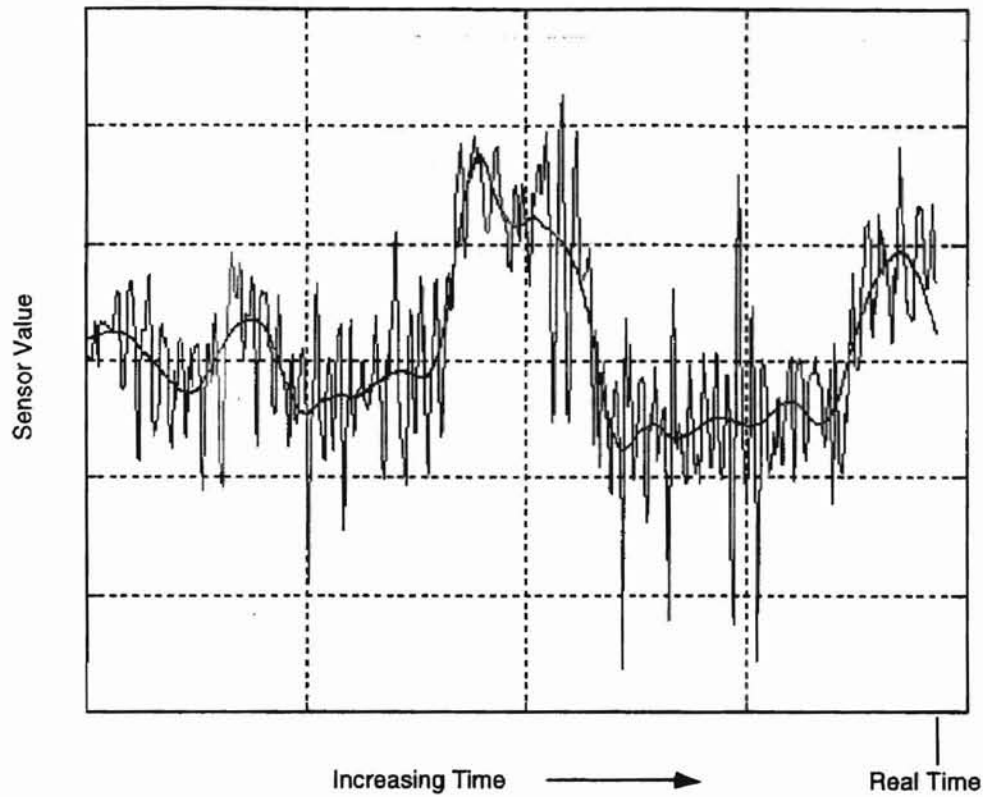


Figure 1-5: Example of end distortion when using discrete wavelet transforms.

This work proposes five techniques to eliminate or minimize the end-distortion associated with wavelet transforms. These techniques are based on the inverted symmetric extension technique originally proposed by Raghavan [Raghavan, 1995]. Each of the five proposed techniques are discussed in detail. Then the five techniques are evaluated to determine which technique provides the best overall trend extraction performance.

1.3 Organization of Thesis

This thesis has five chapters. The second chapter covers the implementation of wavelet transforms and the problem associated with using the convolution filtering approach. All five of the proposed extension techniques are presented in the third chapter. The performance evaluation of all the extension techniques is presented in the fourth chapter. The last chapter summarizes the conclusions for this work and includes recommendations for future work concerning trend extraction using wavelet transforms.

Chapter Two: Wavelets Background

2.1 Introduction

This chapter presents the mathematical background on Fourier signal decomposition and wavelet signal decomposition. The Fourier decomposition is discussed first as an introduction to concepts associated with the more complex wavelet decomposition used for signal smoothing. The procedures for implementing both the Fourier and wavelet decomposition are also provided in this chapter. The last part of this chapter covers the application of Fourier and wavelet decomposition for trend extraction.

2.2 Fourier Representation

A Fourier decomposition converts a time-series signal into a sum of unique sine and cosine functions [Tolstov, 1962 and Weaver, 1983]. The sine and cosine functions are the basis functions for the Fourier series representation. These basis functions are periodic and are referred to as global rather than local basis functions since they are defined for all time. The general form of the discrete Fourier series is as follows:

$$x(j) = \sum_{i=0}^{N-1} \left(a_i \cos \frac{2\pi ij}{N} + b_i \sin \frac{2\pi ij}{N} \right) \quad (2.1).$$

The sensor signal $\bar{\mathbf{X}} = [x(0) \ x(1) \ x(2) \ \dots \ x(N-1)]$ has N elements. The $x(j)$ terms are the sensor measurements at the j^{th} sampling instant. The sensor signal is indexed from $j = 0$ to $j = N-1$. The Fourier coefficients are represented by the a_i and b_i terms and are calculated from the following formula:

$$a_i = \frac{2}{N} \sum_{j=0}^{N-1} x(j) \cos\left[\frac{2\pi ij}{N}\right] \quad (2.2a).$$

$$b_i = \frac{2}{N} \sum_{j=0}^{N-1} x(j) \sin\left[\frac{2\pi ij}{N}\right] \quad (2.2b).$$

There is no b_0 term,
 with $a_0 = 2a_0$ as calculated from above
 and for an even number of N , $a_{N/2+1} = 2a_{N/2+1}$ as calculated from above.

For a signal with an even number of N sensor points, there are a total of N Fourier coefficients which can be used to represent the sensor signal; $N/2+1$ cosine coefficients and $N/2-1$ sine coefficients.

The Fourier coefficients provide a frequency domain representation of the original time domain signal. The Fourier coefficients indicate the amplitude or contribution of an individual frequency to the original time domain signal. The Fourier transform is an exact one-to-one transform [Morrison, 1994]. When converted to the frequency domain, there is exactly one set of Fourier coefficients that represent a sensor signal. Likewise, when the Fourier coefficients are used to reconstruct a signal using Equation 2.1, the resulting signal $\bar{\mathbf{X}}$ is unique.

The alternative representation provided by a Fourier decomposition is illustrated by the following example. Consider the sensor signal $\bar{X} = [x(0) x(1) x(2) x(3) x(4) x(5)] = [3 7 6 4 9 2]$. The Fourier representation for \bar{X} can be generated from Equations 2.2a and 2.2b. The Fourier coefficients are calculated as following:

$$\begin{aligned} a_0 &= 5.17 \\ a_1 &= -1.33 \\ b_1 &= 0.58 \\ a_2 &= -1.67 \\ b_2 &= 2.31 \\ a_3 &= 0.83 \end{aligned}$$

The original sensor signal can be recovered from the Fourier coefficients by application of Equation 2.1. The two vectors provide alternative representations for the same signal \bar{X} with the time domain representation given by [3 7 6 4 9 2] and the frequency domain representation given by [5.17 -1.33 0.58 -1.67 2.31 0.83].

The following figure provides graphical interpretation of Equation 2.1 for the previous example. Each of the terms from Equation 2.1 is plotted with the appropriate coefficient. The higher terms represent the higher frequency components of the signal. At each integer point along the x-axis the values of the periodic functions can be summed to reconstruct the sensor measurement at that sampling instant. For example, at $j = 0$ the values of each of the six basis functions are 5.17, -1.33, 0, -1.67, 0, 0.83. These all sum to 3 which is the value of the first term of the original signal. The conversion back to the time domain results in the reconstruction of the original signal $\bar{X} = [3 7 6 4 9 2]$.

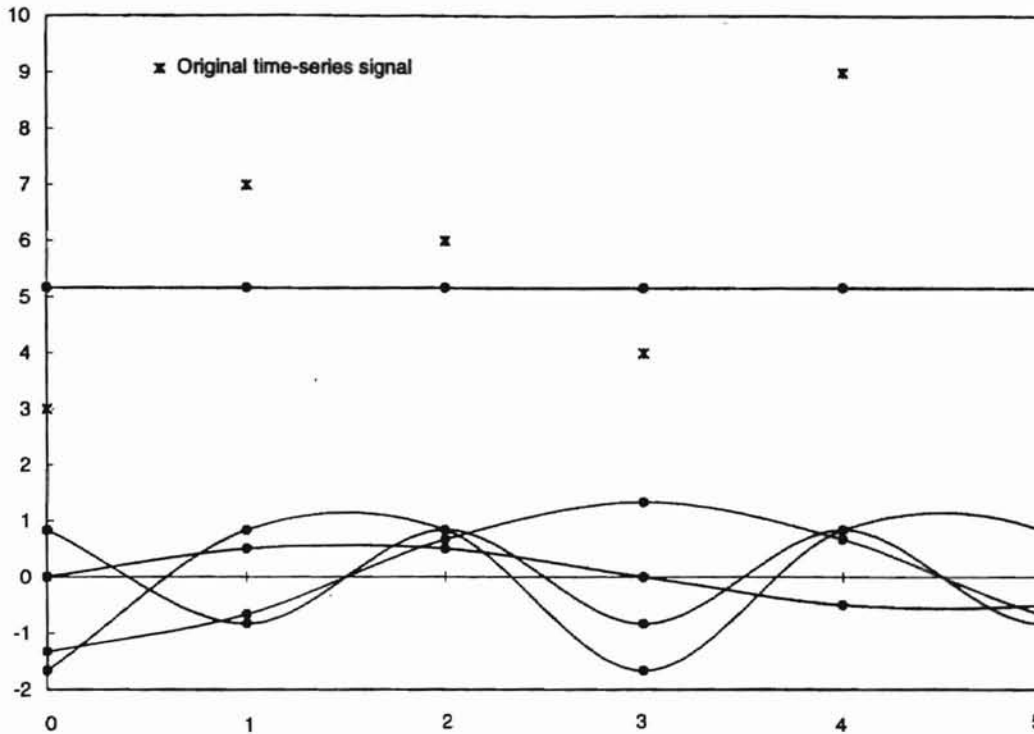


Figure 2-1: Fourier decomposition of a signal and the reconstruction to time domain.

In practice, Equation 2.2 is seldom used to calculate the Fourier coefficients for a signal. Instead, the Fast Fourier Transform (FFT) is typically employed [Press et al., 1992; Gray and Goodman, 1995]. The FFT is a very efficient numerical method to calculate the individual Fourier coefficients for long signals. In order to utilize the Fast Fourier Transforms, the signal length must be a power of 2.

In summary, the Fourier decomposition provides a method to generate alternative representation for a time-series signal. The representation involves two basis functions and provides the ability to decompose a discrete sensor signal into a unique set of

individual frequency components. This result is exploited in the next chapter when it becomes necessary to identify the dominant frequency in a sensor measurement prior to signal extension and trend extraction.

2.3 Wavelet Representation

This section is a brief introduction to wavelet representation. The discussion is limited to provide only a basic understanding and appreciation for the need and development of the extension techniques which are presented in the next chapter. For more detailed and technical information on wavelet transforms, there are numerous articles and books available [Mallat, 1989; Mallat, 1989a; Mallat, 1989b; Daubechies, 1990; Daubechies, 1992; Rioul and Duhamel, 1992; Chui, 1992; Chui, 1992a; Vetterli and Herley, 1992; Rioul and Duhamel, 1992; Strang, 1989; and Strang, 1992].

Wavelet representations are conceptually similar to the Fourier representation. However, the wavelet transforms use different basis functions which are localized in time rather than global [Daubechies, 1992]. The general form for a wavelet representation of a signal is as follows [Chui, 1992; Daubechies, 1990; Daubechies, 1992; Mallat, 1989; Mallat, 1989a]:

$$f(x) = \sum_k a_k^j \phi(2^j x - k) + \sum_k b_k^j \psi(2^j x - k) \quad (2.3).$$

The x^{th} element of the sensor signal is represented by the $f(x)$ term; the a and b terms are the decomposition coefficients. The change in notation from representing the signal as $x(j)$ rather than $f(x)$ is due to the use of the j index in Equation 2.3. The basis functions in Equation 2.3 are ϕ and ψ . These are special functions with ψ defined as the wavelet function and ϕ defined as the scaling function. The wavelet and the scaling function are complex, high order polynomials rather than simple cosine and sine functions as used in the Fourier decomposition.

The special form of Equation 2.3 results in a number of useful characteristics. The 2^j term in both basis functions imparts a dilation property which causes the wavelet and scaling functions to constrict (compress) at higher frequencies and dilate (expand) at lower frequencies. The k term in Equation 2.3 imparts time translation. The translation and the dilation characteristics mean that the basis functions are time-frequency dependent localized functions. The combination of these two characteristics imparts the ability to perform both time and frequency analyses using a discrete wavelet decomposition. The following figure demonstrates how the dilation and translation terms work.

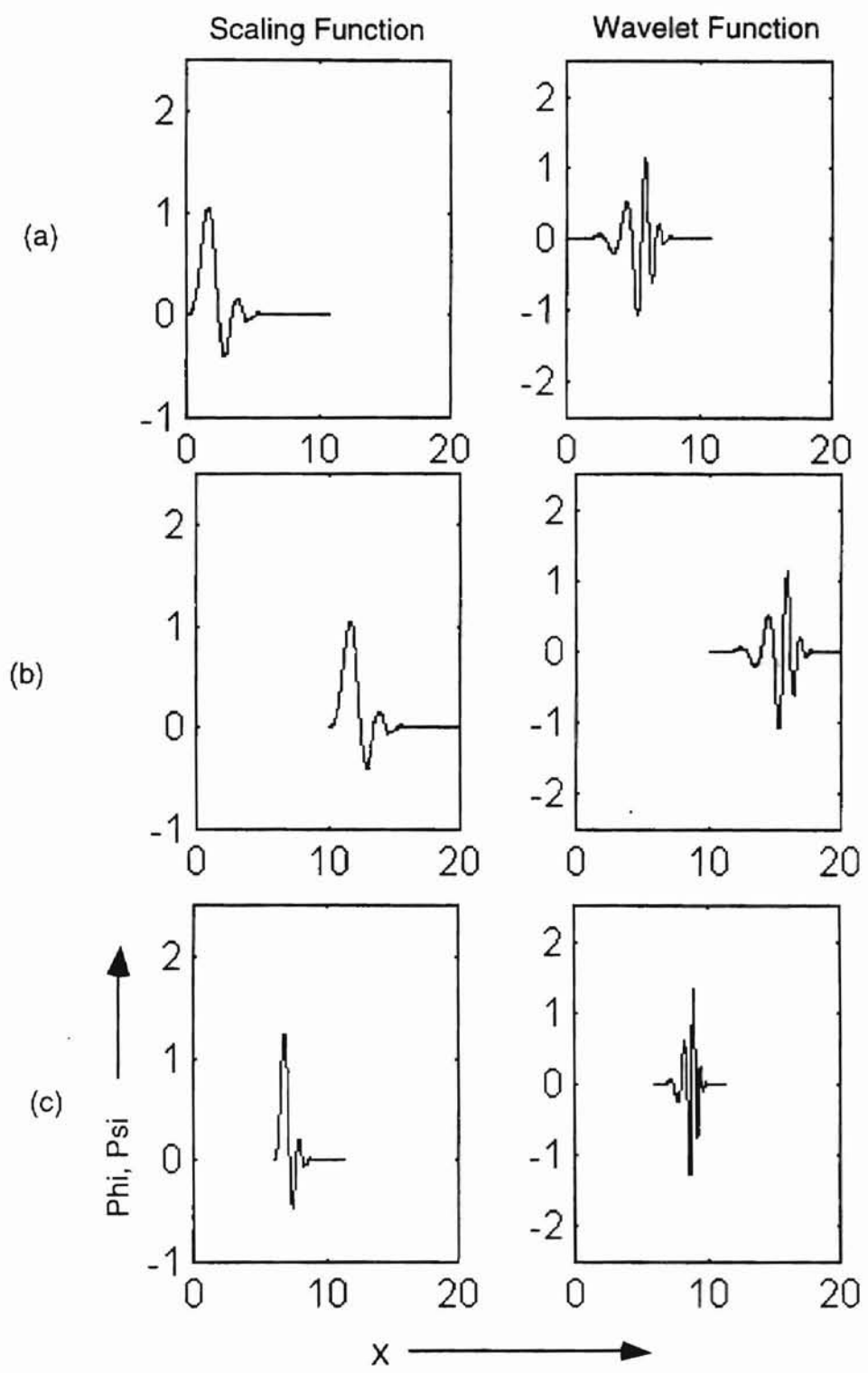


Figure 2-2: (a) Original wavelet function, (b) translated wavelet function, and (c) dilated and translated wavelet function.

The wavelet and scaling basis functions are also special and follow the form:

$$\phi(x) = \sum_k c_k \phi(2x - k) \quad (2.4a).$$

$$\psi(x) = \sum_k d_k \psi(2x - k) \quad (2.4b).$$

The $\phi(x)$ and $\psi(x)$ terms are the scaling function and wavelet function respectively. The c_k and d_k terms are the scaling function coefficients and wavelet function coefficients, respectively. Depending on the constraints imposed on $\phi(x)$ and $\psi(x)$ when these functions are synthesized, the scaling function and wavelet coefficients will change values. Typical constraints include orthogonality and orthonormality of the wavelet and scaling functions. As before, the k term in Equations 2.4a and 2.4b imparts the translation of the wavelet and scaling functions.

As just mentioned, different wavelet and scaling functions can be generated depending on the conditions that the functions are required to meet. This leads to the development of different wavelet “families.” These families are then divided into orders based on the complexity of the wavelet and scaling functions. The order of the wavelet family determines the length of the convolution filters used to implement the wavelet transforms. The work reported in this thesis has focused on the use of the Daubechies family of wavelets [Daubechies, 1990 and Daubechies, 1992]. The results, which are presented in later chapters, are applicable to any family of wavelets however as the fundamental concepts employed are independent of wavelet family and order. The

following illustrations show the wavelet and scaling function for the Daubechies family for different orders.

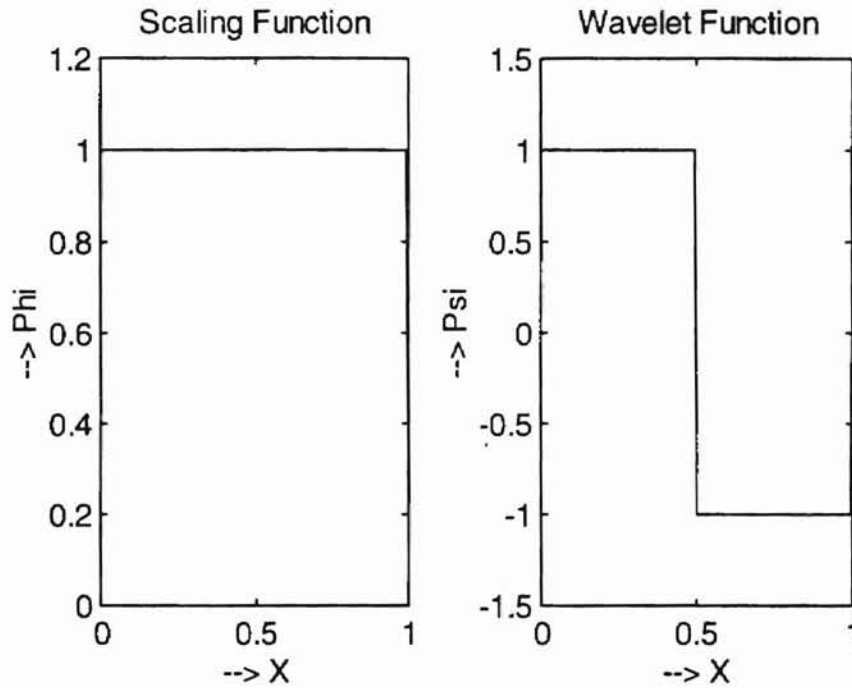


Figure 2-3: Daubechies family wavelet and scaling function with order = 1.

These are the least complex wavelet and scaling functions in the Daubechies family; they represent the lowest order. The next figure shows a more complex wavelet and scaling function.

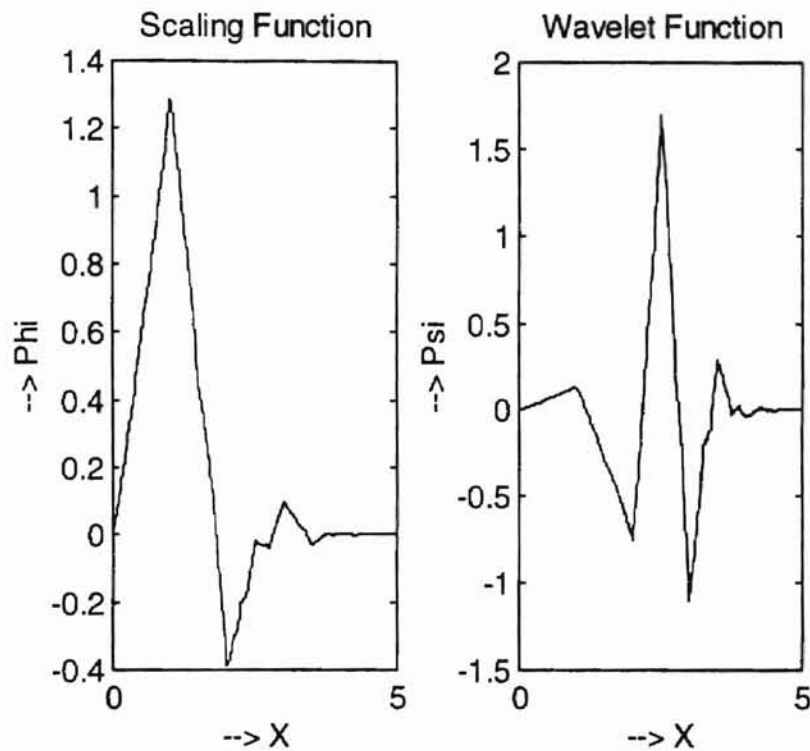


Figure 2-4: Daubechies family wavelet and scaling function with order = 3.

The complexity of the functions increases as the order increases. The higher order increases the order of the polynomial used to generate the wavelet and scaling functions.

For the work in this thesis, the Daubechies family with an order of six was used. The corresponding wavelet and scaling functions are shown in the following figure.

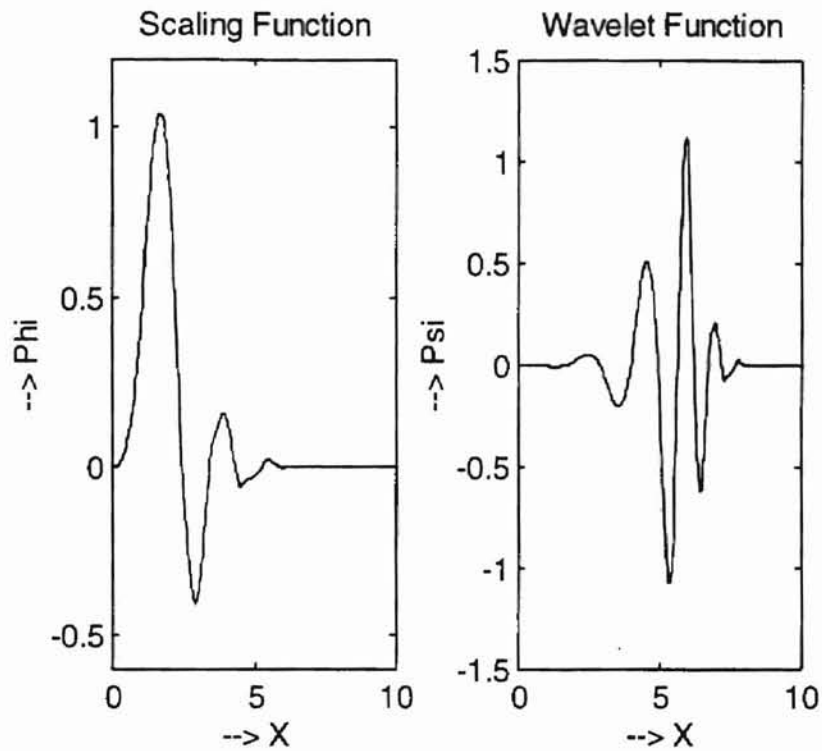


Figure 2-5: Daubechies family wavelet and scaling function with order = 6.

2.4 Wavelet Decomposition

As described in the appropriate literature, there are a number of techniques which can be used to generate a wavelet decomposition of a discrete signal. The most elegant and easily understood technique employs matrix algebra [Strang, 1989 and Strang, 1992]. Unfortunately, the matrix method is industrially impractical due to large, sparse matrices which are computationally difficult to handle. The most common method to perform

wavelet decomposition employs a convolution filtering approach. This is the technique used for this research project.

In order to use this method, the discrete sensor signal must be a power of two. That is, the signal must have 2^N points where N is an integer value. The discrete process sensor signal is then converted into a series of decomposition coefficients (a_k^j and b_k^j) generated from the scaling (c_k^j) and the wavelet (d_k^j) coefficients as per Equations 2.3, 2.4a, and 2.4b.

To illustrate, consider a signal represented by the vector $a_n^0 = [a_1^0 \ a_2^0 \ a_3^0 \ a_4^0 \ a_5^0 \ a_6^0 \ a_7^0 \ a_8^0]$. The superscript denotes the number of levels of decomposition. Hence, each element of the original signal has a superscript of zero. For n points in the original signal, there are $n/2$ coefficients of $a_{n/2}^1$ and $b_{n/2}^1$ generated at the first level of decomposition. These decomposition coefficients are generated by the following formula [Chui, 1992 and Daubechies, 1992]:

$$a_j^1 = \frac{1}{2} \sum_k a_k^0 c_{2j-k} \quad j = 1, \dots, \frac{n}{2}; \quad k = 1, \dots, n \quad (2.5a).$$

$$b_j^1 = \frac{1}{2} \sum_k a_k^0 d_{2j-k} \quad j = 1, \dots, \frac{n}{2}; \quad k = 1, \dots, n \quad (2.5b).$$

The index j denotes the level of decomposition. The a and b coefficients are the decomposition coefficients with the superscript representing the level of decomposition.

The c and d terms are the scaling function and wavelet coefficients, respectively, as defined previously. The decomposition coefficients are calculated from Equations 2.5a and 2.5b and depend on the wavelet family and order.

After one level of decomposition the original eight element example signal from the previous page is represented by the following coefficients $[a_1^1 a_2^1 a_3^1 a_4^1]$ and $[b_1^1 b_2^1 b_3^1 b_4^1]$. The a^1 coefficients are used as the input for the next level of decomposition to generate the a^2 and b^2 coefficients. This results in the following coefficients $[a_1^2 a_2^2]$ and $[b_1^2 b_2^2]$ at the second level of decomposition. The maximum number of levels of decomposition depends on the size of the original signal. With 2^N points, then the signal can be decomposed N levels. For this case, the signal contains eight values so the signal can be decomposed three levels. At the third and final level of decomposition, the coefficients would be $[a_1^3]$ and $[b_1^3]$. At each level of decomposition, there are half as many a 's and b 's as the previous level (except for the zero level of decomposition where there are no b 's).

The original signal can be reconstructed from the decomposition coefficients as follows [Chui, 1992 and Daubechies, 1992]:

$$a_k^0 = \sum_k a_j^1 c_{2j-k} + \sum_k b_j^1 d_{2j-k} \quad k = 1, \dots, n. \quad (2.6).$$

When the original signal (a_n^0) is reconstructed with all the decomposition coefficients, there is perfect reconstruction. The reconstructed signal is an exact duplicate of the original signal.

2.5 Trend Extraction

Wavelet decomposition provides an appealing technique to extract the fundamental trend of a sensor signal. The original signal as denoted by a_n^0 is decomposed into the $a_{n/2}^1$ and $b_{n/2}^1$ coefficients. These coefficients break the signal up into the smoothed part of the signal and the high frequency component of the signal. The a^1 coefficients are the blurred coefficients; they represent the smoothed signal. The b^1 coefficients are the detail coefficients. These coefficients represent the high frequency component of the signal at the frequency associated with the level of decomposition. At each level of decomposition only the blurred coefficients are used to decompose the signal. The following figure illustrates how the decomposition process works.

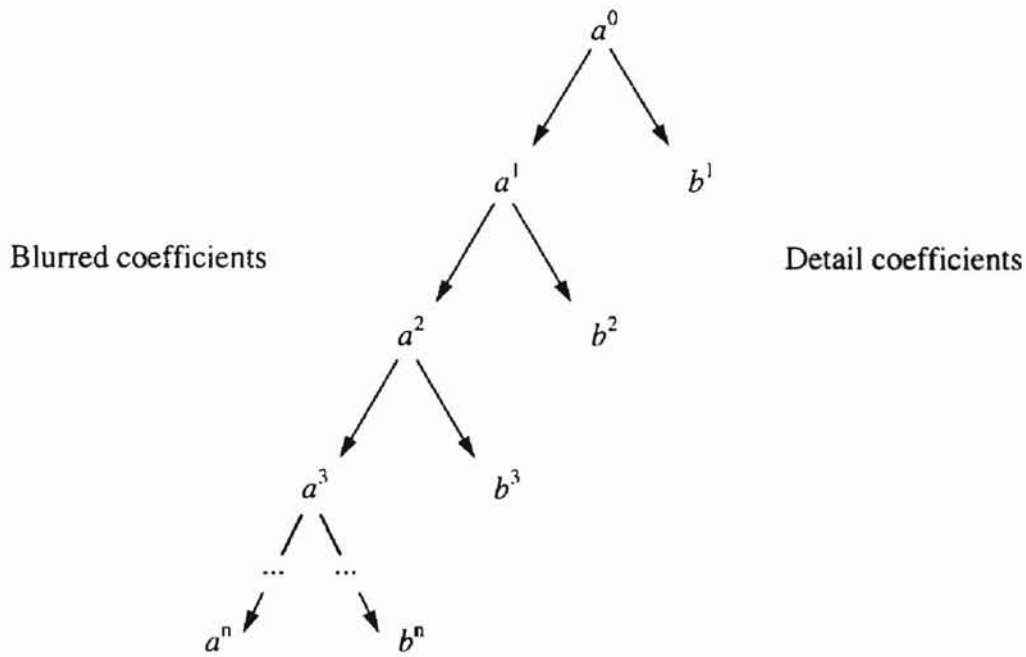


Figure 2-6: Decomposition pyramid for wavelet representation.

At the first level of decomposition, the original signal (a_n^0) is separated into two composite signals; the first composite signal captures the smoothed part of the original signal. It is represented by the blurred coefficients ($a_{n/2}^1$). The other composite signal contains the high frequency component of the original signal. It is represented by the detail coefficients ($b_{n/2}^1$). At the next level of decomposition, the blurred signal ($a_{n/2}^1$) is then further decomposed into the blurred signal and the detail signal. This process is repeated at each level of decomposition; the smoothed signal continues to be broken into a smoothed component and a high frequency component. The following figure illustrates the blurred coefficients as generated at each level of decomposition.

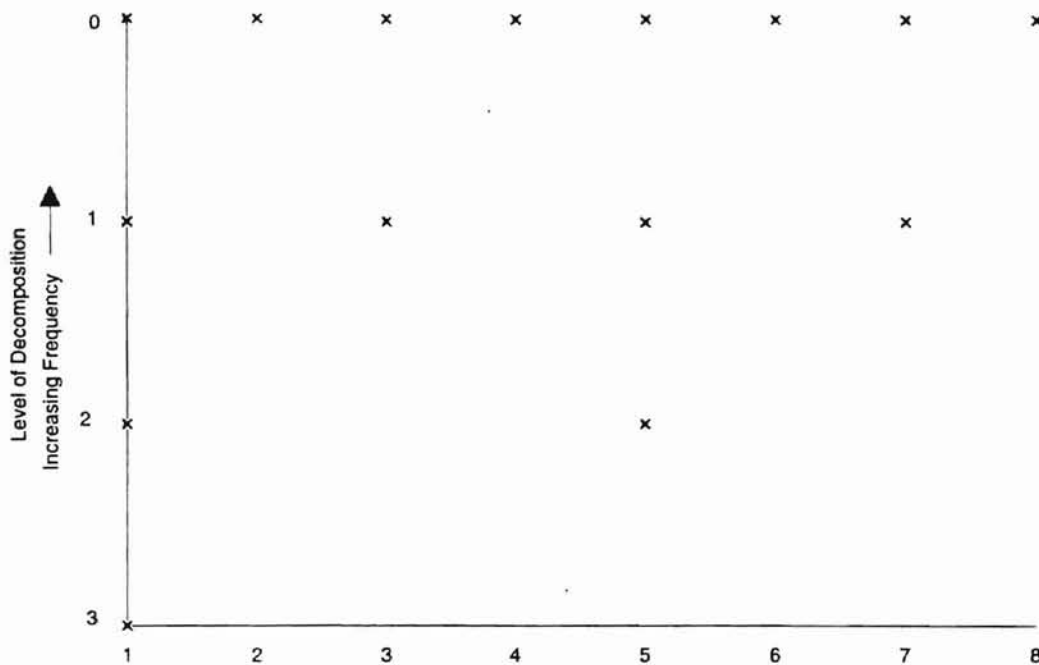


Figure 2-7: Decreasing number of blurred coefficients used to represent a signal at different levels of decomposition.

From Figure 2-7, the number of blurred coefficients decreases by a factor of two at each level of decomposition. For example, there is a signal with eight elements. At the first level of decomposition, there are only four blurred coefficients used to represent the original signal. At the second level of decomposition, there are only two blurred coefficients to represent the smoothed composite of the original signal. At each level, the blurred coefficients cover a larger time scale. The basis functions dilate (expand) with increasing levels of decomposition to capture the original signal over increasingly larger time scales. The blurred coefficients are used to generate an increasingly smoothed approximation of the original signal as the level of decomposition increases.

Trend extraction uses only the blurred coefficients to reconstruct the smoothed signal. At each level of decomposition, the original signal is smoothed further. The blurred coefficients capture the smoothed part of the signal while the detail coefficients capture the high frequency component of the signal. At an appropriate decomposition level, the smoothed signal is reconstructed using only the blurred coefficients for that level. The detail coefficients are ignored. Since, the detail coefficients are not used in the reconstruction of the smoothed signal; the high frequency component of the original signal has been eliminated. This reconstructed signal represents the fundamental sensor trend with no high frequency noise or measurement error. The more levels of decomposition, the more smoothed the reconstructed signal is.

2.6 Implementation of Trend Extraction

As previously stated, this research uses the convolution filtering approach to generate the wavelet decomposition. The blurred and detail coefficients are generated with two different convolution filters. These are special filters generated from the wavelet family and wavelet order [Mallat, 1989; Mallat, 1989a; Mallat, 1989b; Vetterli and Herley, 1992; Rioul and Duhamel, 1992]. The first filter is a lowpass filter designated as H. The second filter is a highpass filter designated by G. These filters are convolved with the original signal (a_n^0) and downsampled to generate the blurred and detail coefficients. The following formula is used:

$$a_{n/2}^1 = \vee(H * a_n^0) \quad (2.7a).$$

$$b_{n/2}^1 = \vee(G * a_n^0) \quad (2.7b).$$

The “ \vee ” represents the downsampling operation where every other term from the convolution operation is kept. The convolution operation is denoted by “ $*$ ” and will be explained in more detail shortly. At each level of decomposition, new blurred and detail coefficients are generated from Equations 2.7a and 2.7b. The signal can be reconstructed perfectly by the reverse process as given in the following formula:

$$a_n^0 = \tilde{H} * (\wedge a_{n/2}^1) + \tilde{G} * (\wedge b_{n/2}^1) \quad (2.8).$$

The “ \wedge ” represents the upsampling operation where a zero is inserted between each term of the a^1 and b^1 coefficients before convolving (“ $*$ ” operation) with the appropriate filter.

For trend extraction only the blurred coefficients are used to reconstruct the smoothed signal. Hence, the following equation is used for trend extraction application.

$$a_n^0 = \tilde{H} * (\wedge a_{n/2}^1) \quad (2.9).$$

When using the convolution operation, each term in the resulting vector is generated from the sum of the product of specific terms from the two convolved vectors. This means that the first and last terms in the resulting vector are generated from only a

few terms from the convolved vectors. In general, the discrete linear convolution operation is defined as [Jong, 1982 and Gray and Goodman, 1995]:

$$y(k) = \sum_{n=0}^k x(n)h(k-n) = \sum_{n=0}^k x(k-n)h(n) \quad (2.10).$$

The expanded form of the convolution results in the following values for $y(k)$ as computed for an n -dimensional vector \bar{x} and m -dimensional vector \bar{h} [MathWorks, 1992; Gray and Goodman, 1995]:

$$\begin{aligned} y(1) &= x(1) \cdot h(1) \\ y(2) &= x(1) \cdot h(2) + x(2) \cdot h(1) \\ y(3) &= x(1) \cdot h(3) + x(2) \cdot h(2) + x(3) \cdot h(1) \\ &\dots \\ y(k-1) &= x(n) \cdot h(m-1) + x(n-1) \cdot h(m) \\ y(k) &= x(n) \cdot h(m) \end{aligned} \quad (2.11).$$

The convolution product has dimension length of $n + m - 1$. The number of terms used to calculate $y(k)$ can be illustrated in the following figure with a 4-dimensional vector \bar{x} and 5-dimensional vector \bar{h}

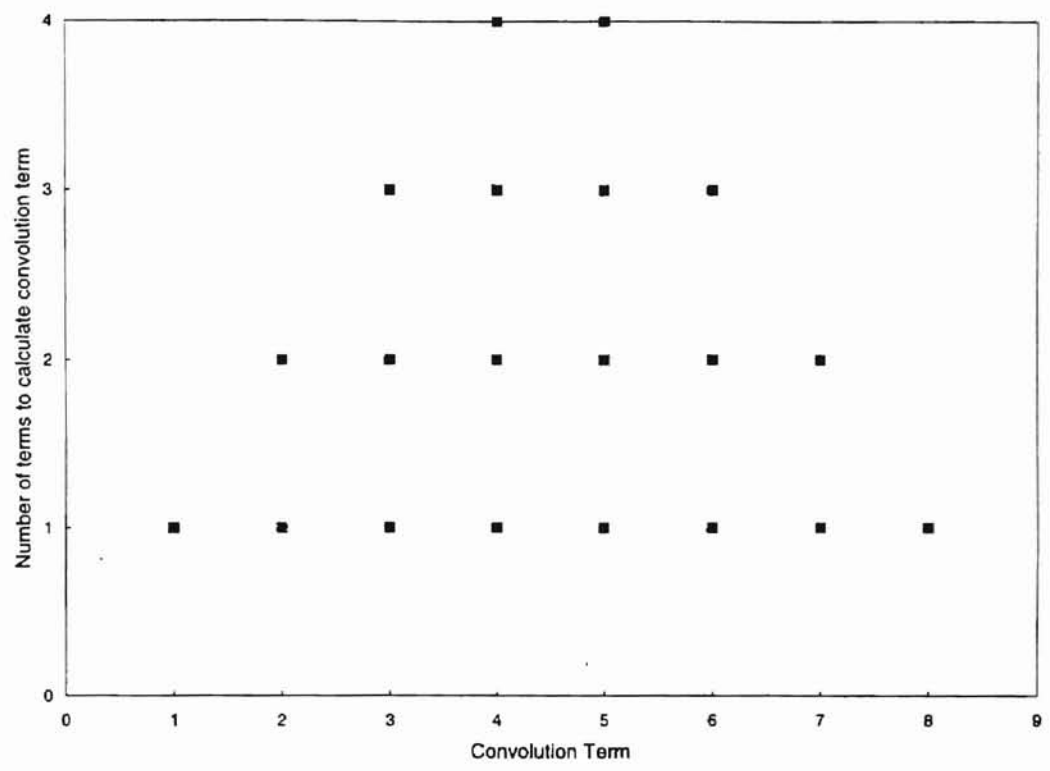


Figure 2-8: Number of terms used to calculate the convolution term at each index.

The expanded form shows that the terms for $y(k)$ at each end of the series are calculated from only a few terms from $x(n)$ and $h(m)$. Because only a few elements of \bar{x} and \bar{h} are used in the calculation, these end terms contain a distorted representation. The middle $y(k)$ terms contain enough of $x(n)$ and $h(m)$ terms so there is no distortion. For the case where $m > n$, then the first and last $n-1$ elements of $y(k)$ exhibit distortion due to the convolution operation. For the case $n > m$, then the first and last $m-1$ elements of $y(k)$ exhibit distortion. For the example in Figure 2-8, the $y(4)$ through $y(5)$ terms would have no end-distortion as caused by the convolution operation. If $y(k)$ is needed with no distortion at each end of the signal, then the number of $x(n)$ terms or $h(m)$ terms is increased so that the resulting k -elements of \bar{y} have no distortion. For the case where

$m > n$, then add $2n - 2$ elements to $h(m)$ so that k -elements of \bar{y} have no distortion. For the case $n > m$, then add $2m - 2$ elements to $x(n)$ so that k -elements of \bar{y} have no distortion. This leads to the need for appropriate extension techniques when wavelet smoothing is employed via the convolution method for trend extraction. The original signal must be extended on each end to compensate for the end-distortion which is guaranteed to occur.

2.7 Chapter Summary

Wavelet representation and Fourier representation both provide techniques to accomplish trend extraction. Trend extraction is accomplished by removing the high frequency content of a sensor signal. The wavelet decomposition is preferred for trend extraction because this technique uses localized basis functions rather than global basis functions used in Fourier representation. The nature of the wavelet decomposition is such that the smoothed approximations that can be generated via this method are much better than those which can be generated using only the low frequency components of a Fourier decomposition.

The wavelet representation uses the convolution operation to perform the wavelet decomposition and reconstruction. The convolution approach is preferred over the matrix method since the matrix technique is computationally intensive and impractical on an industrial scale. The convolution operation used for the wavelet decomposition causes end-distortion in the extracted trend. End-distortion at the real-time end of a signal is

unacceptable since the smoothed approximation is not representative of the true trend.
Hence, a method to avoid end-distortion and preserve the true trend of a signal is essential.

Chapter Three: Proposed Extension Techniques

3.1 Introduction

Wavelet smoothing is used to provide accurate real-time trend extraction in the pattern-based monitoring methods being developed by Dr. Whiteley's research group at Oklahoma State University. However, as described in the previous chapter, the implementation of the wavelet smoothing causes end-distortion of the extracted trend. To avoid this problem, the windowed sensor signal is extended on each side to provide a composite extended signal. This composite extended signal is then smoothed. The extended parts of the signal on each end are then removed to leave the smoothed trend of the original signal without any end-distortion. The resulting smoothed signal provides a reasonable approximation of the fundamental trend of the signal.

When this approach is applied for real-time trend extraction, a problem arises on the real-time end of the sensor signal. The "old" end of the signal can be extended with existing sensor data. However, the required extension data on the real-time end corresponds to a future prediction of plant performance. This data must be synthesized in a manner which does not adversely affect the smoothed trend which is extracted.

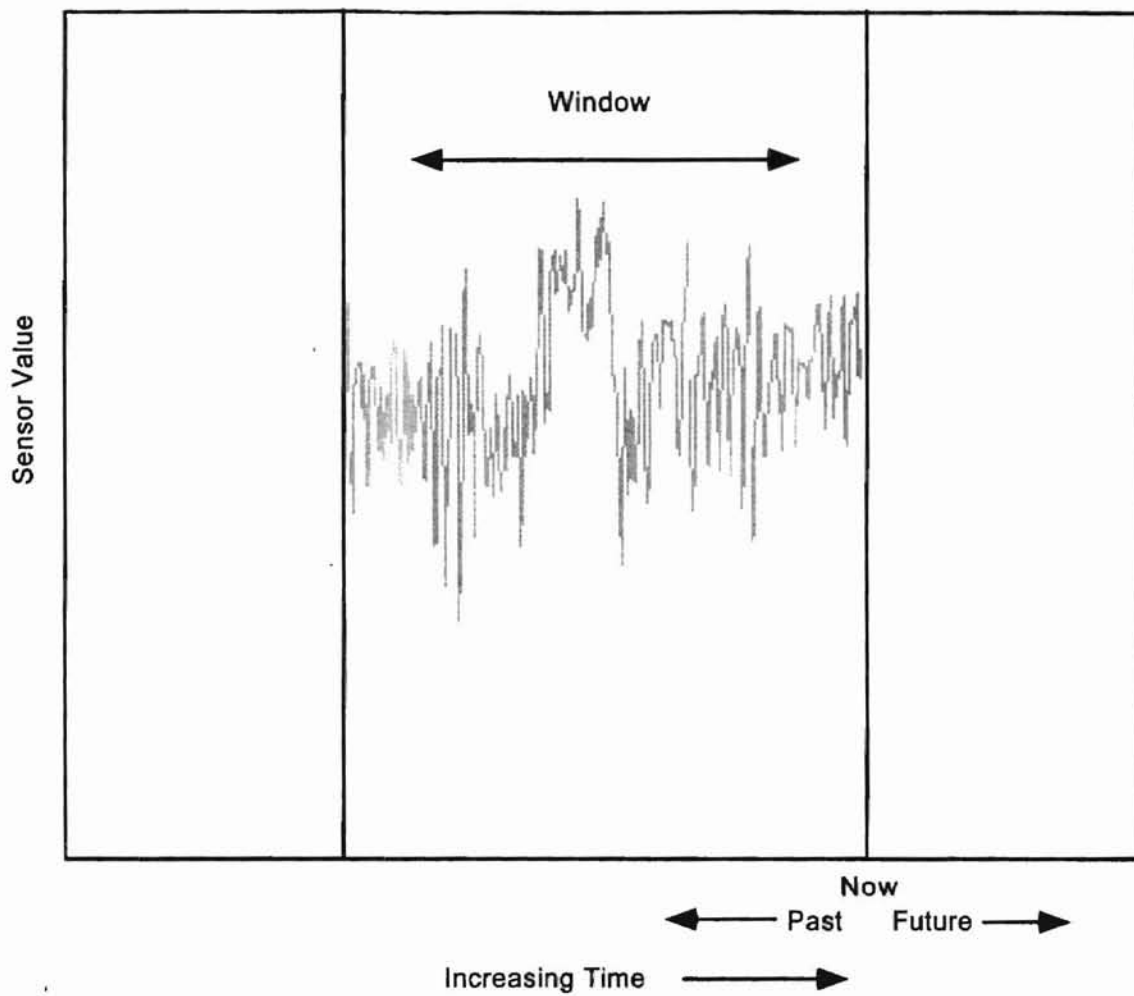


Figure 3-1: Original sensor signal to be smoothed.

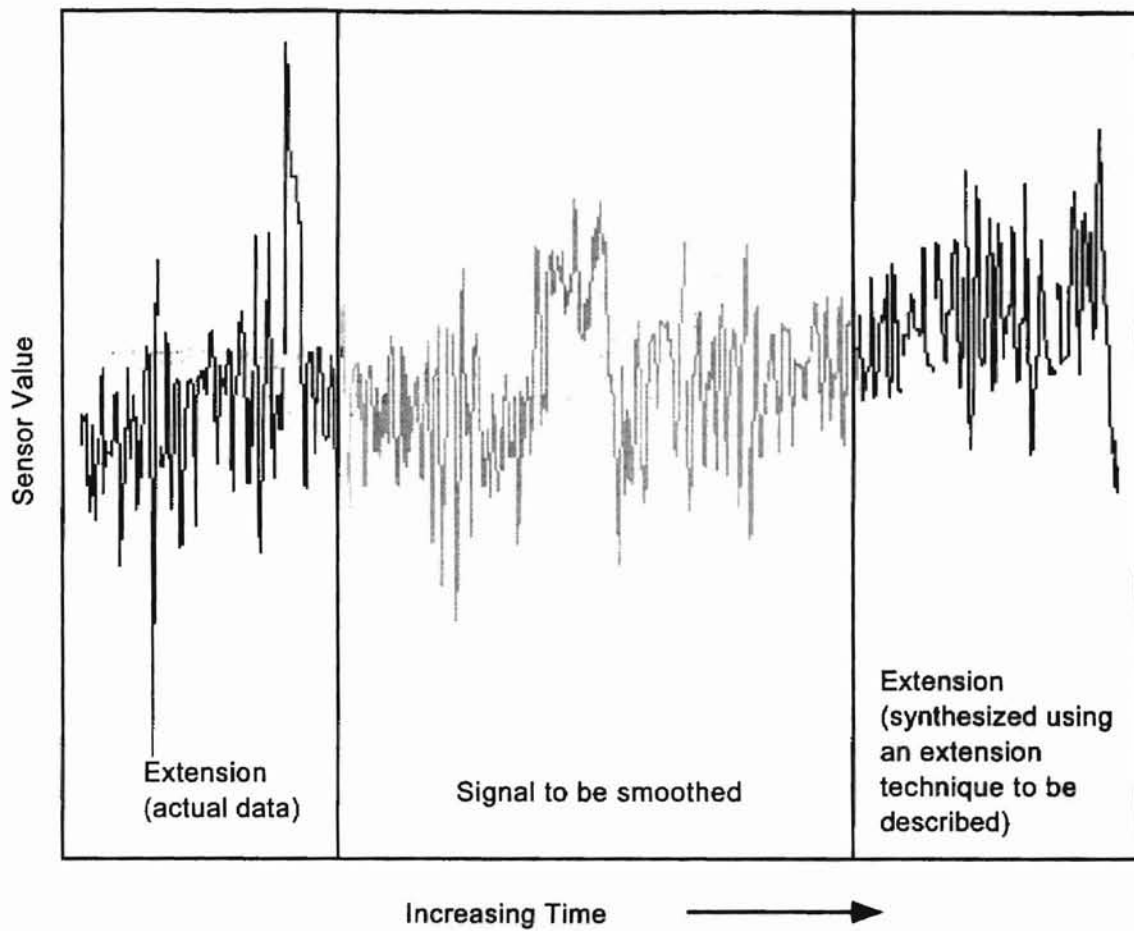


Figure 3-2: Extended signal prior to wavelet smoothing.

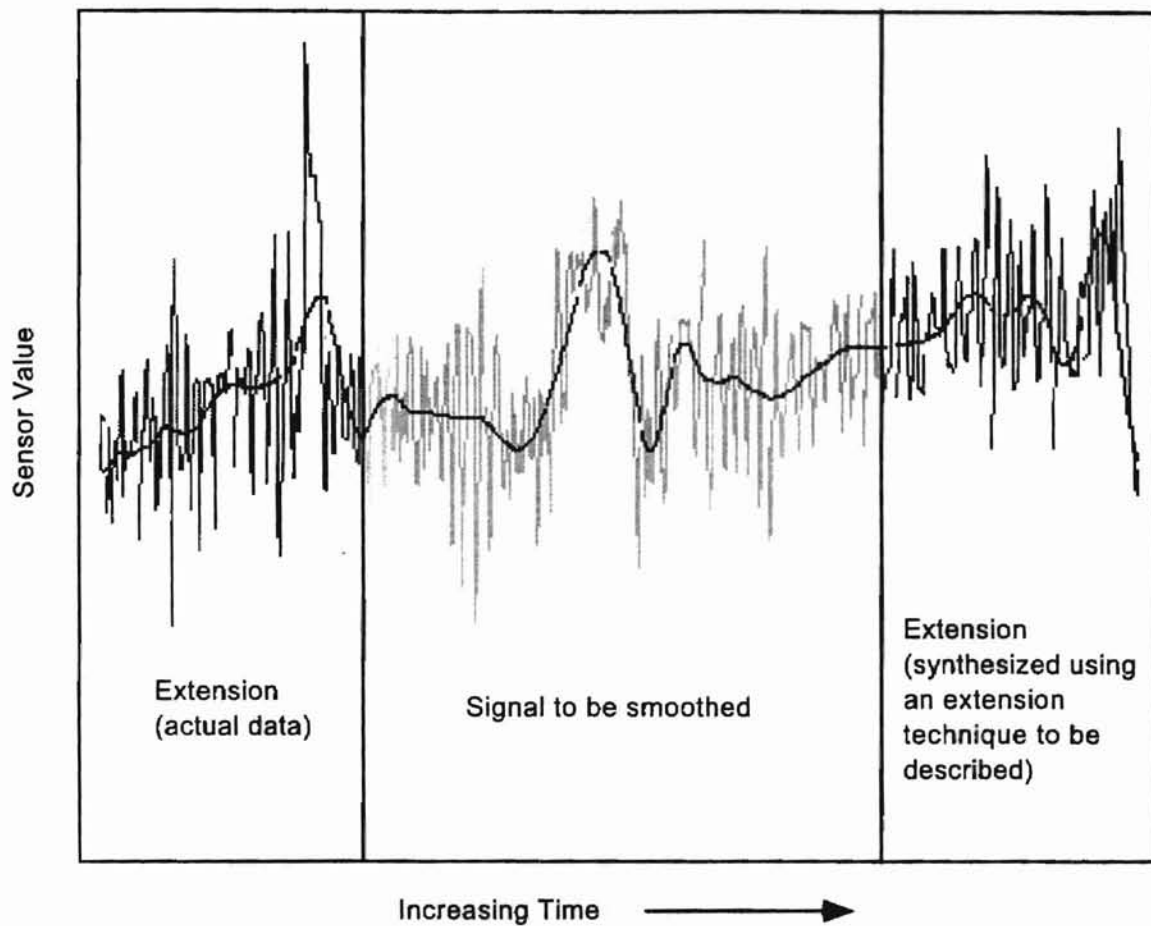


Figure 3-3: Extended signal after wavelet smoothing.

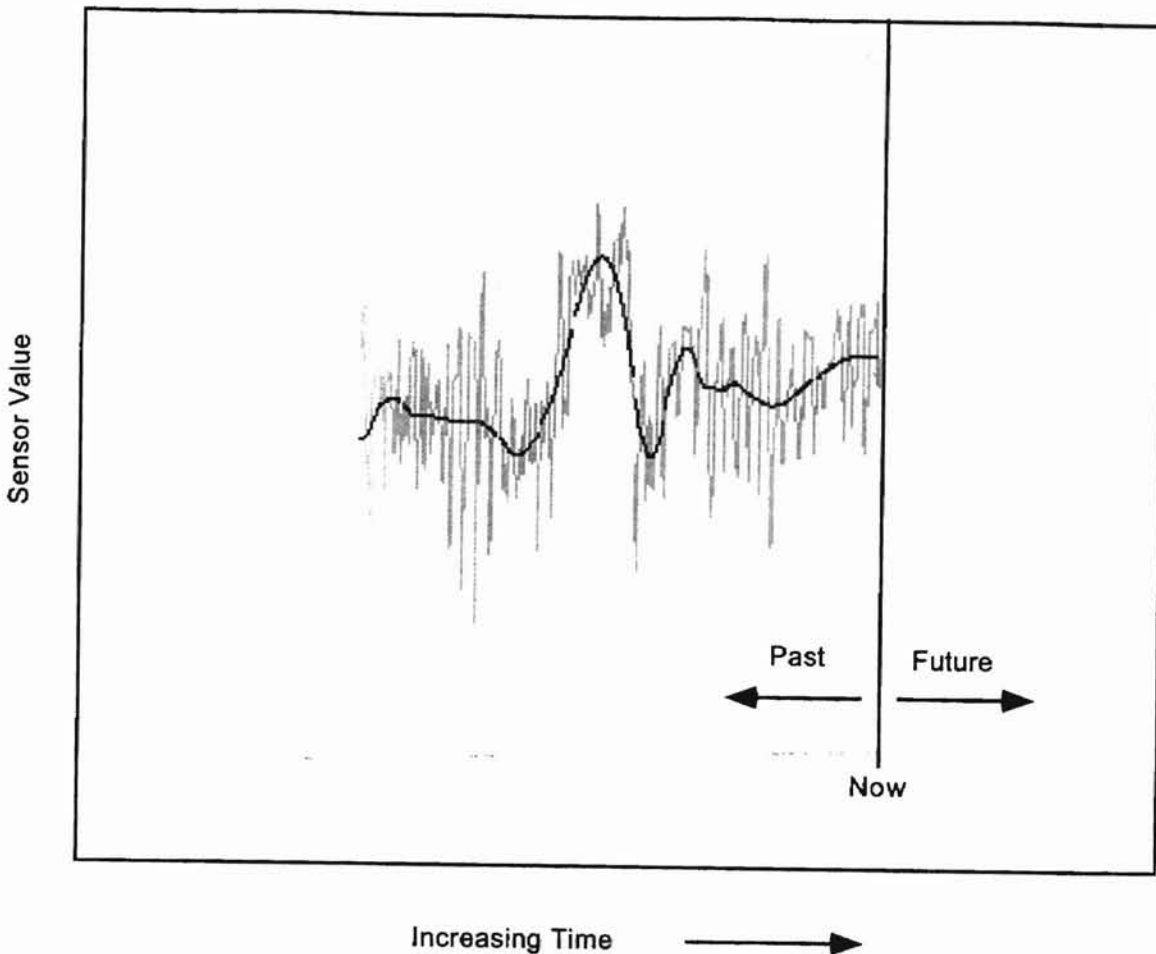


Figure 3-4: The original sensor signal with the extracted trend.

This real-time extension problem requires development of extension techniques specifically designed for wavelet smoothing. As a first effort, Mr. V. Raghavan developed the Raghavan NET2 technique to provide a method for real-time extension [Raghavan, 1995]. The Raghavan NET2 technique uses the most current sensor data to extend the real-time end of the sensor signal. However, the Raghavan NET2 technique was developed for a specific application and requires extensive trial-and-effort to adapt for other applications. The main contribution of this thesis is the generalization of the

Raghavan NET2 technique. Specifically, the work reported in this thesis employs Raghavan's basic ideas in a methodology which adapts to the characteristics of the signal being smoothed. There are five new proposed extension techniques presented in this chapter. Each of the five new proposed extension techniques are described in detail. Before doing so, the Raghavan NET2 technique is reviewed.

3.2 Raghavan NET2 technique

Raghavan examined several possible extension techniques in his Master of Science thesis [Raghavan, 1995]. He examined several of the extension techniques, including the following techniques: extending the signal with zeroes, extending the signal with constant values, extending the signal as a periodic extension, extending the signal as a mirror image of the current signal [Raghavan, 1995]. He concluded that none of the traditional extension techniques are adequate for real-time trend extraction. He then proceeded to propose a new extension technique called the Raghavan NET2 technique.

Raghavan's solution was an inverted symmetric extension approach [Raghavan, 1995]. The idea behind the Raghavan NET2 technique is to extend the signal by mimicking the most recent sensor signal values. If the sensor signal is increasing, then the extended signal needs to increase at the same rate. If the sensor signal is staying relatively constant, then the extended signal needs to stay relatively constant. The extended signal is the inverted signal of the mirror image of the signal. This results in an

extended signal that is generated using the most recent sensor signal data. The extension is attached to the windowed sensor signal at the tie-point. The following figure demonstrates how the inverted symmetric extension technique works.

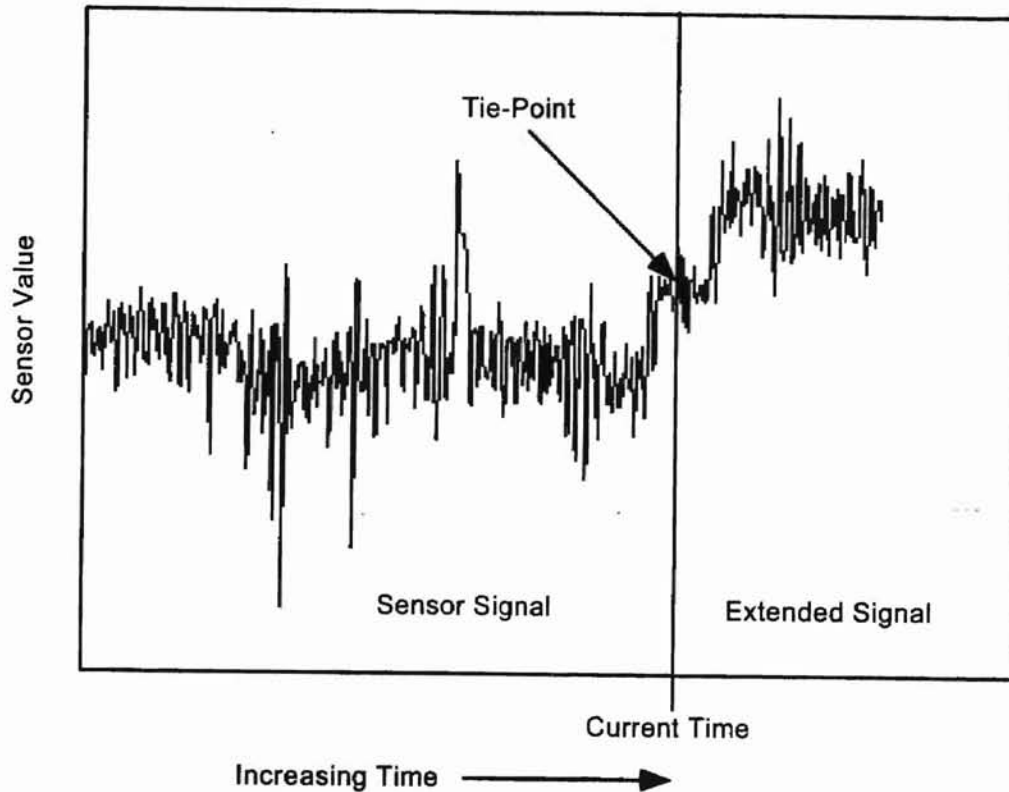


Figure 3-5: The inverted symmetric extension technique. The extended signal is generated from the windowed signal.

The tie-point between the extension and the windowed sensor signal has a dramatic effect on the shape of the composite signal. If the tie-point is high, then the extended signal is shifted up. If the tie-point is low, then the extended signal is shifted down. The following two figures illustrate the two cases:

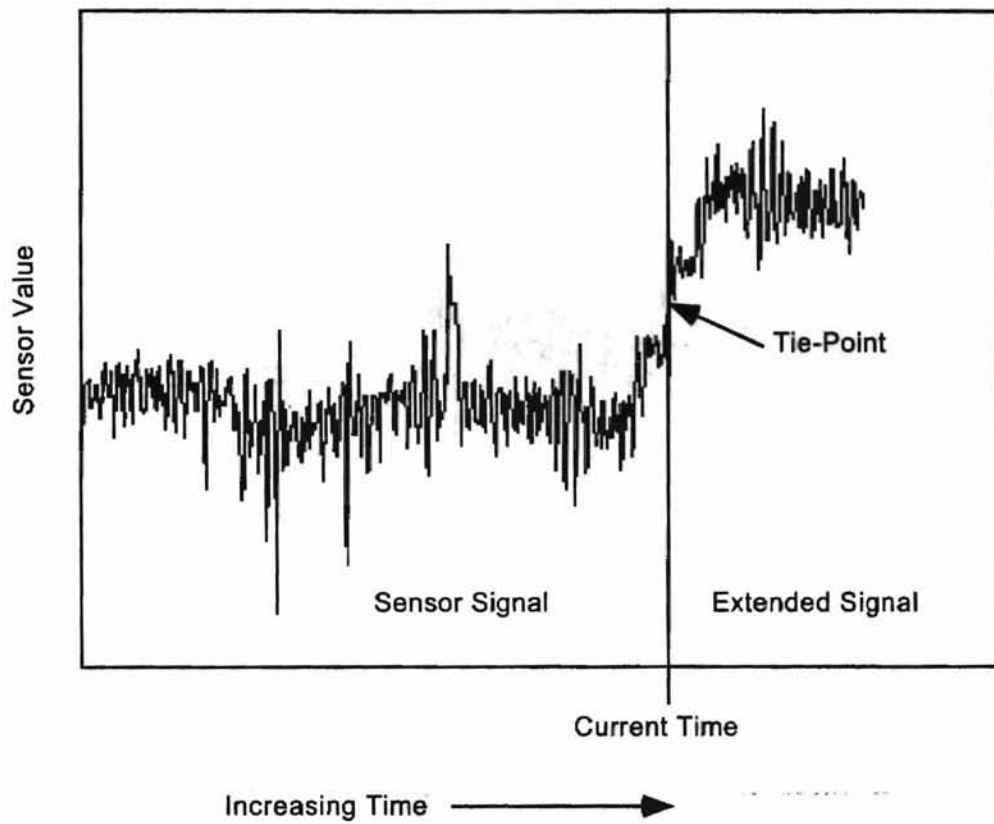


Figure 3-6: Inverted symmetric extension technique with high tie-point.

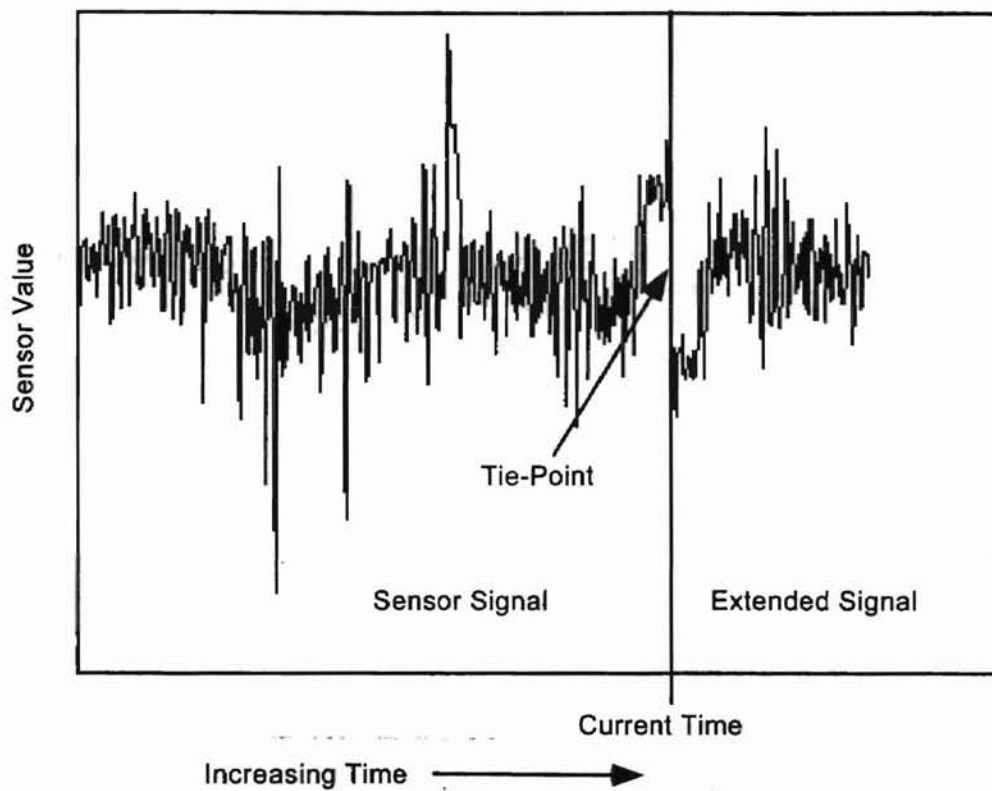


Figure 3-7: Inverted symmetric extension technique with a low tie-point.

The tie-point is the key parameter that must be specified in the Raghavan NET2 technique. If the tie-point is too high or too low, then the extracted trend which is generated does not adequately represent the sensor signal. The following figures use the previous examples to illustrate how the choice of the tie-point affects the smoothed trends that are ultimately extracted.

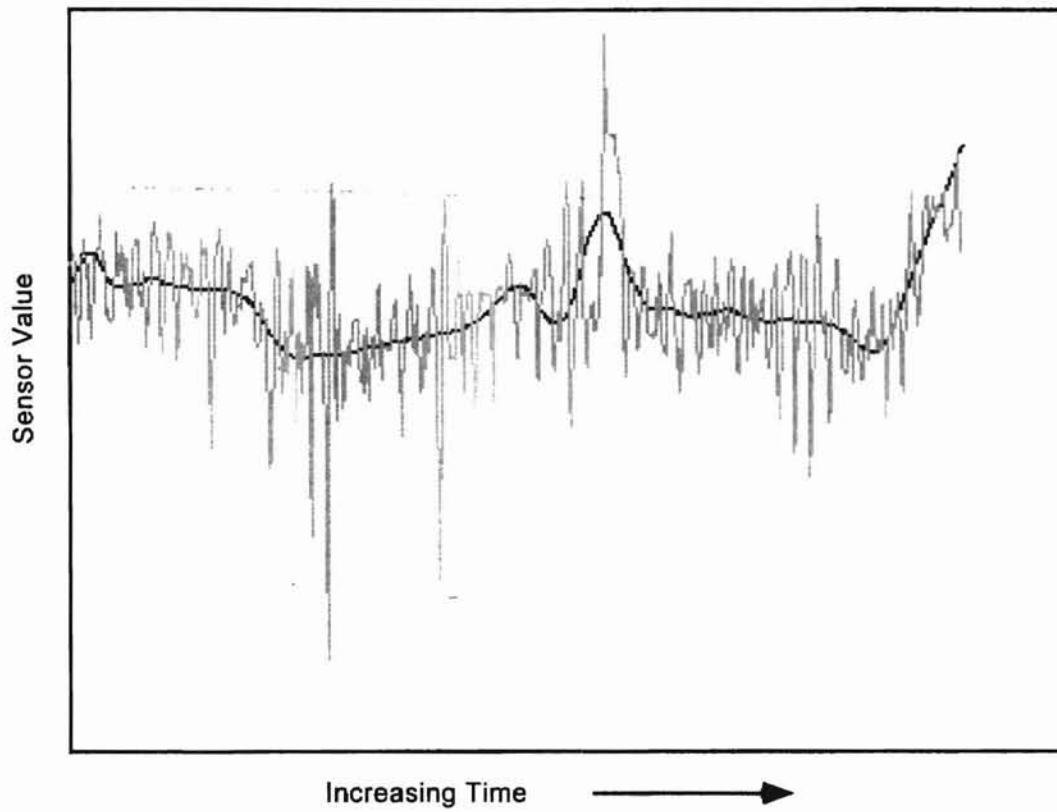


Figure 3-8: Smoothed signal using inverted symmetric extension technique with high tie-point.

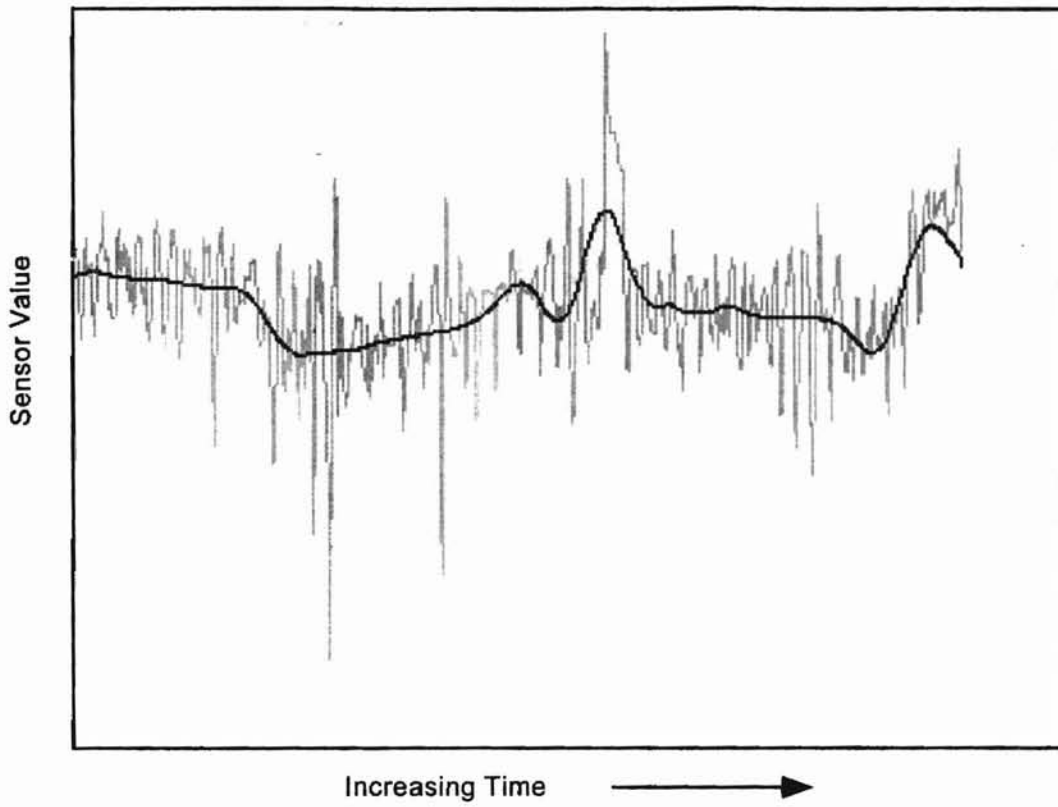


Figure 3-9: Smoothed signal using inverted symmetric extension technique with low tie-point.

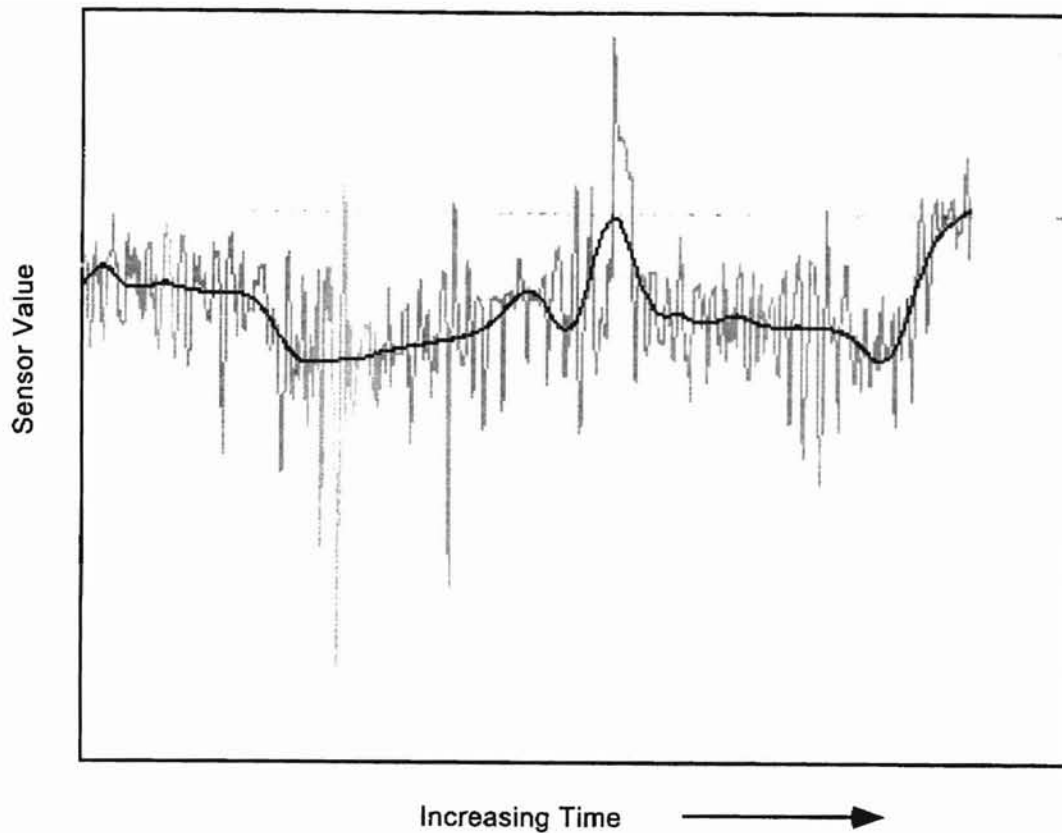


Figure 3-10: Smoothed signal using inverted symmetric extension technique with correct tie-point.

In the Raghavan NET2 technique, the tie-point is calculated as the average of the most recent sensor data. The maximum number of sensor signal values included in the tie-point calculation is called the threshold number, α . From the threshold number, a dead-zone number, β , is calculated. The dead-zone is the minimum number of sensor signal values to be included in the calculation of the tie-point. The Raghavan NET2 technique uses a threshold size of 50 sensor signal values. Hence, the dead-zone is calculated as 40 percent of the threshold size. So, the dead-zone is 20 sensor signal values. There are no guidelines as to establish this threshold number. The threshold

number and dead-zone values listed above are hard-coded in the Raghavan NET2 technique. These values were empirically determined for the specific application under investigation at the time of Raghavan's research.

The mean associated with each point between the dead-zone and the threshold is calculated according to the following equation:

$$\bar{x}_m = \frac{1}{m} \sum_{i=1}^m x_i, \quad \alpha \geq m \geq \beta \quad (3.1).$$

Note that x_0 denotes the sensor reading at the current instant in time. The mean is then used to calculate the mean squared deviation (M) as following:

$$M = \sum_{i=0}^m \frac{(x_i - \bar{x}_m)^2}{i}, \quad \alpha \geq m \geq \beta \quad (3.2).$$

A total of $(\alpha - \beta)$ M 's are calculated in this manner. The value of the mean \bar{x}_j for the smallest M_j is used as the tie point \bar{x}_{TP} .

The extended signal is calculated with the tie-point and the most recent process sensor signal. The extended signal is calculated from the following formula:

$$x(-k) = [x(k) - \bar{x}_{TP}] + x(k), \quad k = 1, 2, \dots, \frac{n}{3} \quad (3.3).$$

The extension for the real-time end of the signal $[x(-1), x(-2), \dots, x(-\frac{n}{3})]$ is calculated from the current process sensor signal, $x(-k)$, and the tie-point, \bar{x}_{TP} . The signal is indexed with the tie-point having an index of $k = 0$. The k index is negative for future sampling instances consistent with the traditional process control convention. The length of the extension is $n/3$ where n is the number of points in the windowed signal. Wavelet smoothing is then applied to the fully extended signal. After smoothing, the extended part of the smoothed signal is removed to leave the original sensor signal without end-distortion.

The Raghavan NET2 technique offers no guidelines for calculating either the threshold size or the dead-zone size. The size of threshold and dead-zone are determined on a purely empirical basis. This situation demanded more research to develop an improved extension technique to provide accurate trend extraction from different sensor signals.

3.3 Dead-Zone

The dead-zone requires that a certain number of sensor points be included in the calculation of the tie-point. Based on the current author's research, the dead-zone needs to be based on the period of the dominant frequency in the signal which is being smoothed. The following figure is representative of the trends produced by most plant sensors. A periodic component is clearly visible in the sensor data.

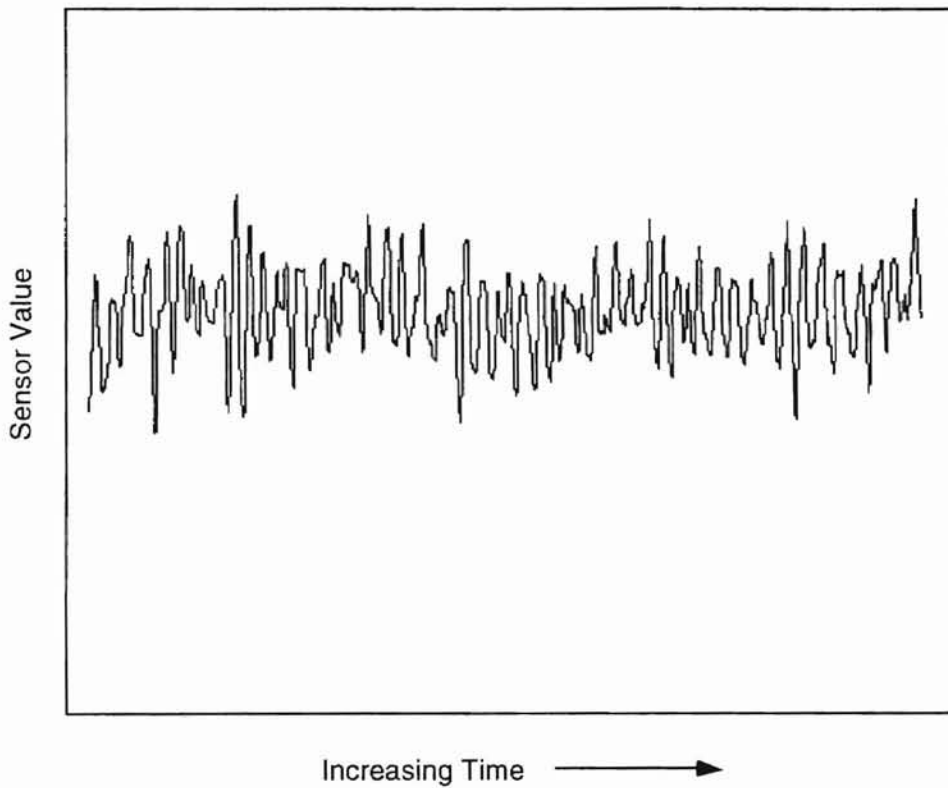


Figure 3-11: Sensor signal showing the periodic frequency of the sensor noise.

The periodic changes in the sensor are due to sensor noise and interactions between the process and the control system. This creates a dominant frequency that can be calculated using a power spectral analysis with Fourier transforms.

The following method is used to determine the dominant frequency in the signal. First, a range of the sensor signal is selected. A Fourier transformation is performed on this sensor signal to obtain the Fourier coefficients. Then the power spectrum is plotted [Press *et al.*, 1992]. The power spectrum measures the relative intensity (or magnitude) of each frequency. The more intense frequencies correspond to the frequencies which have a larger amplitude in the Fourier transform. The power spectrum for the previous sensor signal is represented by the following figure.

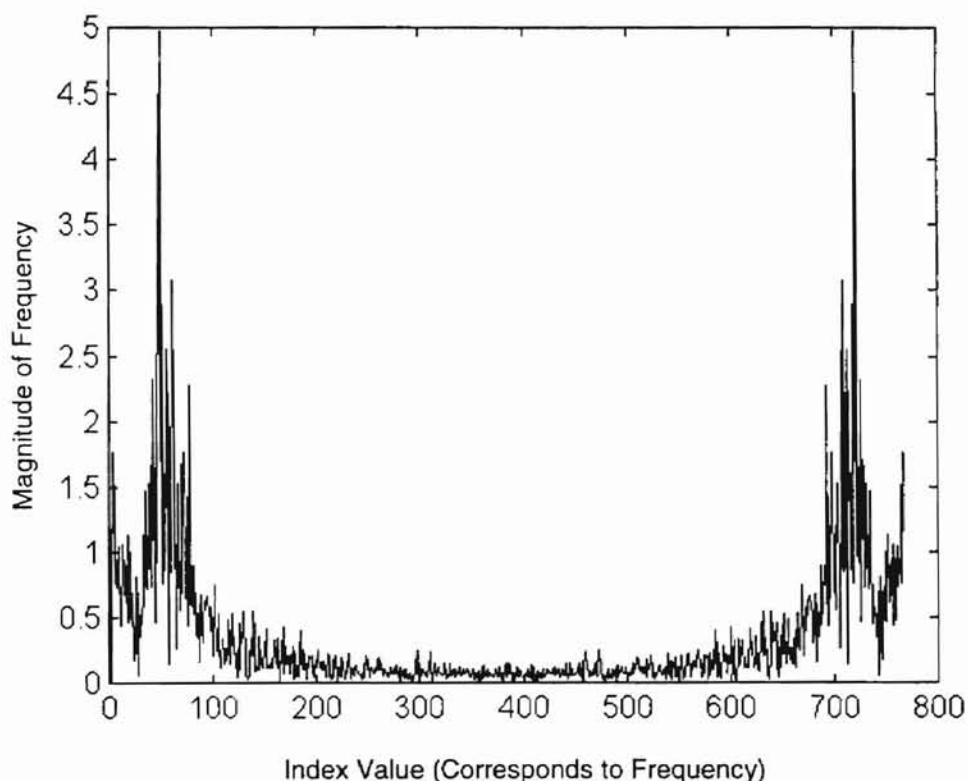


Figure 3-12: Power spectrum of sensor signal.

This power spectrum shows that there is a large spike around the index value of 50. This index can be converted to the corresponding frequency with the following equation [Press *et al.*, 1992]:

$$f = f_s (k / N) \quad (3.4).$$

The f term is the frequency (cycles per second), f_s is the sampling rate of the original signal (samples per second), k represents the index from the power spectrum, and N is the number of points in the original sensor signal. For the previous example, the dominant frequency would be calculated from the index value of 50. The original signal is sampled every minute, and there are 768 sensor values in the original signal. Using Equation 3.4, the dominant frequency is calculated as 1.085×10^{-3} cycles per second. The period can be calculated from the following equation [Bueche and Wallach, 1994]:

$$T = \frac{1}{f} \quad (3.5).$$

The period, T , is the time required to make one complete cycle. So, the period from the previous example is 922 seconds per cycle. The number of sampled sensor values in the period can be calculated from the following equation [Bueche and Wallach, 1994]:

$$T_n = T f_s \quad (3.3).$$

The number of points in the period, T_n is calculated from the sampling rate, f_s , and the period, T . For the previous example, the number of points in the period is 15.

The dead-zone should be set to the number of points for the dominant period. If the dead-zone is less than the dominant period then the extended signal tends to fluctuate due to the noise in the sensor signal. The following illustration with a periodic signal demonstrates how the dead-zone works.

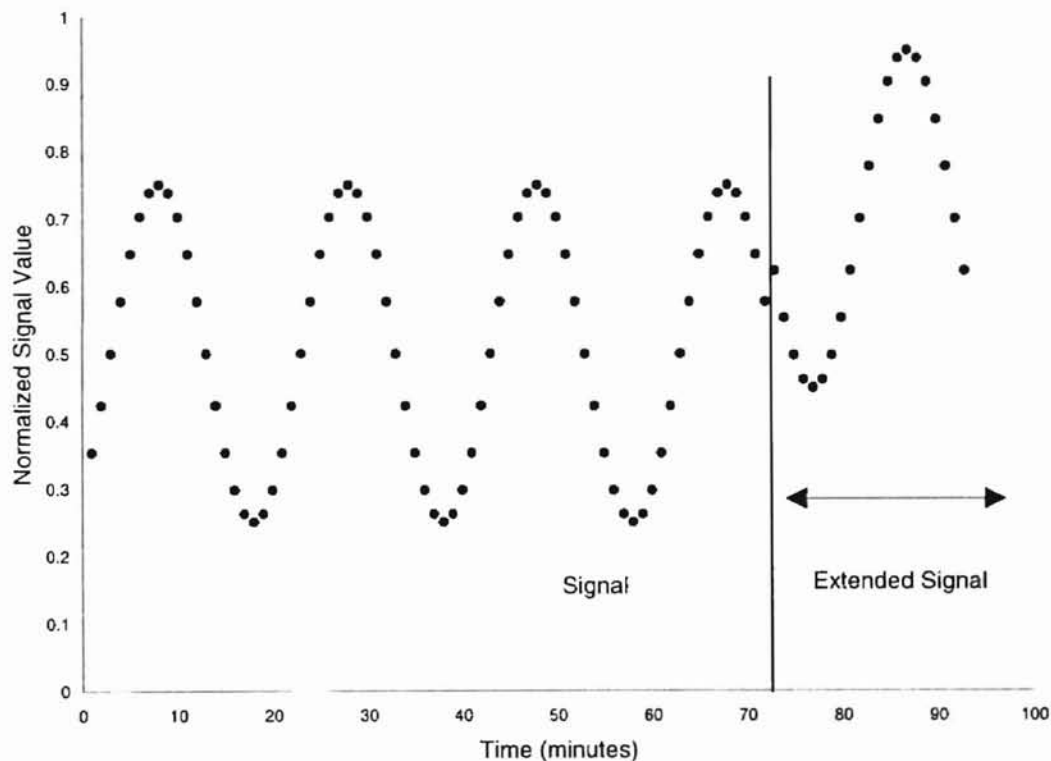


Figure 3-13: Extended signal with dead-zone less than dominant period of the sensor signal. Notice the extended signal is moved up.

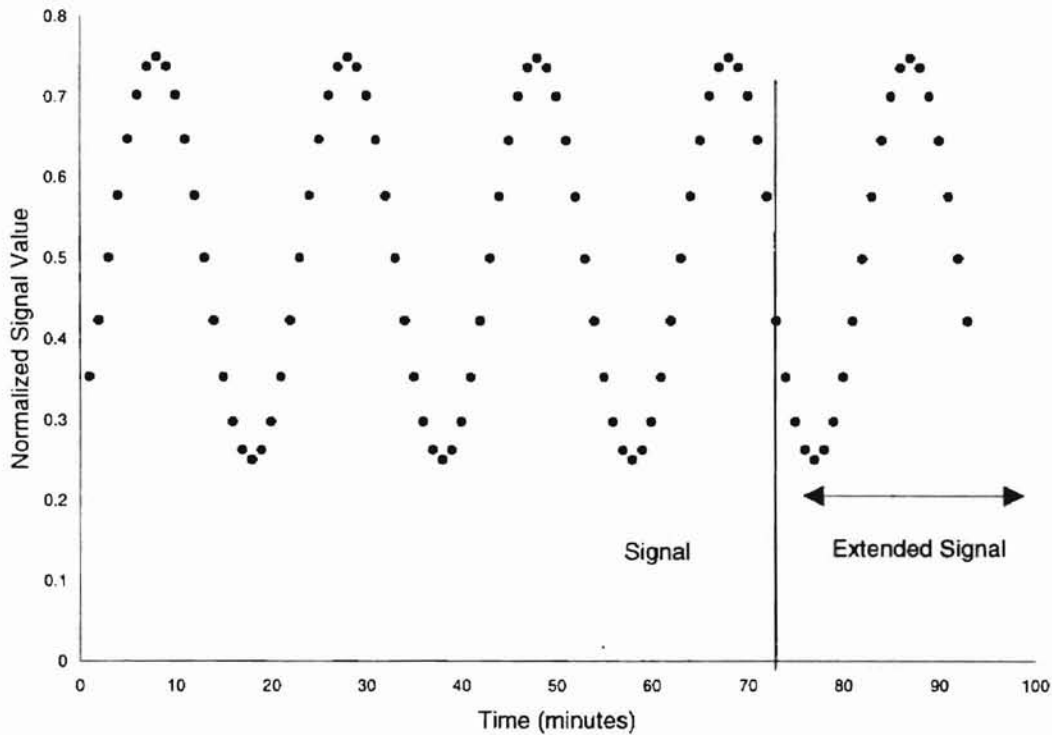


Figure 3-14: Extended signal with dead-zone equal to dominate period of the sensor signal. Notice the extended signal is very similar to the current sensor values.

When the dead-zone is much larger than the dominant frequency of the sensor signal, then the extended signal tends to be slower to react to actual changes in the sensor value. The larger dead-zone tends to damp any true changes in the trend of the sensor signal. When the dead-zone is smaller than the dominant sensor signal, then the extended signal tends to shift up and down when the signal should be stable as in the previous figures. Therefore, it is important to calculate the dead-zone accurately. All of the proposed extension techniques use the dead-zone to calculate the tie-point for the inverted symmetric extension similar to the Raghavan NET2 extension technique.

The following five sub-sections describe the improved real-time extension techniques proposed by the current author. All of the techniques employ adaptive

mechanisms to specify the threshold, dead-zone and tie-point in Raghavan's NET2 technique.

3.4 Adaptive NET2 Extension Technique

The first technique is an adaptive version of the Raghavan NET2 technique with two differences. First, the dead-zone, β , is established by the period of the dominant frequency rather than a fixed number. Second, the threshold number, α , is calculated from the following equation:

$$\alpha = 3\beta. \quad (3.7).$$

The threshold number, α , is limited to three times the dead-zone size, β . The tie-point for the Adaptive NET2 extension technique is calculated the same as the Raghavan NET2 technique. The extended part of the signal for the Adaptive NET2 is synthesized using Equation 3.3. The Adaptive NET2 extension technique is essentially the same as the Raghavan NET2 technique, but the dead-zone calculation is automated to do what Raghavan discovered empirically. The computer algorithm for the Adaptive NET2 extension technique is presented in Appendix B.

3.5 Square Root Objective Function Extension Technique

The second technique proposed is similar to the Adaptive Raghavan NET2 technique with a single exception. This technique uses a different criteria to calculate the tie-point. The dead-zone, β , is calculated based on the number of points in the dominant frequency of the sensor signal. Likewise, the threshold number, α , is calculated from equation 3.7.

A different objective function is used in the place of the mean squared error function defined for the Raghavan NET2 technique. The modified objective function is presented below:

$$\gamma_i = \frac{1}{\sqrt{n}} \sum_{i=1}^n (x_i - \bar{x})^2, \quad 3\beta \geq n > \beta \quad (3.8).$$

The only difference in Equation 3.2 and 3.8 is that a weighting factor of $1/\sqrt{n}$ is used instead of $1/n$. This has the effect of decreasing the weighting influence of the number of points used to calculate the mean squared error. The potential benefit of this modification is to generate a tie-point that has less squared error associated with that particular tie-point. The computer algorithm for the Square Root Objective Function extension technique is presented in Appendix B.

3.6 First Point Past Dead-Zone Extension Technique

The third technique proposed does not use an objective function of any sort to locate the tie-point. The tie-point is simply calculated as the mean for all points in the dead-zone plus the first point past dead-zone.

$$\bar{x}_{TP} = \frac{1}{\beta + 1} \sum_{i=\beta}^{\beta+1} x_i \quad (3.9).$$

This has two different effects. First, the tie-point calculation procedure is simplified. Second, tie-point is calculated with the least number of signal values. The potential benefit is that the tie-point is more responsive to any true changes in the sensor signal. The computer algorithm for the First Point Past Dead-Zone extension technique is presented in Appendix B.

3.7 Fuzzy Tie-Point Extension Technique

This proposed extension technique uses fuzzy logic [Zadeh, 1965] to calculate the tie-point. The appendix contains a section with background information on fuzzy logic.

The tie-point is determined using a fuzzy logic rule-base and approximate reasoning with two inputs. The two inputs are the mean squared deviation as defined in Equation 3.2 and the number of points used to calculate the mean squared deviation. The dead-zone is incorporated into the fuzzy logic technique. This tie-point is used to extend the signal based on the inverted symmetric extension technique in the normal fashion.

The procedure employed in this technique is as follows. First, the dead-zone is set to the period associated with the dominant frequency of the signal. Then the mean squared deviation is calculated with Equation 3.2. The values for both variables are then converted into fuzzy membership functions. The membership functions are adaptive depending on the dead-zone parameter. The baseline mean squared deviation, M_{β} , is calculated for the dead-zone (Equation 3.2 where $i = \beta$). All the subsequent M_i 's are scaled by M_{β} . The index number for every point is also scaled back by β . The membership functions are listed below.

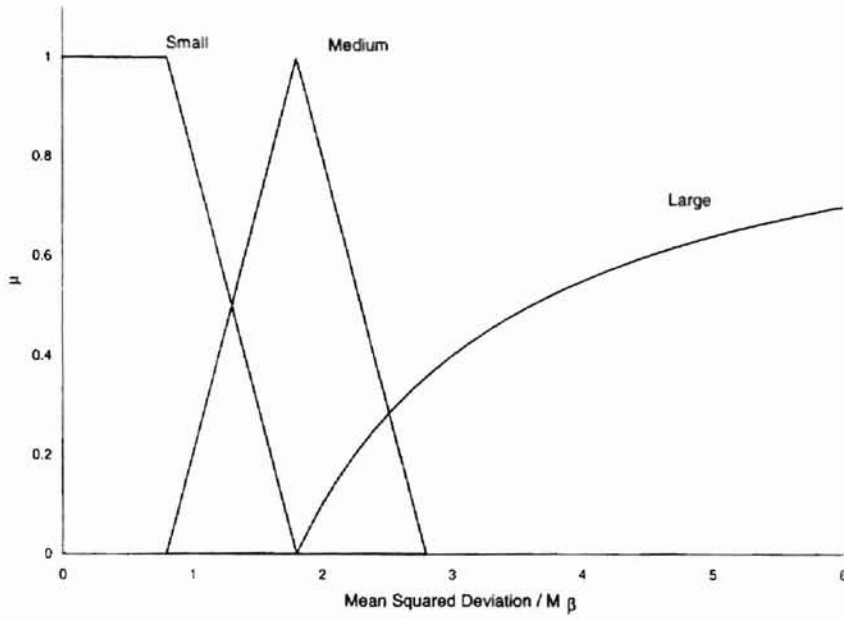


Figure 3-15: Fuzzy Tie-Point extension technique membership function for the mean squared deviation.

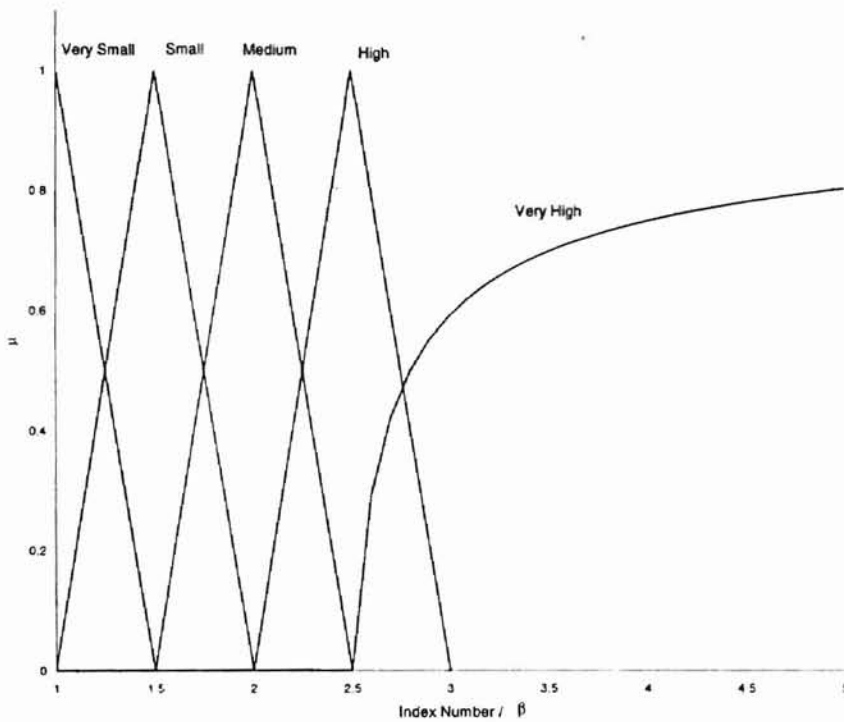


Figure 3-16: Fuzzy Tie-Point extension technique membership function for the index number.

Both the index number and the mean squared deviation variables are used to fire the tie-point evaluation rules. The following is the rule-base for the Fuzzy Tie-Point extension technique. The higher ratings are desired.

Table 3-I: Rule-base used to rate the tie-point for Fuzzy Tie-Point extension technique.

		Index Number		
		small	medium	large
Mean Squared Deviation	vsmall	vexc	good	ok
	small	good	ok	bad
	medium	exc	good	ok
	high	good	ok	bad
	vhigh	ok	bad	bad

The tie-point is desired with a smaller mean squared deviation so that the tie-point has less mathematical deviation from the rest of the signal. The smaller index number is desired so that the tie-point is more responsive to any true changes in the signal. The if-then rule-base works in the following way. With a small index number and a very small mean squared deviation, then the tie-point evaluation rating would be very excellent. Or, if the index number is medium and the mean squared deviation is high, then the tie-point evaluation rating would be ok.

This Fuzzy Tie-Point extension technique uses Mamdani's implication rule which is a max-min operator for each rule [Mamdani 1975; Mamdani and Assilian, 1975; Mamdani, 1977]. The rules are combined using "and" defined as the intersection (minimum) of each rule with the other rules. The output membership function is based on a rating calculated for each index number.

NORTH ALABAMA STATE UNIVERSITY

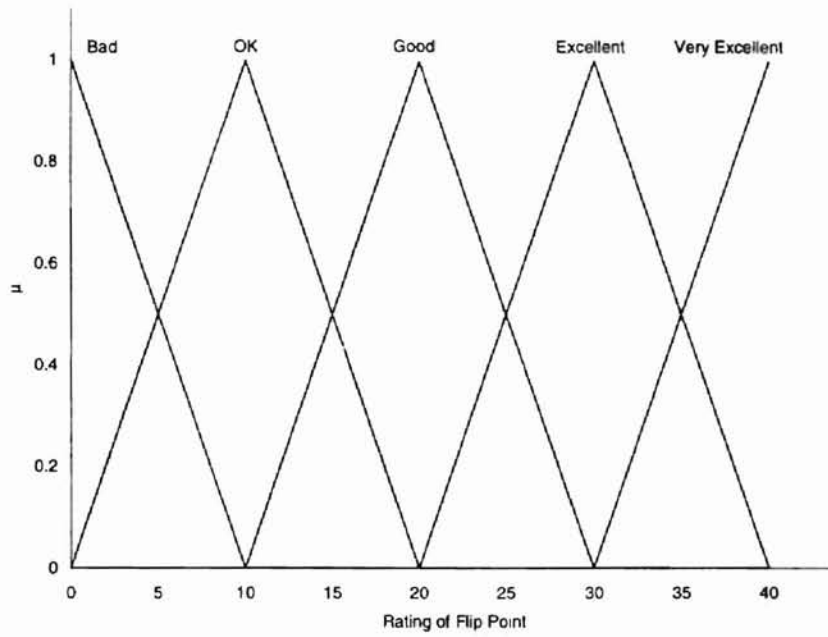


Figure 3-17: Fuzzy Tie-Point extension technique tie-point rating membership function.

After the rating membership function is determined, then the fuzzy membership function is converted into a crisp number. The rating membership function is defuzzified using the center of area defuzzification method [Brae and Rutherford, 1978; Lee, 1990a; Hellendorn and Thomas, 1993; Mendel, 1995]. The crisp rating value is calculated based on the following equation [Lee, 1990a]:

$$z = \frac{\sum_{j=0}^{40} \mu_z(x_j) \cdot x_j}{\sum_{j=0}^{40} \mu_z(x_j)} \quad (3.10).$$

The x_j terms are the rating value calculated from the rule-base and Figure 3-17. These x_j elements are discretized over j points ($j = 0$ to 40). The μ_z term is the membership value

ASTORIA STATE UNIVERSITY

for rating value from Figure 3-17. The output, z , is the crisp rating for each index number.

This technique takes the output membership function and converts into a crisp number using Equation 3.10. The output is a crisp number representing the fuzzy rating value of the tie-point at each index number. The tie-point is assigned as the mean corresponding to the index number with the highest rating. The signal is then extended using the inverted symmetric extension technique with the tie-point as defined as Equation 3.3.

The Fuzzy Tie-Point extension technique incorporates human intuition and experience in the calculation of the tie-point. The expected effect of this technique is to calculate a tie-point that is more reflective of the signal. The benefits of this technique is the separation of the two parameters used to calculate the tie-point, the index number and the mean squared deviation. This separation of the input parameters allows the possibility to weight each parameter independently of the other parameter. However, the fuzzy logic makes this extension technique more complex to implement than the other proposed techniques. The technique is also more computationally intensive than the other techniques. The computer algorithm for the Fuzzy Tie-Point extension technique is presented in the Appendix B.

3.8 Weighted Fuzzy Tie-Point Extension Technique

This technique modifies the procedure described in the previous subsection by use of a weighting factor [Sugeno and Murakami, 1985; Lee, 1990a] to rate each possible tie-point. All the points that lie in the dead-zone are assigned a weighting factor of one. The points laying outside the dead-zone are assigned a weighting factor using fuzzy logic. This weighting factor is based on the mean squared deviation and index number for each point.

The Weighted Fuzzy Tie-Point extension technique starts in the same manner as the Fuzzy Tie-Point extension method. The number of points in the dead-zone are used to calculate the mean square deviation. The two input variables are the scaled mean squared deviation and the scaled index number. Both values are converted into fuzzy values using the following membership functions:

UNIVERSITY OF ALABAMA STATE UNIVERSITY

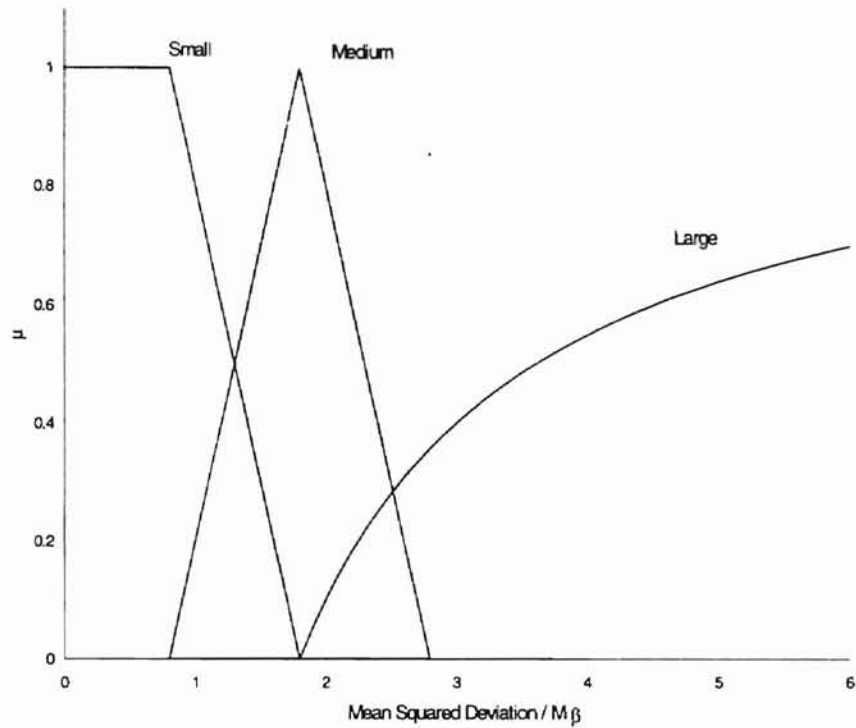


Figure 3-18: Weighted Fuzzy Tie-Point extension technique membership function for the mean squared deviation.

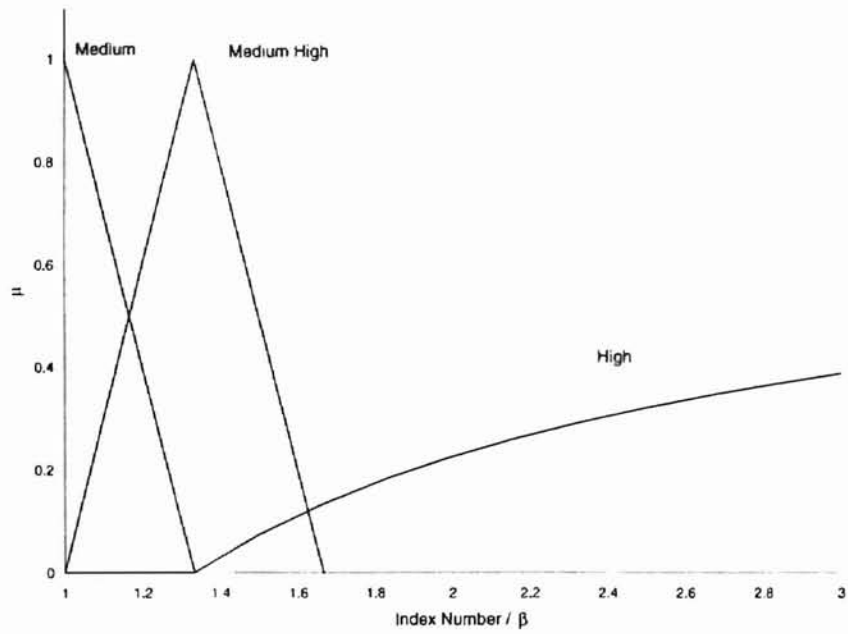


Figure 3-19: Weighted Fuzzy Tie-Point extension technique membership function for the index number.

ANNA AMBA SASTRI UNIVERSITY
 ANNA AMBA SASTRI UNIVERSITY
 ANNA AMBA SASTRI UNIVERSITY

Both the scaled index number and the scaled mean squared deviation variables are used to fire the tie-point evaluation rules. The following is the rule-base for the Weighted Fuzzy Tie-Point extension technique. The higher ratings are desired.

Table 3-II: Rule-base for rating each possible tie-point for the Weighted Fuzzy Tie-Point extension technique.

		Index Number		
		medium	med high	high
Mean Squared Deviation	small	exc	ok	bad
	medium	good	ok	bad
	large	ok	bad	bad

Each possible tie-point is rated for a smaller scaled mean squared deviation and a smaller index number. The if-then rule-base works in the following way. With a medium index number and a small mean squared deviation, then the tie-point evaluation rating would be excellent.

The technique uses Mamdani's rule of inference (max-min operation) with "and" defined as the intersection (minimum) of the values. The output membership function is the rating of each of the points generated by the following figure. The rating membership function is converted to a crisp number with the center of area defuzzification technique (Equation 3.10). This crisp rating value is used as the weighting factor.

NORTH CAROLINA STATE UNIVERSITY
 LIBRARY

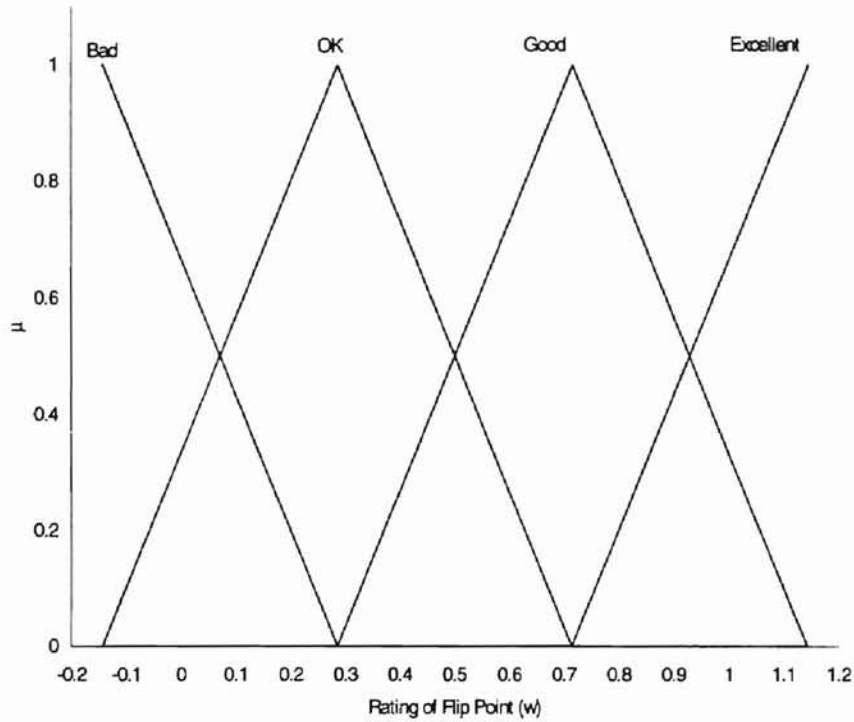


Figure 3-20: Weighted Fuzzy Tie-Point extension technique rating of each point.

The weighting factor is then used to calculate the tie-point from the following equation:

$$z = \frac{\sum_{i=\beta}^{\alpha} w_i \cdot x_i + \sum_{j=0}^{\beta} x_j}{\beta + \sum_{i=\beta}^{\alpha} w_i} \quad (3.10).$$

The weighting factor, w_i , is calculated from the rule-base and Figure 3-20. The x_i terms are the possible tie-points that are used to generate the weighting factors (w_i). There are β points in the dead-zone while $\alpha - \beta$ points lie outside the dead-zone. The β points laying in the dead-zone each receive a weighted factor (w_i) of one while the $\alpha - \beta$ points laying

outside the dead-zone have a weighting factor (w_i) assigned as described previously. The actual tie-point (z which is a crisp value) is then used in the extension technique. The signal is then extended around the tie-point using the inverted symmetric extension technique with Equation 3.3.

This Weighted Fuzzy Tie-Point extension technique is based on human experience and intuition. The tie-point is calculated over the range of sensor values from the most current sensor value up to the threshold sensor value. The benefit of this extension technique is that the tie-point represents more sensor values than the other proposed extension techniques. This hopefully results in a tie-point which is more representative of the sensor signal. The drawbacks of this technique is that it is more computationally intensive to calculate the tie-point. This technique, like the other Fuzzy Tie-Point extension technique, is more complex to implement than the other extension techniques. The complete computer algorithm for the Weighted Fuzzy Tie-Point extension technique is presented in the Appendix.

3.9 Chapter Summary

The convolution operation used to implement wavelet smoothing causes the end-distortion of the extracted trend. An extension technique can be used to minimize if not eliminate the end-distortion. To smooth a windowed sensor signal, the signal is extended on each end. The extended signal is then wavelet smoothed. The extended part of the

smoothed signal is removed to leave the smoothed signal window without end-distortion. After Raghavan examined the existing traditional extension technique, he concluded that none of these existing techniques performed adequately [Raghavan, 1995]. He then proposed the Raghavan NET2 technique based on the inverted symmetric extension of the signal. The extended signal is appended to the windowed sensor signal at the tie-point. The tie-point is the key parameter in the Raghavan NET2 technique. The calculation of the tie-point is based on empirical research for the sensor signal that Raghavan examined.

This thesis presents an adaptive method to calculate the tie-point based on the characteristics of the sensor signal. The tie-point is calculated based on the period of the dominant frequency of the sensor signal. The dominant frequency is determined from the power spectrum plot of the sensor signal. This thesis presents five proposed extension techniques. All proposed extension techniques use the adaptive dead-zone size to extend the signal with an inverted extension method. The first proposed extension technique is the Adaptive NET2 extension technique. This technique is essentially the same as the Raghavan NET2 technique except the tie-point is calculated with an adaptive dead-zone. The second technique, the Square Root Objective Function extension technique, uses a different objective function to calculate the tie-point. The third technique, the First Point Past Dead-Zone extension technique, uses only the sensor values in the dead-zone plus one to calculate the tie-point. The last two proposed extension techniques incorporate fuzzy logic. One technique, the Fuzzy Tie-Point extension technique, uses the fuzzy logic to calculate the tie-point. The other, the Weighted Fuzzy Tie-Point extension technique,

A
S
T
R
A
L
S
C
I
E
N
C
E
S
I
N
T
E
R
N
A
T
I
O
N
A
L
J
O
U
R
N
A
L
O
F
S
O
C
I
E
T
Y
A
N
D
I
N
D
I
A
N
A
C
A
D
E
M
Y
O
F
S
C
I
E
N
C
E
S

calculates the tie-point using a weighting factor determined from the fuzzy logic reasoning.

Associação Brasileira de Normas Técnicas
ABNT NBR 13274-10000

Chapter Four: Performance of Proposed Extension Techniques

4.1 Introduction

This chapter documents the performance of the five proposed extension techniques and the Raghavan NET2 extension technique. Three different criteria are used to evaluate the trend extraction properties of each technique. The first rates the ability of each extension technique to minimize end-distortion. The second evaluates the robustness of the extension techniques when the size of the dead-zone differs from the period of the dominant frequency in the sensor signal. The final test quantifies the effect of signal sampling rate on trend extraction performance. For each evaluation procedure, both the rationale behind the procedure and the quantification method are elaborated. Results for each of the extension techniques are presented. The results of all of the three evaluations are used to identify the “best” of the proposed extension techniques.

4.2 Ability to Minimize End-Distortion

To be useful, an extension technique must make the trend of the smoothed signal accurately match the trend of the raw sensor data. The perfect extension technique would eliminate all end-distortion caused by the convolution operation used to implement the wavelet transforms. In this perfect smoothing situation, the extracted trend at the end of

the signal would be the same as if the extracted trend was generated from the middle of the sensor signal because the future trend of the sensor signal beyond the window of interest is actually known. Consequently, for the first evaluation criteria, which measures end-distortion, we elected to use the extracted trend as smoothed in the middle of the signal as the test standard. This test standard is then compared to the smoothed trend as extracted from the real-time end of the signal. The following figure shows a sensor signal when the trend is extracted from the middle of the signal. This trend represents the test standard.

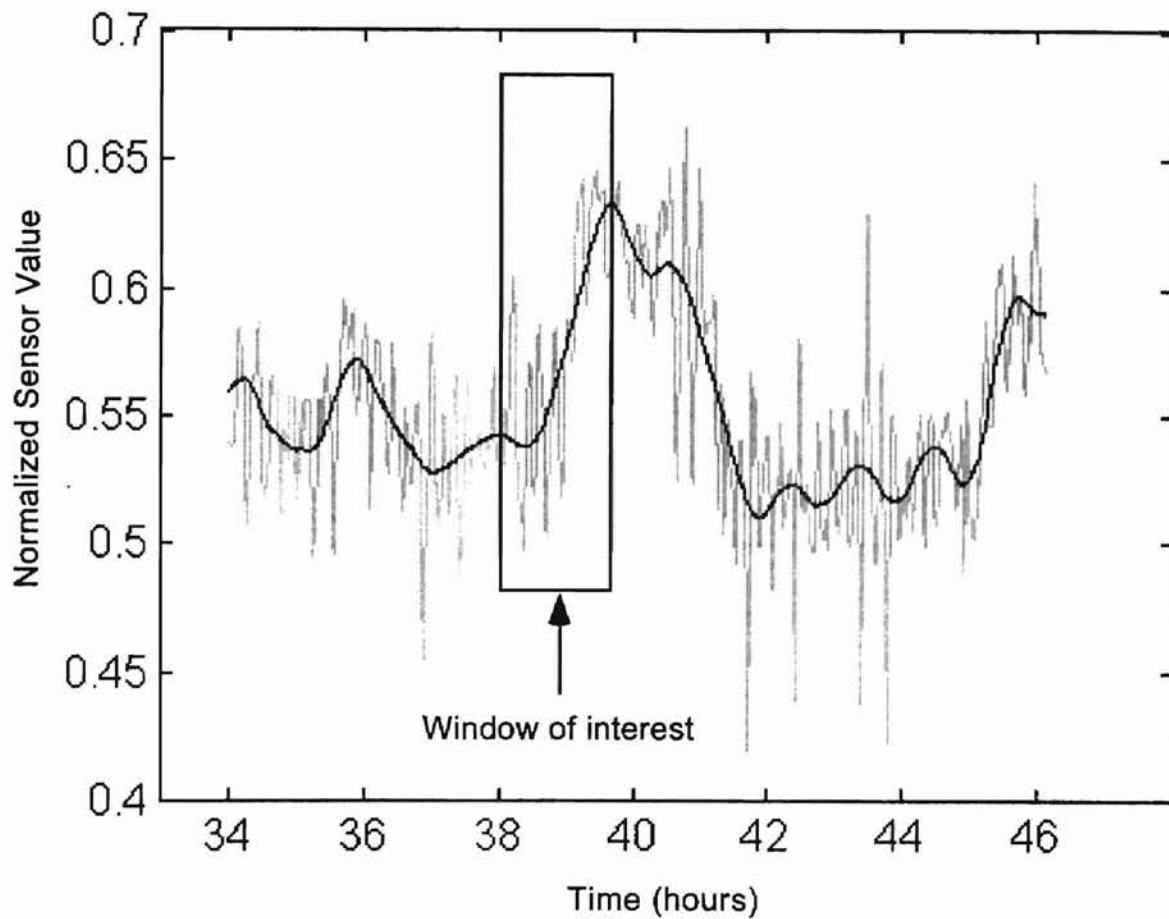


Figure 4-1: The test standard: Extracted trend as smoothed in the middle of the signal.

The following figure shows the same sensor signal smoothed at the real-time end of the signal using the First Point Past Dead-Zone extension technique.

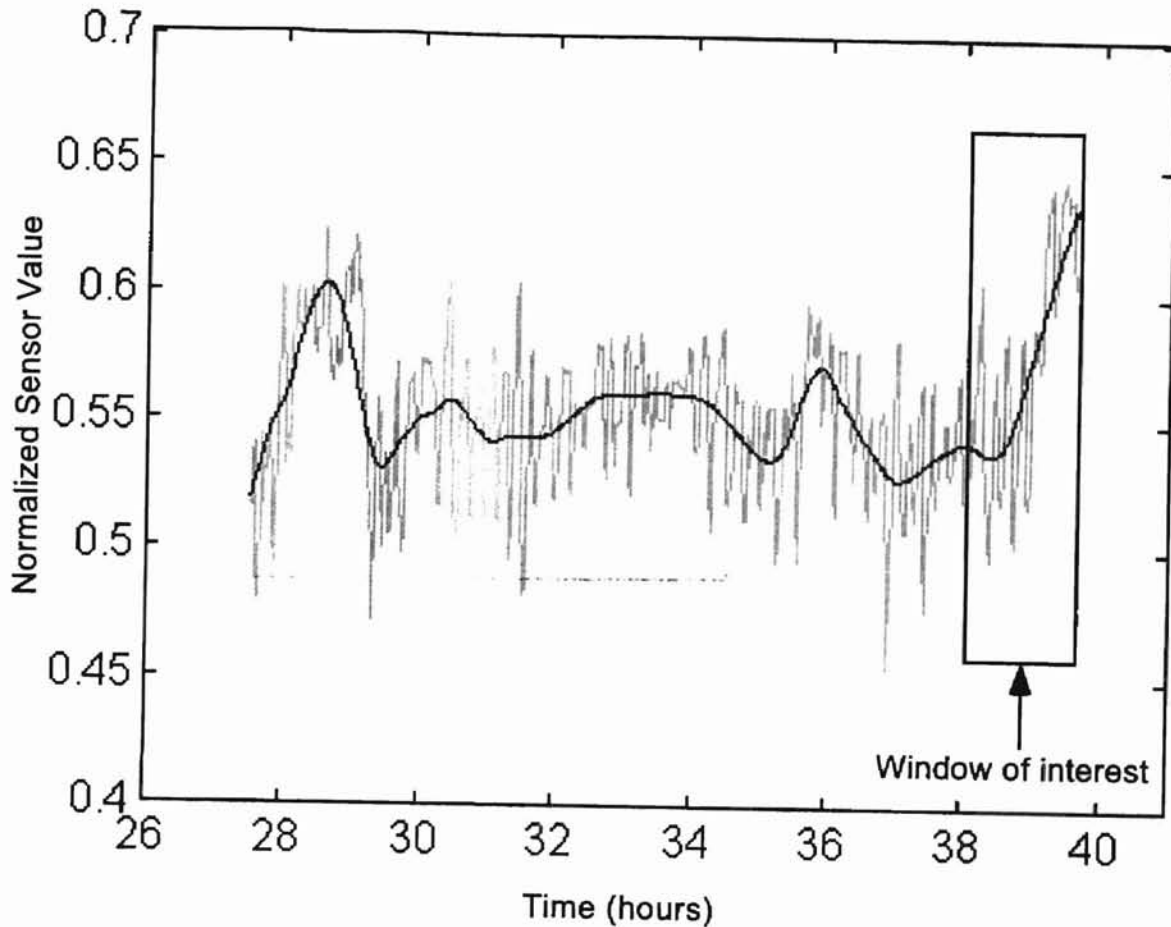


Figure 4-2: Compared trend: Extracted trend as smoothed at the end of the signal.

The closer the two extracted trends in Figures 4-1 and 4-2 match each other, the better the extension technique performed in minimizing end-distortion. If the two extracted trends match each other perfectly, then the extension technique eliminated all end-distortion.

This comparison technique provides a means to quantify trend extraction fidelity.

This is the quantitative test procedure.

1. Select the window of sensor data to be used for evaluation purposes. For all of the results reported in this thesis, a window size of 768 sensor values was used with a sample period of one minute. The windowed signal is extended to 1024 points (128 points on each side).
2. Wavelet smooth the extended signal from step one using the 6th order Daubechies wavelet family with four levels of decomposition. After reconstructing the smoothed signal, trim the 128 points on each end of the signal which are affected by end-distortion. The remaining 768 point smoothed signal is the comparison case.
3. Take the last 384 sensor values from the window of interest from step one. Then append the next 640 sensor values using sensor values from the original sensor data. This results in a signal of 1024 points. Wavelet smooth the 1024 windowed signal using the 6th order Daubechies wavelet family with four levels of decomposition. After reconstructing the smoothed signal, trim the last 640 points on right side of the smoothed signal. The remaining 384 point smoothed signal is the test standard.
4. Sample the comparison and test standards at every third value for 16 sample points. The sampling for the comparison case starts on the real-time end and proceeds back in time. The test sample is sampled so that the sampled points for the test case correspond to the same points in time as the comparison case.

5. Compare the two sampled trends by calculating the absolute difference between the sampled points for both trends. The following equation is used:

$$d = \sum_{i=0}^{16} |x(i) - y(i)| \quad (4.1).$$

The d term is the absolute difference between the sampled standard trend $x(i)$ and the sampled comparison trend $y(i)$ with i representing sample index ($\max(i) = 16$ in both cases).

6. Compare the two sampled trends by calculating the squared difference between sampled points for both trends. The following equation is used:

$$s = \sum_{i=0}^{16} [x(i) - y(i)]^2 \quad (4.2).$$

The s term is the squared difference between the sampled standard trend as given by $x(i)$ and the sampled comparison trend given by $y(i)$.

This evaluation procedure is repeated after shifting the window one minute in time. For a source signal of 10,080 points with 6845 windows are analyzed.

The analysis was performed using seven days of actual plant data and two different sensors. The first sensor is a flow meter. The characteristics of this sensor are very representative of the usual sensor encountered in a plant situation. Sensor measurements were first normalized to a 0 - 1 scale using the sensor zero and the measurement span. The next figure is the flow sensor signal for the seven days worth of data.

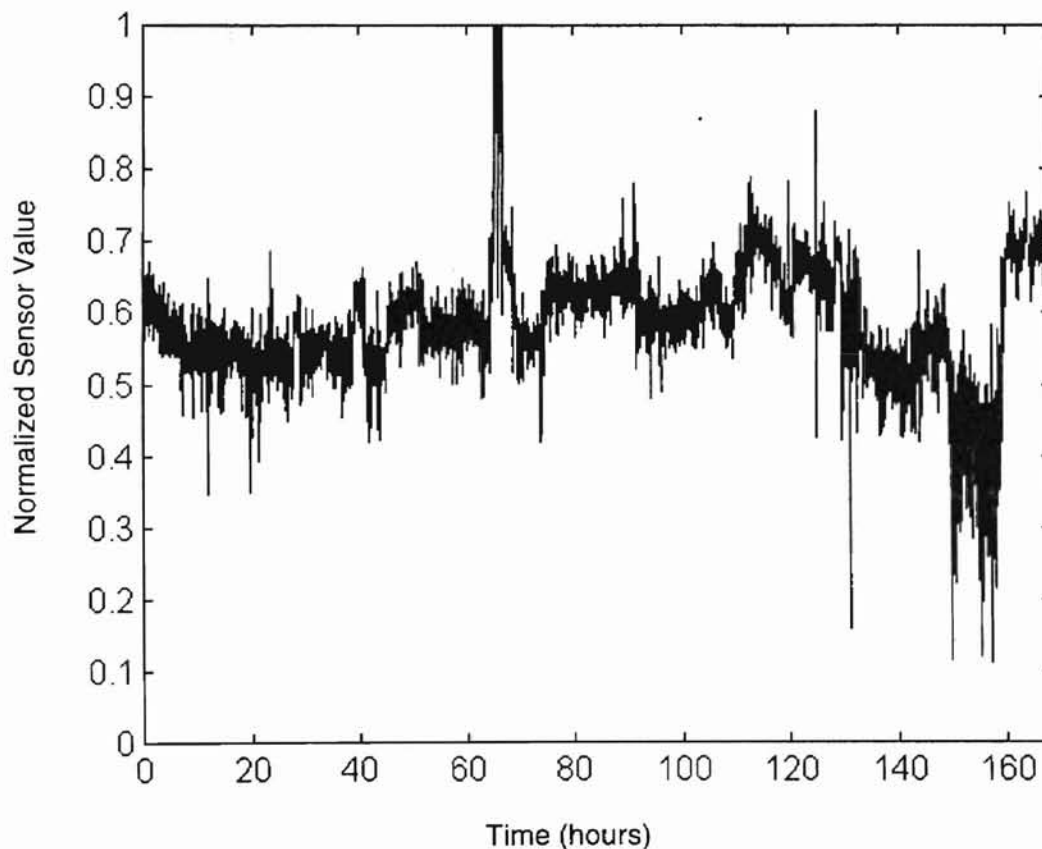


Figure 4-3: Flow meter sensor signal for the seven days. Sample period is one minute.

The period of the dominant frequency for this sensor was determined from the power spectrum which is presented in the following figure. The power spectrum plot was generated from the sensor values from Figure 4-3 using only values from 78 to 95 hours. The sample period is one minute so there are a total of 1000 sensor values used to generate the power spectrum plot.

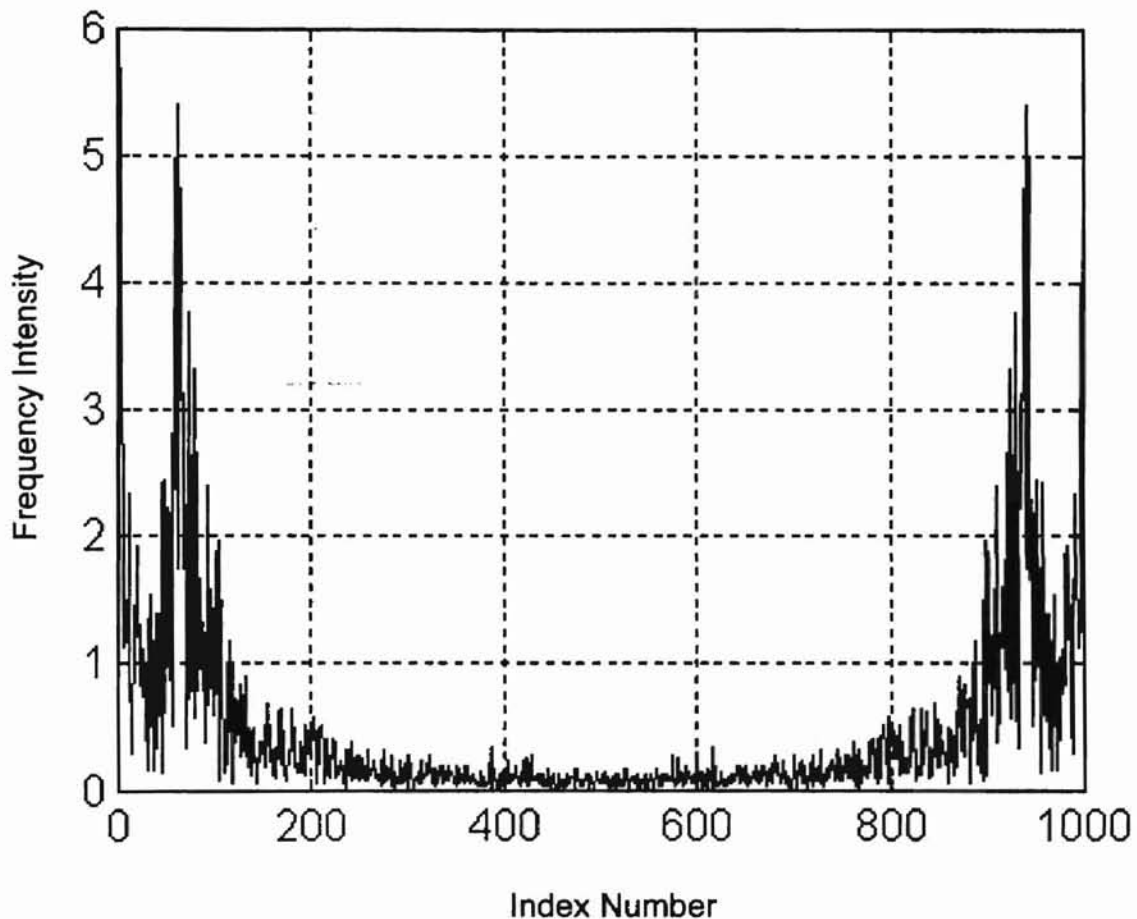


Figure 4-4: Flow meter sensor power spectrum plot. 1000 sensor values used with a sample period of one minute.

From a detailed examination of Figure 4-4, there is a spike at the index value of 62. So using Equations 3.4, 3.5, and 3.6, the period of the dominant frequency is about 16 minutes. This period corresponds to a dead-zone of 16 points for the flow sensor.

The second sensor used for evaluation purposes was a temperature sensor. The following figure shows the seven days worth of data which was used.

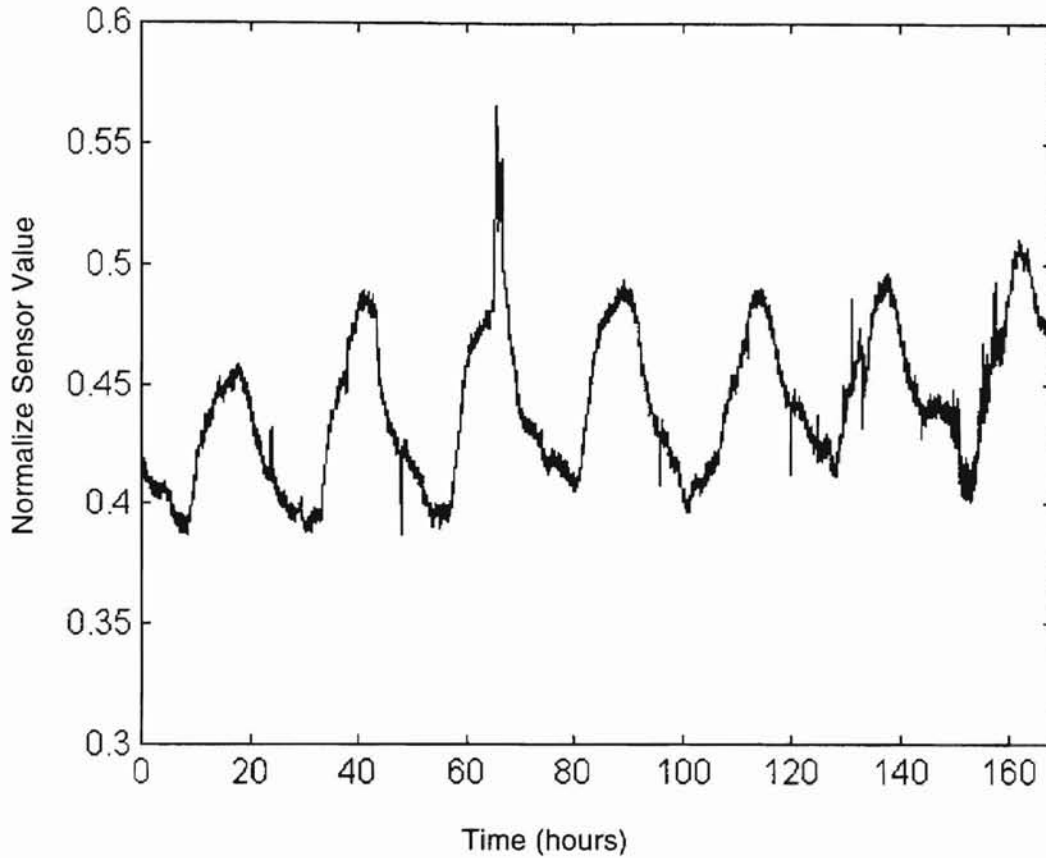


Figure 4-5: Temperature sensor signal for the seven days. Sample period is one minute.

Figure 4-6 presents the power spectrum plot for the temperature sensor. The power spectrum plot was generated from the sensor values from Figure 4-5 using only values from 78 to 95 hours with a sample period of one minute. This corresponds to 1000 sensor values. From a more detailed examination of Figure 4-6, there is a large spike at the index of 33. Using equations 3.4, 3.5, and 3.6, this corresponds to a period of 30

minutes for the dominant frequency. A dead-zone size of 30 points was used for the temperature sensor.

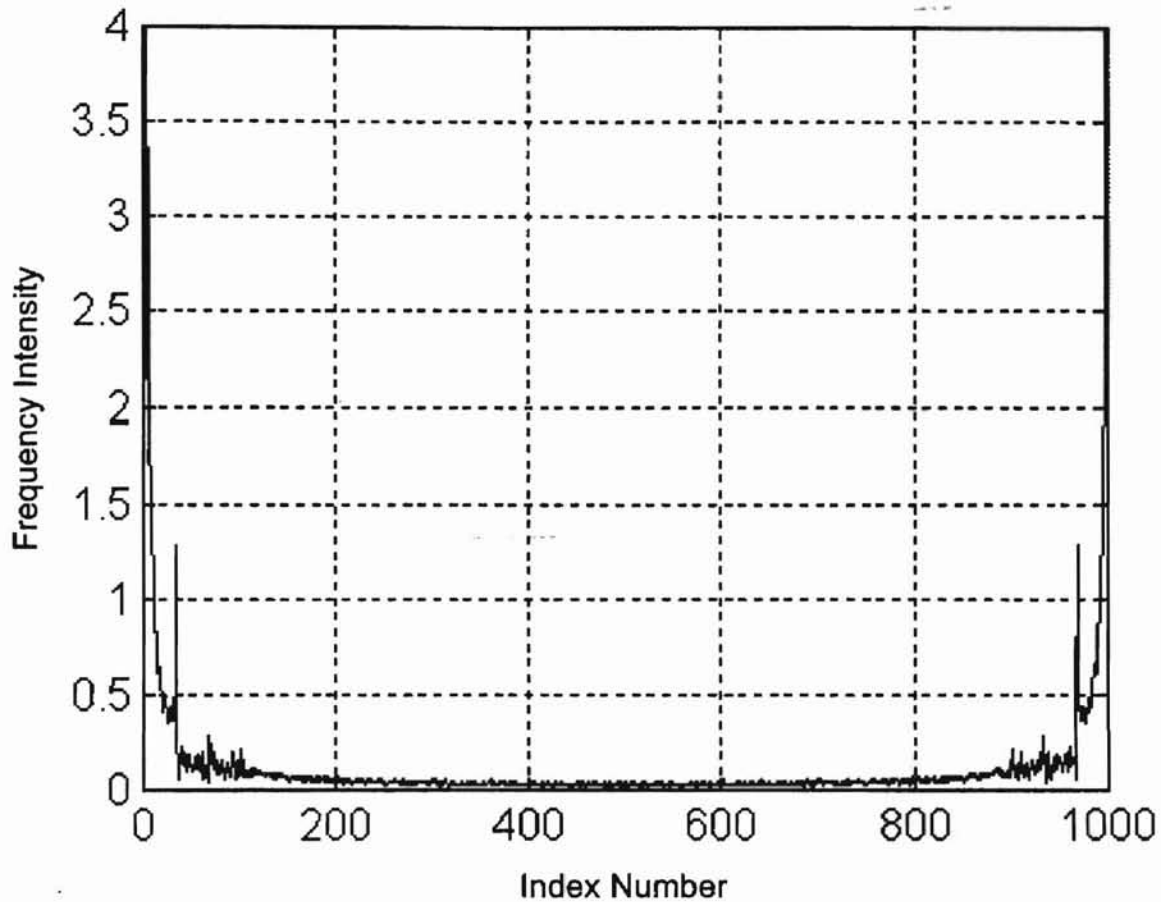


Figure 4-6: Temperature sensor power spectrum plot. 1000 sensor values were used with a sample period of one minute.

In both cases, the seven days of data provided 6845 test cases. The maximum absolute difference and maximum squared difference over all 6845 case evaluations are presented for the flow meter sensor in Table 4-I. These maximums quantify the worst performance of each extension technique. When the worst case for one extension technique is compared to the other extension techniques, it can be determined as to which

extension technique does the best at minimizing end-distortion under the worst case conditions. The worst case occurs when the sensor trend exhibits an abrupt change.

The other test measurements are the sum of the absolute difference and sum of the squared difference which are also tabulated in Table 4-I for all 6845 cases. This quantifies the performance of each extension technique over the entire seven day test period. Lower scores are indicative of better performance.

Table 4-I: Evaluation of extension techniques to minimize end-distortion for the flow meter sensor.

Flow Meter Sensor				
Extension Technique	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Raghavan NET2	0.081	0.92	9.83	574
Adaptive NET2	0.194	1.42	7.07	344
Squared Objective Function	0.152	1.22	5.73	323
First Point Past DZ	0.076	0.78	4.84	314
Fuzzy Winner	0.134	1.34	5.91	322
Weighted Fuzzy	0.082	0.92	4.93	309

The maximum values for the maximum squared difference and the maximum absolute difference occur when the flow meter undergoes a change in the fundamental sensor trend. The Raghavan NET2, the First Point Past Dead-Zone, and the Weighted Fuzzy extension techniques all have essentially the same maximum squared difference for the flow meter. The First Point Past Dead-Zone has a slightly lower maximum absolute difference than the other two extension techniques. This means that these three extension techniques all perform better in adjusting the tie-point when there are fundamental changes in the sensor signal. These three techniques perform better at minimizing the

end-distortion when the sensor undergoes a sudden change in the fundamental sensor trend. The other three extension techniques, the Adaptive NET2, the Squared Objective Function, and the Fuzzy Tie-Point, all have higher maximum squared and maximum absolute difference values. These extension techniques do not adjust the tie-point as fast as the other three extension techniques when the fundamental sensor trend changes.

The sum of the squared difference and the sum of the absolute difference provide insight into the performance of each extension technique used to calculate a tie-point over an extended period of time. When examining the sum of squared difference and the sum of absolute difference, the Weighted Fuzzy technique and the First Point Past Dead-Zone perform better than the other four extension techniques. The Squared Objective Function and the Fuzzy Tie-Point extension techniques perform better than the Adaptive NET2 and the Raghavan NET2 extension techniques. The Raghavan NET2 extension technique performs much worse than the other extension techniques. This technique performs almost twice as bad as the Weighted Fuzzy and the First Point Past Dead-Zone extension technique in selecting a tie-point over all the 6845 cases. The Raghavan NET2 technique does a poor job at minimizing the end-distortion based on the overall analysis. The Raghavan NET2 technique has a tendency to overreact. Poor empirical evidence of this tendency was the driving factor for the work reported in this thesis.

In summary, for the flow meter sensor, the Raghavan NET2, the Weighted Fuzzy, and the First Point Past Dead-Zone extension techniques perform better at selecting a tie-point when the sensor undergoes fundamental changes. This leads to a better job in

minimizing the end-distortion for the sudden change in fundamental sensor trend.

However, when examined over the 6845 cases, the Raghavan NET2 technique does not perform well at minimizing the end-distortion. The Weighted Fuzzy and the First Point Past Dead-Zone extension techniques have less of a tendency to overreact that is reflected in lower cumulative differences over time. These two extension techniques, Weighted Fuzzy and the First Point, are the better extension techniques for the flow meter sensor. The following figures show the worst case for the First Point Past Dead-Zone and the Weighted Fuzzy extension techniques.

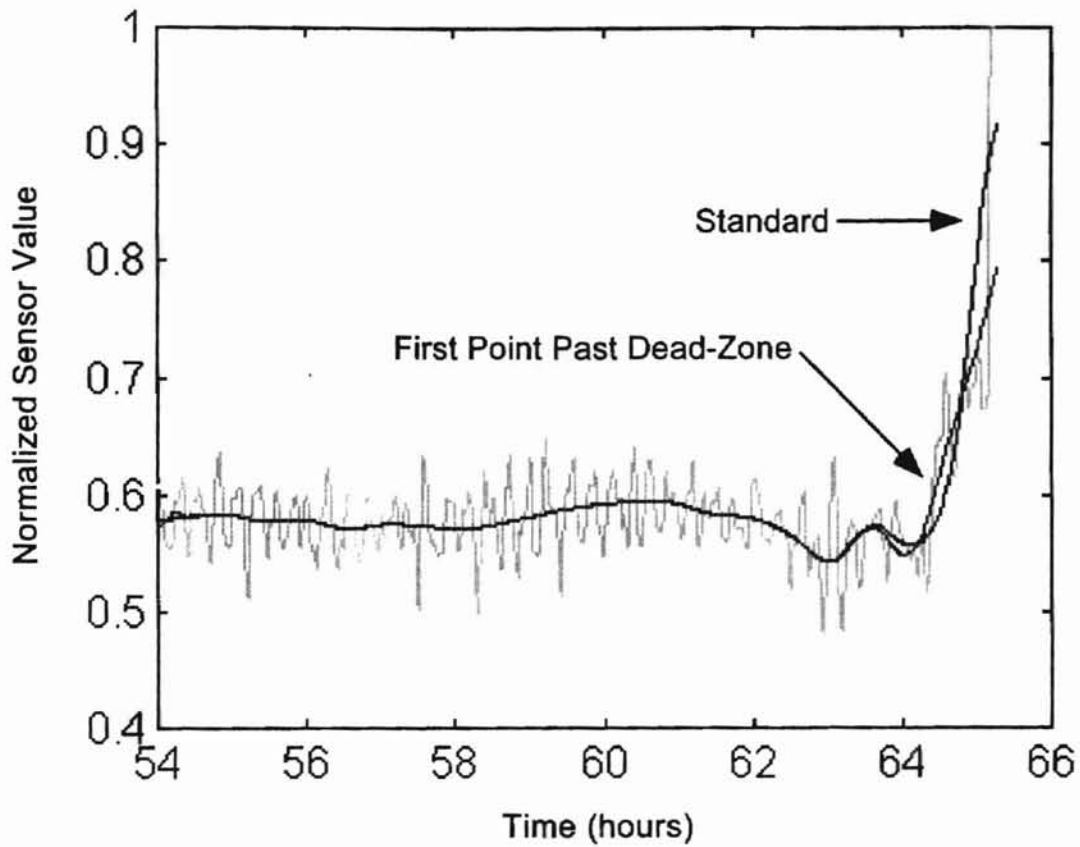


Figure 4-7: Worst case for smoothed flow meter sensor signal for the seven days using First Point Past Dead-Zone extension technique.

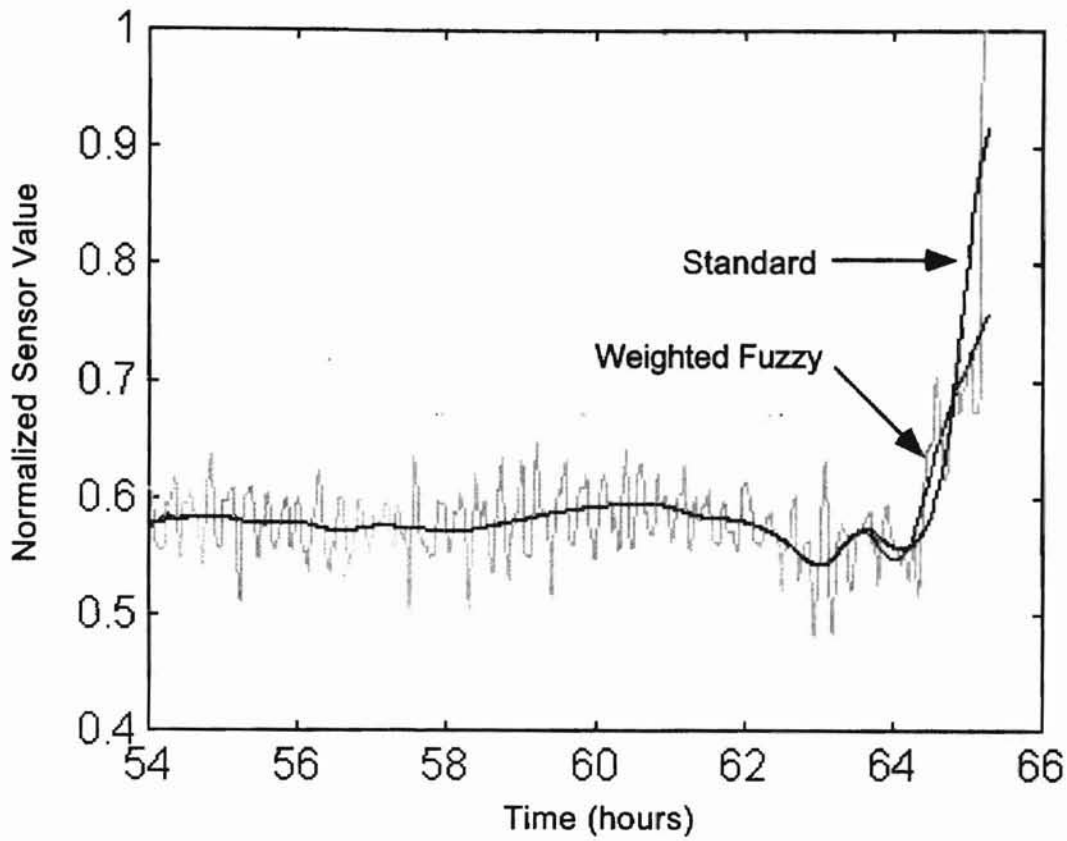


Figure 4-8: Worst case for smoothed flow meter sensor signal for the seven days using Weighted Fuzzy Tie-Point extension technique.

The same tests were performed using the temperature sensor. This sensor has a large natural period with a small high frequency noise content when compared to the previous flow meter. The maximum absolute difference and maximum squared difference for the 6845 case evaluations are presented for the temperature sensor in Table 4-II. These maximums quantify the worst performance of each extension technique. The sum of absolute difference and sum of squared difference are also presented in Table 4-II.

Table 4-II: Evaluation of extension techniques to minimize end-distortion for the temperature sensor.

Temperature Sensor				
Extension Technique	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Raghavan NET2	0.0010	0.12	0.132	64
Adaptive NET2	0.0138	0.37	0.527	99
Squared Objective Function	0.0068	0.27	0.419	93
First Point Past DZ	0.0070	0.27	0.403	92
Fuzzy Winner	0.0104	0.34	0.504	91
Weighted Fuzzy	0.0075	0.28	0.449	99

The Raghavan NET2 technique performs much better than the other extension techniques. This improved performance results from the fact that the Raghavan NET2 technique has a small dead-zone and the temperature sensor has a very small amplitude of noise with a large period. The Raghavan NET2 technique can respond to any fluctuations in the temperature sensor much faster with the small dead-zone size. This means that the maximum absolute difference and the maximum squared difference for the Raghavan NET2 technique are much smaller than the other extension techniques. Because the Raghavan NET2 technique uses such a small dead-zone size, the sum of absolute difference and the sum of squared difference are also smaller than the other extension techniques.

Among the five proposed extension techniques, the Weighted Fuzzy, the Squared Objective Function, and the First Point Past Dead-Zone extension techniques all have the lowest maximum absolute difference and maximum squared difference. These three techniques perform better at minimizing the end-distortion when the temperature sensor undergoes a fundamental trend change. All of the proposed extension techniques seem to

work equally well at minimizing the end-distortion over the entire 6845 cases. None of the proposed extension techniques perform significantly better than any of the other extension techniques.

In conclusion, the Raghavan NET2 technique shows better performance on the temperature sensor than the other proposed extension techniques. This results from a small dead-zone size that allows the Raghavan NET2 technique to adjust the tie-point quickly to reflect changes in the sensor. The other extension techniques use a larger dead-zone size that limits the response time to any fluctuations in the sensor. Of the five proposed extension techniques, the Squared Objective Function, the Weighted Fuzzy, and the First Point Past Dead-Zone extension techniques all performed about equally well. The following figures show the worst case for the Weighted Fuzzy and the First Point Past Dead-Zone extension techniques.

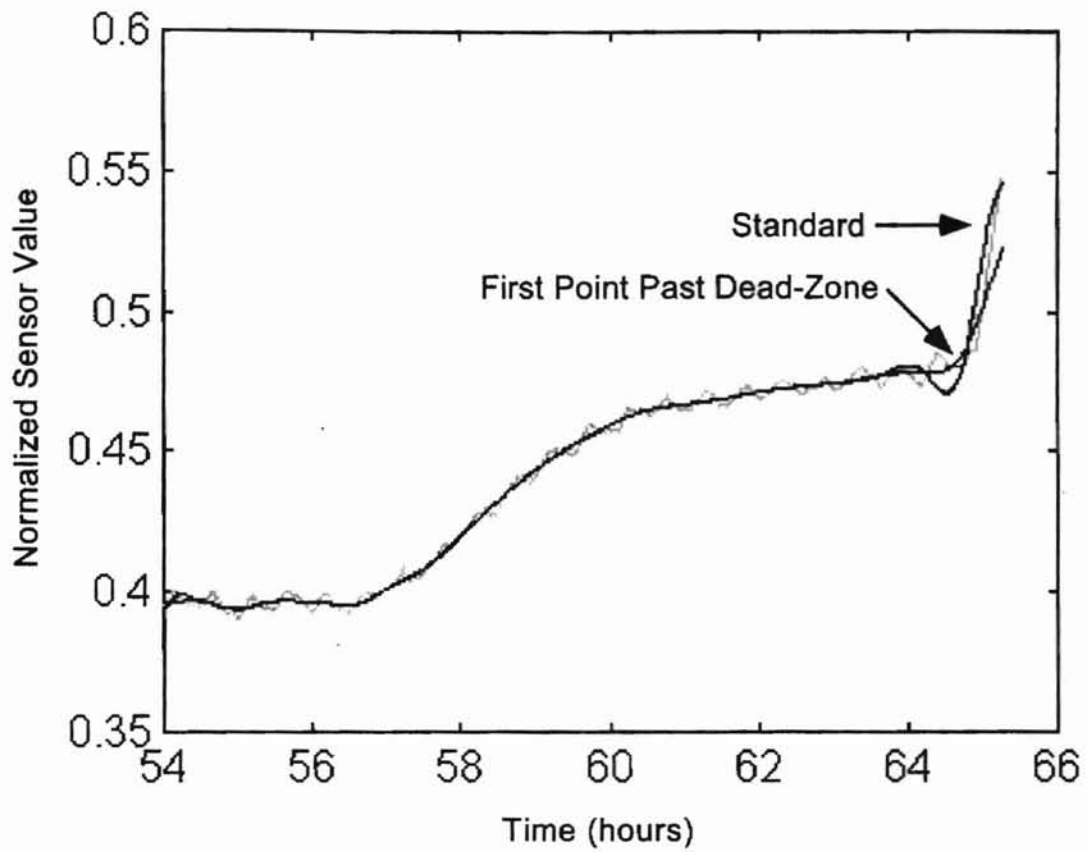


Figure 4-9: Worst case for smoothed temperature sensor signal for the seven days using First Point Past Dead-Zone extension technique.

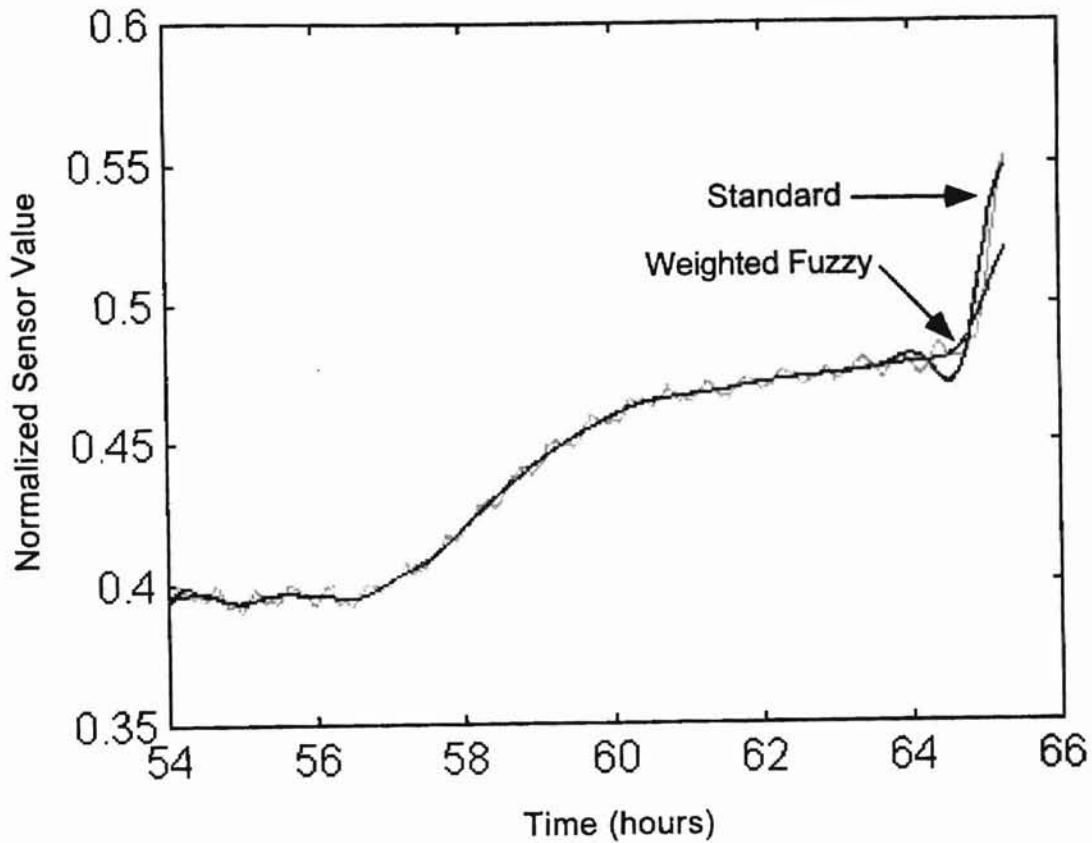


Figure 4-10: Worst case for smoothed temperature sensor signal for the seven days using Weighted Fuzzy Tie-Point extension technique.

More sensors encountered in plant situations exhibit dynamics more similar to the flow meter than the temperature sensor. For this reason, the flow meter sensor results are weighted more heavily when evaluating the overall performance of each extension technique. Consequently, we conclude the Weighted Fuzzy extension technique and the First Point Past Dead-Zone extension technique perform better at minimizing the end-distortion than the other proposed extension techniques.

4.3 Robustness Evaluation

The purpose of this test was to evaluate performance with an inaccurate estimate of the dead-zone. The top two performers from the end-distortion test, the First Point Past Dead-Zone and the Weighted Fuzzy extension technique, as well as the Adaptive NET2 extension technique, were selected for evaluation.

All of the proposed extension techniques use the dead-zone as calculated from the dominant frequency of the sensor signal (as explained in Section 3.3). However, there can be some difficulty in calculating the dominant frequency. This robustness test evaluates performance when different values of the dominant frequency are used to set the dead-zone size.

The test was performed using the same data for the flow meter and the temperature sensor shown in Figures 4-3 and 4-5, respectively. The test procedure is identical to the end-distortion test except that different values of the dead-zone were used in the extension techniques. For the robustness test on the flow meter, the dead-zone size is decreased by 25 percent and then 50 percent for dead-zone sizes of 12 and 8 points. The baseline dead-zone size of 16 points is then increased by 25 percent and 100 percent for dead-zone size of 20 and 32 points.

Table 4-III contains the robustness evaluation for the flow meter. The size of the dead-zone decreases moving down the table.

Table 4-III: Evaluation of extension technique for robustness on flow meter.

Flow Meter Sensor				
	Dead Zone = 32			
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.228	1.53	12.20	437
First Point Past DZ	0.137	1.22	8.77	387
Weighted Fuzzy	0.168	1.37	10.20	415

	Dead Zone = 20			
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.228	1.52	8.68	373
First Point Past DZ	0.085	0.98	5.44	329
Weighted Fuzzy	0.110	1.08	6.32	340

	Dead Zone = 16			
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.194	1.42	7.07	344
First Point Past DZ	0.076	0.78	4.84	314
Weighted Fuzzy	0.082	0.92	4.93	309

	Dead Zone = 12			
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.151	1.28	5.52	324
First Point Past DZ	0.060	0.77	3.93	310
Weighted Fuzzy	0.065	0.79	3.91	294

	Dead Zone = 8			
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.105	1.08	5.40	370
First Point Past DZ	0.060	0.72	4.93	390
Weighted Fuzzy	0.063	0.73	4.37	360

For all three extension techniques, the maximum absolute difference and the maximum squared difference decreases as the dead-zone decreases. The smaller dead-zone allows the tie-point to adjust faster to reflect changes in the fundamental sensor trend. The sum of absolute difference and the sum of squared difference also decreases as the dead-zone size decreases up to the smallest dead-zone. At the smallest dead-zone size, the tie-point is subject to noise fluctuations in the sensor signal. These noise fluctuations cause the extension techniques to calculate a tie-point that is not reflective of the actual sensor trend. This leads to the increase in the sum of absolute difference and sum of squared difference. In general, a smaller dead-zone gives better performance than a larger dead-zone size. However, if the dead-zone size is too small, then the extension technique does not perform well at all.

Both the First Point Past Dead-Zone and the Weighted Fuzzy extension techniques perform better than the Adaptive NET2 technique at the larger dead-zone sizes. The First Point Past Dead-Zone extension technique has lower values for all four evaluated criteria for the larger dead-zone sizes. However at dead-zone sizes of 16 and 12 points, both the First Point Past Dead-Zone and the Weighted Fuzzy extension techniques perform equally as well. At the smallest dead-zone size, all three techniques perform about the same. Overall, First Point Past Dead-Zone extension technique performs slightly better than the Weighted Fuzzy and significantly better than the adaptive NET2 technique. If there is a problem in estimating the dead-zone, it is better to underestimate rather than overestimate the size of the dead-zone.

The robustness test was also performed using the temperature sensor. For temperature sensor, the baseline dead-zone is 30 points. This baseline dead-zone was decreased by 33 percent and 50 percent for a dead-zone size of 20 and 15 points, respectively. The baseline dead-zone was also increased by 33 percent and 100 percent for dead-zone size of 40 and 60 points, respectively. The results of the test are presented in Table 4-IV.

As before, the maximum squared difference and absolute difference decrease as the dead-zone decreases. The extension techniques react faster to any changes in the sensor signal so the maximum differences are lower with the smaller dead-zone. The sum of squared difference and sum of absolute difference also decrease as the dead-zone size decreases. The extension techniques perform better when the dead-zone estimate is smaller than actual. Results of this test and the earlier end-distortion test suggest that the magnitude of the high frequency noise in a sensor signal should also be used to set the size of the dead-zone.

All three proposed extension techniques perform better on the temperature sensor than the flow meter sensor. The First Point Past Dead-Zone extension technique performs better than the other two extension techniques for all the different dead-zone sizes. The Weighted Fuzzy and the Adaptive NET2 both perform about the same.

Table 4-IV: Evaluation of extension technique for robustness on temperature meter.

Temperature Sensor				
Dead Zone = 60				
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.0168	0.420	1.02	151
First Point Past DZ	0.0190	0.343	0.74	138
Weighted Fuzzy	0.0120	0.358	0.95	161

Dead Zone = 40				
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.0149	0.385	0.72	117
First Point Past DZ	0.0087	0.300	0.51	106
Weighted Fuzzy	0.0097	0.329	0.61	121

Dead Zone = 30				
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.0138	0.370	0.53	99
First Point Past DZ	0.0070	0.270	0.40	92
Weighted Fuzzy	0.0075	0.280	0.45	99

Dead Zone = 20				
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.0090	0.300	0.32	79
First Point Past DZ	0.0044	0.240	0.24	71
Weighted Fuzzy	0.0051	0.240	0.28	75

Dead Zone = 15				
	Max Squared Difference	Max Absolute Difference	Sum Squared Difference	Sum Absolute Difference
Adaptive NET2	0.0046	0.210	0.24	72
First Point Past DZ	0.0031	0.180	0.18	65
Weighted Fuzzy	0.0037	0.207	0.20	66

The combines results of the two robustness test indicate that the First Point Past Dead-Zone extension technique performs slightly better than the Weighted Fuzzy and much better than the Adaptive NET2 extension techniques. When using different dead-zone sizes, the First Point Past Dead-Zone has the less end-distortion than the other two

proposed extension techniques. All of the extension techniques perform better when the size of the dead-zone is underestimated rather than overestimated. So if there is any problem in estimating the period of the dominant frequency, it is better to err in a manner which makes the dead-zone smaller.

4.4 Sampling Rate Evaluation

The purpose of this test is to determine if the extension techniques are independent of the signal sampling rate. The tie-points used by the First Point Past Dead-Zone extension technique and the Weighted Fuzzy extension technique were compared using a sampling rate of one minute and a second sampling rate of ten seconds. The tie-point was recorded and then compared with the tie-point calculated at the other sampling rate. The comparisons were made using the same flow sensor and temperature sensor data as before.

The test procedure is given below.

1. Select the window of sensor data to be used for evaluation purposes. A window size of 768 sensor values was used for the one minute sampling rate which is extended to 1024 points for smoothing. A window size of 4608 sensor values was used for the ten second sampling rate that is extended to 8192 points for smoothing.

2. Calculate the tie-point for both sampling rates.

3. Compare the tie-point as calculated at the different sampling rates using the absolute difference as calculated from the following equation:

$$d = |z_1 - z_2| \quad (4.3).$$

The d term is the absolute difference between the tie-point as calculated at ten second sampling rate (z_1) and the tie-point as calculated from the one minute sampling rate (z_2).

4. Compare the tie-point as calculated at the different sampling rate for each extension technique using the squared difference as calculated from the following equation:

$$s = (z_1 - z_2)^2 \quad (4.4).$$

The s term is the squared difference between the tie-point as calculated at ten second sampling rate (z_1) and the tie-point as calculated from the one minute sampling rate (z_2).

The window of interest moves so all 6845 cases are considered. The results are given below.

Table 4-V: Sampling rate comparison of extension techniques.

	Flow Meter Sensor		Temperature Sensor	
	Sum Squared Difference	Sum Absolute Difference	Sum Squared Difference	Sum Absolute Difference
First Point Past DZ	0.0107	5.788	0.00021	0.520
Weighted Fuzzy	0.0105	5.766	0.00022	0.707

The sum absolute difference for the flow meter is 5.8 for both extension techniques over the 6845 cases. The temperature sensor shows an even smaller sum of absolute difference and sum of squared difference between the tie-points. This improved performance is due to the difference in the characteristics of the flow and temperature signals. However, the conclusion of this test is that the First Point Past Dead-Zone extension technique and the Weighted Fuzzy extension technique work regardless of the sampling rate. Neither extension techniques is affected by the sampling rate of the sensor.

4.5 Evaluation of Composite Results

Based on the results presented, the First Point Past Dead-Zone extension technique is the preferred choice. The First Point Past Dead-Zone extension technique generally outperformed the other methods on all three tests. Based on the ability to minimize end-distortion, the First Point Past Dead-Zone extension technique has the lowest maximum squared difference, the lowest absolute difference, and the sum of the absolute difference for both sensors of the proposed extension techniques. This technique

ranks best as the sum of the squared difference for both sensors. For the robustness evaluation, this technique has the lowest sum of squared difference and sum of absolute difference for six out of the eight comparison cases. For the other two cases, it ranks as second best. The sampling test verified the technique is unaffected by sampling rate as desired.

Two other criteria not listed tend to favor the First Point Past Dead-Zone extension technique. The First Point Past Dead-Zone extension technique has less computing requirements. This technique requires minimal effort to calculate the tie-point. The runner-up, the Weighted Fuzzy tie-point extension technique, is computationally intensive. The other criteria is based on the ease of implementation. The First Point Past Dead-Zone extension technique is much simpler to implement than the Weighted Fuzzy tie-point extension technique. Furthermore, Weighted Fuzzy tie-point extension technique has membership functions and the defuzzification algorithm which must be maintained. The simplicity of the First Point Past Dead-Zone extension technique contrasts sharply with the complexity of the Weighted Fuzzy tie-point extension technique.

The following comparison shows the improvement from the Raghavan NET2 to the First Point Past Dead-Zone extension technique. A periodic sensor signal with a period of 20 points is used to simulate a steady process measurement which has been contaminated by noise. The signal stays constant around a value of 0.5. The extracted trend should stay flat along the signal value of 0.5.

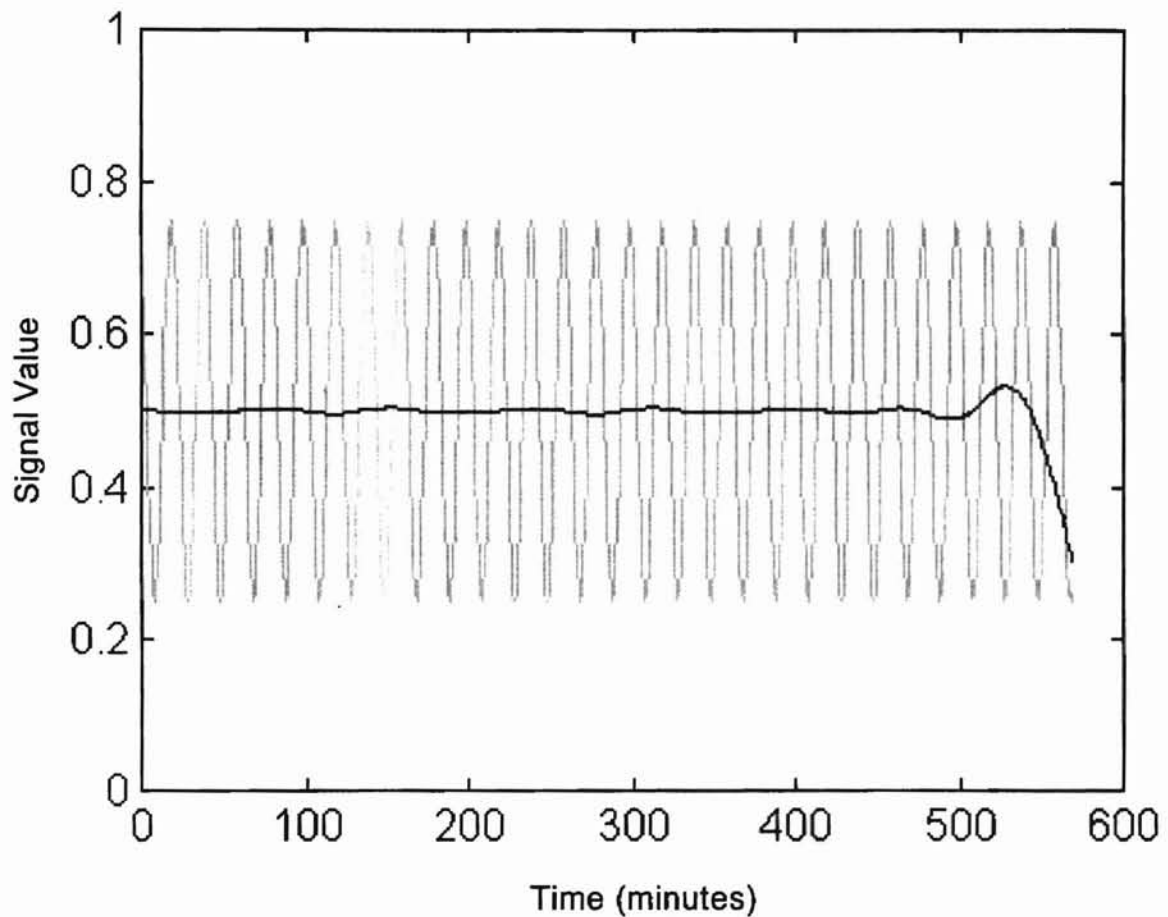


Figure 4-11: Example of Raghavan NET2 extension technique.

Use of the Raghavan NET2 technique results in distortion at the end of the signal. The smoothed trend should lie flat along the middle of the graph. This distortion is due to the small dead-zone of 3 points as hard-coded in the Raghavan NET2 technique.

The next figure is the same signal with the First Point extension technique.

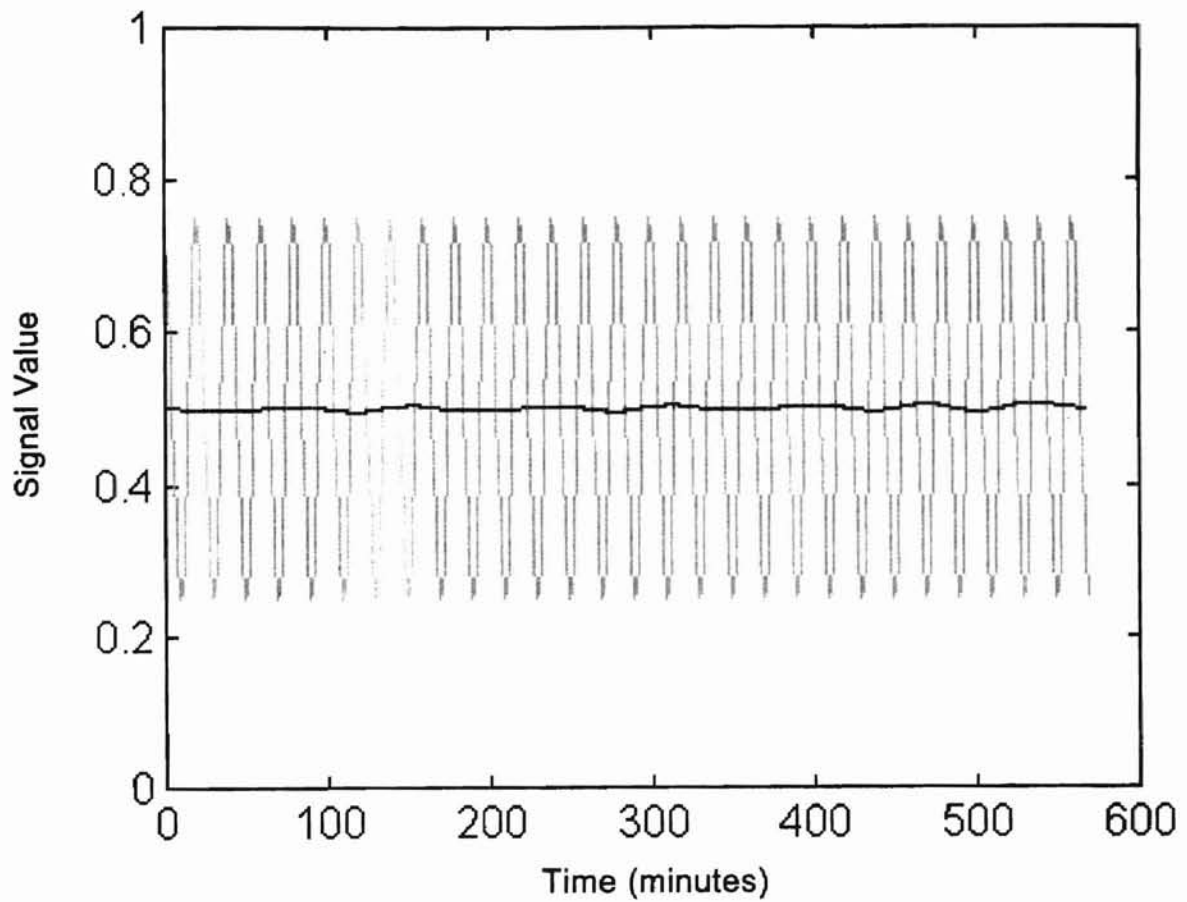


Figure 4-12: Example of First Point Past Dead-Zone extension technique.

This extension technique smoothes the trend with no distortion at the end of the signal. The smoothed trend stays relatively constant around the signal value of 0.5. The success of the First Point Past Dead-Zone extension technique is attributable to the adaptive nature of the technique which is driven by the characteristics of the signal being smoothed.

CHAPTER FIVE: CONCLUSIONS

This chapter summarizes the work presented in this thesis. After briefly reviewing the motivation for this research, a final analysis of the five proposed extension techniques is presented. Use of the First Point Past Dead-Zone extension technique is recommended as the preferred approach. Finally, recommendations for future work are presented. There are several areas that could benefit from more research.

5.1 Conclusions

The motivation for this research stems from the desire to apply wavelet smoothing for trend extraction. However, implementation of wavelet smoothing using traditional convolution filters causes end-distortion of the extracted trends. Five new extension techniques were proposed as a way to minimize the end-distortion. Each of these extension techniques use the dead-zone as calculated from the dominant frequency in the signal to be smoothed. The adaptive nature of the dead-zone calculation significantly improves the fidelity of extracted trends from sensor signals with differing frequencies and amplitudes of noise. Definition and use of the dead-zone was one of the contributions made by this research.

There are three main conclusions resulting from the research presented in this thesis. First, all five of the proposed extension techniques minimize the end-distortion caused by the wavelet smoothing. Second, the First Point Past Dead-Zone is the best of the five proposed extension techniques. It is the best technique at minimizing the end-distortion. This technique is the most simple to implement while requiring the least amount of computational effort. It is unaffected by the sampling rate. Yet, this technique is robust enough to provide trend extraction even with inaccurate estimates of the dead-zone. The final conclusion is that the First Point Past Dead-Zone extension technique is easily adaptable and can be applied with minimal effort.

5.2 Recommendations for Future Work

The following list summarizes suggested areas of future research in wavelet smoothing and signal extension:

1. Evaluate the performance of the five proposed extension techniques when using different wavelet families or with varying degrees of smoothing.
2. Develop code to combine the natural frequency calculation for the dead zone along with the extension and smoothing algorithms to provide a complete wavelet smoothing program for trend extraction and other possible monitoring and control applications.

3. Simplify the fuzzy algorithm for improved usability. This could allow a user to tune the rule-base and membership functions as to maximize performance for specific wavelet smoothing applications where the First Point Past Dead-Zone technique fails to provide the desired level of performance.

LIST OF REFERENCES

- Anderson, J. J. A Pattern-Based Approach to Gain Scheduling, School of Chemical Engineering. Oklahoma State University: Stillwater, Oklahoma (1993).
- Bellanger, M. Digital Processing of Signals Theory and Practice. Second Edition, John Wiley and Sons, New York (1990).
- Braae, M. and D. A. Rutherford. "Fuzzy Relations in a Control Setting." *Kybernetes* **7** (3), 185-188 (1978).
- Bracewell, R. M. The Fourier Transform and Its Applications. McGraw-Hill Book Company, New York (1965).
- Bueche, F. and D. Wallach. Technical Physics. Fourth Edition, John Wiley and Sons, Inc., New York (1994).
- Chui, C. K. An Introduction to Wavelets. Academic Press, Inc., Boston (1992).
- Chui, C. K., Ed. Wavelets: A Tutorial in Theory and Applications. Academic Press, Inc., Boston (1992a).
- Daubechies, I. "The Wavelet Transforms, Time-Frequency Localization and Signal Analysis." *IEEE Transactions On Information Theory* **36** (5), 961-1005 (1990).
- Daubechies, I. Ten Lectures on Wavelets. Society For Industrial and Applied Mathematics, Philadelphia (1992).
- Elliot, D. F., Ed. Handbook of Digital Signal Processing Engineering Applications. Academic Press, San Diego (1987).
- Gray, R. M. and J. W. Goodman. Fourier Transforms An Introduction for Engineers. Kluwer Academic Publishers, Boston (1995).
- Hellendorn, H. and C. Thomas. "Defuzzification in Fuzzy Controllers." *Journal of Intelligent and Fuzzy Systems* **1** 109-123 (1993).

- Jong, M. T. Methods of Discrete Signal and System Analysis. McGraw-Hill Book Company, New York (1982).
- King, R., M. Ahmadi, et al. Digital Filtering in One and Two Dimensions Design and Applications. Plenum Press, New York (1989).
- Korner, T. W. Fourier Analysis. Cambridge University Press, Cambridge, Great Britain (1988).
- Kreyszig, E. Advanced Engineering Mathematics. Fifth Edition, John Wiley and Sons, New York (1983).
- Lee, C. C. "Fuzzy Logic in Control Systems: Fuzzy Logic Controller - Part I." *IEEE Transactions on Systems, Man, and Cybernetics* **20** (2), 404-418 (1990).
- Lee, C. C. "Fuzzy Logic in Control Systems: Fuzzy Logic Controller, Part II." *IEEE Transactions on Systems, Man, and Cybernetics* **20** (2), 419-435 (1990a).
- Mallat, S. G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11** (7), 674-693 (1989).
- Mallat, S. G. "Multifrequency Channel Decomposition of Images and Wavelet Models." *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37** (12), 2091-2109 (1989a).
- Mallat, S. G. "Multiresolution Approximations and Wavelet Orthonormal Bases of $L_2(\mathbb{R})$." *Transactions of the American Mathematical Society* **315** (1), 69-87 (1989b).
- Mamdani, E. H. "Application of Fuzzy Algorithms for Control of Simple Dynamic Plant." *Proceedings of the IEEE* **121** (12), 1585-1588 (1974).
- Mamdani, E. H. and S. Assilian. "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller." *International Journal of Man-Machine Studies* **7** 1-13 (1975).
- Mamdani, E. H. "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis." *IEEE Transactions on Computers* **C-26** (12), 1182-1191 (1977).
- Mendel, J. M. "Fuzzy Logic Systems for Engineering: A Tutorial." *Proceedings of the IEEE* **83** (3), 345-377 (1995).
- Misawa, E. A. Fuzzy Systems Theory and Applications. Lecture Notes for MAE 5733 Spring Semester Course at Oklahoma State University: Stillwater, Oklahoma (1994).
- Morrison, N. Introduction to Fourier Analysis. John Wiley and Sons, New York (1994).

Press, W. H., S. A. Teukolsky, et al., Eds. Numerical Recipes in C The Art of Scientific Computing. Second Edition, Press Syndicate of the University of Cambridge, New York (1992).

Raghavan, V. K. Wavelet Representation of Sensor Signals for Monitoring and Control, School of Chemical Engineering. Oklahoma State University: Stillwater, Oklahoma (1995).

Raghavan, V. K. and J. R. Whiteley. Wavelet Representation of Sensor Patterns for Monitoring and Control. *American Institute of Chemical Engineers 1993 Annual Meeting*, St. Louis, Missouri, November 7-12 (1993).

Rioul, O. and P. Duhamel. "Fast Algorithms for Discrete and Continuous Wavelet Transforms." *IEEE Transactions on Information Theory* **38** (2), 569-585 (1992).

Sinha, M. Pattern-Based Process Characterization and Gain-Scheduling for Nonlinear Chemical Processes, School of Chemical Engineering. Oklahoma State University: Stillwater, Oklahoma (1995).

Smith, W. W. and J. M. Smith. Handbook of Real-Time Fast Fourier Transforms. IEEE Press, New York (1995).

Sugeno, M. and K. Murakami. "An Experimental Study on Fuzzy Parking Control Using a Model Car." Industrial Applications of Fuzzy Control. M. Sugeno, Ed. Amsterdam, The Netherlands, Elsevier Science Publishing Company, Inc., 125-138 (1985).

Tolstov, G. P. Fourier Series. translated by R. A. Silverman, Dover Publications, Inc., New York (1962).

Vetterli, M. and C. Herley. "Wavelets and Filter Banks: Theory and Design." *IEEE Transactions on Signal Processing* **40** (9), 2207-2232 (1992).

Walter, G. G. "Discrete Discrete Wavelets." *SIAM Journal of Mathematical Analysis* **23** (4), 1004-1014 (1992).

Weaver, H. K. Applications of Discrete and Continuous Fourier Analysis. John Wiley and Sons, New York (1983).

Whiteley, J. R. and J. F. Davis. "Qualitative Interpretation of Sensor Patterns." *IEEE Expert* **8** 54-63 (1993).

Whiteley, J. R. and J. F. Davis. "A Similarity-Based Approach to Interpretation of Sensor Data Using Adaptive Resonance Theory." *Computers and Chemical Engineering* **18** (7), 637-661 (1994).

Zadeh, L. A. "Fuzzy Sets." *Information and Control* 8 338-353 (1965).

Zimmermann, H. J. Fuzzy Set Theory and Its Applications. Second Edition, Kluwer Academic Publishers, Boston (1991).

APPENDIX A

A.1 Fuzzy Logic

Fuzzy logic was developed by L. Zadeh [Zadeh, 1965] as a way to quantify vagueness and uncertainty. Fuzzy logic is a branch of mathematics known as fuzzy set theory. Fuzzy set theory has developed from the more basic crisp set theory [Zadeh, 1965]. In crisp set theory, an element either belongs to a set or it is excluded from a set. For example, the set of positive integer numbers less than five would be defined as {1, 2, 3, 4}. In fuzzy set theory, an element is assigned to a set along with the degree of membership to the set. This degree of membership quantifies the extent to which the element belongs with the set. The degree of membership is defined by the Greek letter μ . The degree of membership is generally defined from 0 to 1. The definition of a set in fuzzy set theory is as follows [Zimmermann, 1991]:

$$A = \{(x, \mu_A(x)) | x \in X\} \quad (\text{A.1}).$$

The $\mu_A(x)$ term is the degree of membership that the element x (which is defined over the entire set of X) has in the set A . For example a crisp set of height of tall people could include anyone with a height above 6 feet. This crisp set could be defined as the following:

$$B = \{x | x \geq 6\} \quad (\text{A.2}).$$

x is defined in feet.

The crisp set B represents all the tall people over the height of six feet. For a similar example in fuzzy set theory, the height of tall people could be defined as the following set:

$$\tilde{B} = \{(6, .2), (6.25, .4), (6.5, .6), (6.75, .8), (7, 1)\} \quad (\text{A.3}).$$

This fuzzy set \tilde{B} represents the set of tall people defined over the elements of 6, 6.25, 6.5, 6.75, and 7 with the degree of membership associated with each element. The elements which occur between elements listed are linearly interpolated while any elements outside the range are taken as degree of membership of zero. So based on the fuzzy set, a person who is 6.3 feet would be assigned a $\mu = 0.44$. This fuzzy set can be represented by the following as a mathematical equation:

$$\tilde{B} = \{(x, \mu_{\tilde{B}}(x)) | x \in X\} \quad (\text{A.4}).$$

where

$$\mu_{\tilde{B}}(x) = \begin{cases} 0, & x \leq 5.75 \\ 0.8x - 4.6, & 5.75 < x \leq 7 \\ 1, & x > 7 \end{cases} \quad (\text{A.5}).$$

The set \tilde{B} is defined over the range of elements x as defined by equation A.5. This fuzzy set can also be represented using graphical notation as the following [Zimmermann, 1991]:

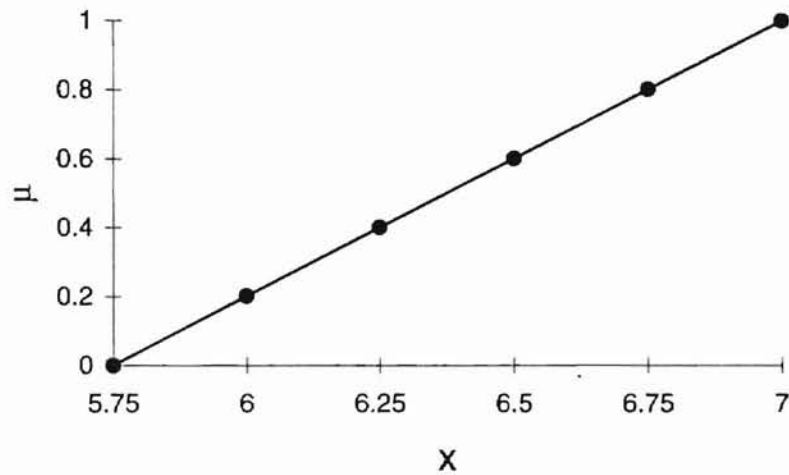


Figure A-1: Graphical representation of fuzzy set of tall people.

Each of the three representations illustrate the combined use of an element and the degree of membership to represent fuzzy sets. This example of tall people is arbitrarily based on the criteria established using human intuition. The example can be expanded to include a fuzzy set of short people, medium people, extra-tall people, and so-on. Each defined fuzzy set contains elements which may belong to other sets.

A.2 Approximate Reasoning

Approximate reasoning provides the mathematical framework to use fuzzy logic. The use of approximate reasoning allows the introduction of rule-bases to facilitate the application of fuzzy logic. The rules provide the systematic technique to quantify the fuzzy logic as applied to fuzzy statements containing the linguistic variables. The general form is as follows [Lee, 1990; Lee, 1990a; Misawa, 1993; Mendel, 1995]:

Rule 1:	If x is A	then y is B
Suppose:	<u>x is A'</u>	
Result:		y is B'

Figure A-2: Fuzzy logic implication with one condition.

The terms A, A', B, and B' are the linguistic variables for the fuzzy statements. These linguistic variables are typically fuzzy sets which are previously defined. Approximate reasoning provides the structure to mathematically solve the above situation. The rule defines the value for y implied from the value of x. The following example demonstrates the application of the rules with implication using linguistic variables.

Rule 1: If Apple is Red then Apple is Ripe
Suppose: Apple is Very Red
Result: Apple is Very Ripe

Figure A-3: Example of fuzzy logic implication with one condition.

The premise (Rule 1) states that if the apple is red then the apple is ripe. Suppose an apple to be very red, then the implied inference result is that the apple is very ripe.

The implication used in this research is Mamdani's rule of implication [Mamdani 1975; Mamdani and Assilian, 1975; Mamdani, 1977]. The general form of one condition rule-base is defined as following:

Rule 1: If x is A then y is C
Suppose: x is A'
Result: y is C'

Figure A-4: Fuzzy logic implication with one condition.

Then Mamdani's rule of implication would be defined as following:

$$C' = A \circ (A \rightarrow C) \quad \Leftrightarrow \quad \mu_C(y) = \bigvee_x \mu_{A'}(x) \wedge \mu_A(x) \wedge \mu_C(y) \quad (\text{A.6}).$$

The $\mu_A(x)$ is the premise input (Rule 1), $\mu_C(y)$ is the premise (Rule 1) result, and $\mu_{A'}(x)$ is the actual input variable. The implication is the maximum-minimum operator (abbreviated max-min). This means that $\mu_C(y)$ is the maximum of the intersection of $\mu_A(x)$, $\mu_{A'}(x)$, and $\mu_C(y)$. The more general approach used in this research is the two conditions rule-base such as the following:

Rule 1:	If x is A	and	y is B	then z is C
Suppose:	x is A'	and	y is B'	
Result:				z is C'

Figure A-5: Fuzzy logic two conditions implication rule-base.

Equation A.6 becomes the following with the addition of the two conditions rule-base:

$$C' = (A', B') \circ [(A, B) \rightarrow C] \Leftrightarrow [A' \circ (A \rightarrow C)] \wedge [B' \circ (B \rightarrow C)] \quad (A.7).$$

When using multiple rules then the general form for Mamdani's implication is as follows:

Rule 1:	If x is A ₁	and	y is B ₁	then z is C ₁
Rule 2:	If x is A ₂	and	y is B ₂	then z is C ₂
Rule 3:	If x is A ₃	and	y is B ₃	then z is C ₃
Suppose:	x is A'	and	y is B'	
Result:				z is C'

Figure A-6: General form of Mamdani's implication rule for two conditions.

The output for z is calculated using the union of each rule with the other rules. So the general form for two conditions using Mamdani's implication for N rules:

$$C' = (A', B') \circ \{[(A_1, B_1) \rightarrow C_1] \cup [(A_2, B_2) \rightarrow C_2] \cup \dots \cup [(A_N, B_N) \rightarrow C_N]\} \quad (A.8).$$

The output (C') set from the implication rule-base is a fuzzy membership function. This fuzzy membership function is defined over the range of z values for each rule that is fired. Once the output membership function, z , is calculated the resultant is usually converted back to a crisp number. The process to convert a fuzzy set into a crisp set is known as defuzzification.

A.3 Defuzzification

For this application, the inferred output fuzzy function is converted back to a crisp numerical value. The defuzzification technique used is the center of area also known as the center of mass [Brae and Rutherford, 1978; Lee, 1990a; Hellendorn and Thomas, 1993; Mendel, 1995]. The crisp numerical output from the output fuzzy function is calculated as the center of gravity for the total output fuzzy function. The fuzzy membership function is discretized over the range of elements (x). The crisp numerical output is calculated based on the following equation [Lee, 1990a]:

$$z = \frac{\sum_{j=1}^n \mu_z(x_j) \cdot x_j}{\sum_{j=1}^n \mu_z(x_j)} \quad (\text{A.9}).$$

The x_j term is the element which is discretized over j points ($j = 1$ to n). The μ_z term is the membership value for each element (x_j) over the whole range of j that x exists. The output crisp value is the z term. This crisp value is used in the evaluation of the output membership function. Based on several different criteria, the center of area is rated as one of the better defuzzification techniques [Lee, 1990a; Hellendorn and Thomas, 1993].

APPENDIX B

B.1 Computer Code for Adaptive NET2 Extension Technique

This is the computer code for the Adaptive NET2 extension technique. The windowed signal is the vector represented by $a[]$ with 768 points. The return vector $a[]$ with 1024 can be smoothed with the wavelet smoothing code.

```
for (k=0;k<200+1;k++){
    tmp[k]=a[n-k-1];
}

for (k= index_num_one_freq;k<3*index_num_one_freq;k++) { /* k=length of skip
window */
    for (kk=0;kk<k+1;kk++) { /* calculate the mean */
        mean1 +=tmp[kk];
    }
    mean1=mean1/(k+1);

    for (kk=0;kk<k+1;kk++) { /* calculate the sum_sq_error deviation */
        sum_sq_error += (mean1-tmp[kk])*(mean1-tmp[kk]);
    }

    sum_sq_error=sum_sq_error/(k+1);

    if (sum_sq_error <=min_sum_sq_error)
    {
        min_mean=mean1;
        min_sum_sq_error=sum_sq_error;
        flip_index=k+1;
    }

sum_sq_error=0.0;
mean1=0.0;

} /* end of iteration k */

for (k=0;k<n/3;k++) a[n+k]=2*min_mean-a[n-k-1];
```

Return vector $a[]$ with 1024 points - this is the extended signal.

B.2 Computer Code for Square Root Objective Function Extension Technique

This is the computer code for the Square Root Objective Function extension technique. The windowed signal is the vector represented by a[] with 768 points. The return vector a[] with 1024 can be smoothed with the wavelet smoothing code.

```
    for (k=0;k<200+1;k++){
        tmp[k]=a[n-k-1];
    }

for (k= index_num_one_freq;k<3*index_num_one_freq;k++) { /* k=length of skip
window */
    for (kk=0;kk<k+1;kk++) { /* calculate the mean */
        mean1 +=tmp[kk];
    }
    mean1=mean1/(k+1);

    for (kk=0;kk<k+1;kk++) { /* calculate the sum_sq_error deviation */
        sum_sq_error += (mean1-tmp[kk])*(mean1-tmp[kk]);
    }
    root=sqrt(k+1); /* root is sqrt of n */
    sum_sq_error=sum_sq_error/root;

    if (sum_sq_error <=min_sum_sq_error)
    {
        min_mean=mean1;
        min_sum_sq_error=sum_sq_error;
        flip_index=k+1;
    }

sum_sq_error=0.0;
mean1=0.0;

} /* end of iteration k */

    for (k=0;k<n/3;k++) a[n+k]=2*min_mean-a[n-k-1];
```

Return vector a[] with 1024 points - this is the extended signal.

B.3 Computer Code for First Point Past Dead-Zone Extension Technique

This is the computer code for the First Point Past Dead-Zone extension technique. The windowed signal is the vector represented by a[] with 768 points. The return vector a[] with 1024 can be smoothed with the wavelet smoothing code.

```
for (k=0;k<200+1;k++){
    tmp[k]=a[n-k-1];
}

for (kk=0;kk<index_num_one_freq+2;kk++) { /* calculate the mean */
    mean1 +=tmp[kk];
}
min_mean=mean1/(k+1);

for (k=0;k<n/3;k++) a[n+k]=2*min_mean-a[n-k-1];
```

Return vector a[] with 1024 points - this is the extended signal.

B.4 Computer Code for Fuzzy Tie-Point Extension Technique

This is the computer code for the Fuzzy Tie-Point extension technique. The windowed signal is the vector represented by a[] with 768 points. The return vector a[] with 1024 can be smoothed with the wavelet smoothing code.

```
for (k=0;k<200+1;k++){
    tmp[k]=a[n-k];
}

/* this calculates the sum of error at one freq */
for(i=0;i<index_num_one_freq; i++){
    mean_one_freq += tmp[i];
}

mean_one_freq=mean_one_freq/index_num_one_freq;

for(i=0;i<index_num_one_freq; i++){
    sum_error_one_freq += (mean_one_freq-tmp[i])*(mean_one_freq-tmp[i]);
}
}
```

```

sum_error_one_freq=sum_error_one_freq/index_num_one_freq;

for (k=index_num_one_freq;k<3*index_num_one_freq;k++) {
/* k=length of skip window to how far back*/

for (kk=0;kk<k+1;kk++) { /* calculate the mean */
    mean1 +=tmp[kk];
    }
    mean1=mean1/(k+1); /*remember index_num=k+1 */

for (kk=0;kk<k+1;kk++) { /* calculate the sum_error deviation */
    sum_error += (mean1-tmp[kk])*(mean1-tmp[kk]);
    }

sum_error = sum_error/(k+1); /* this is sum_sq_dev/n */

index_num=k+1; /* remember k is one less than index */

rating=fuzzlets(sum_error, index_num, sum_error_one_freq, index_num_one_freq);

    if (rating >=max_rating){
        flip_mean=mean1;
        max_rating=rating;
        index_max_rate=k+1;
    }

sum_error=0.0;
mean1=0.0;

} /* end of iteration k */

for (k=0;k<n/3;k++) a[n+k]=2*flip_mean-a[n-k-1];

} /*end of net2 */

/*****
/*$$$$$$          fuzzlets.c          $$$$$$$$*/
/*****

/* written 1/10/96 by Jonathan Hynson all rights reserved under US copyright laws */

```

```

/* program using Fuzzy Logic Reasoning to approximate the best choice for the */
/* flip point (average value) of wavelet extension technique developed by Vinod */
/* Raghavan*/
/* singlet inputs with two fuzzy variables: sum_error (sum of squared deviation) */
/*      & index_num(index number back from end of sensor data) higher number */
/* further back from end */
/* Fuzzy Parameters: singlet inputs (two input variables with three func for each), */
/*      one output variable (rating of flip pt with 4 func), Rules are listed below, */
/*      Mamdani's implication rule (inference: max-min), COA (Center of Mass or */
/* Centroid) defuzzification */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double fuzzlets(double sum_error, double index_num, double sum_error_one_freq,
double index_num_one_freq)
{
/* These are the inputs: index_num & sum_error (actually sum_sq_deviation) */
/* sum_error (really sum_sq_deviation/n) & index_num increases as go back away from */
/* end of data */
/* double sum_error, index_num; */
/* sum_error_one_freq is the sum deviation sq at one period of freq/n*/
/* index_num_one_freq is the number of pts (index) for one period of freq */
/* double index_num_one_freq, sum_error_one_freq; */

/* membership func: func return a mu for a given x value (index_num_* or sum_error_* ) */
/*
double mu_sum_error_sm_fun(double,double), mu_sum_error_med_fun(double,double),
mu_sum_error_lg_fun(double,double);
double mu_index_num_vsmall_fun(double,double),
mu_index_num_small_fun(double,double), mu_index_num_med_fun(double,double);
double mu_index_num_high_fun(double,double),
mu_index_num_vhigh_fun(double,double);

/* max-min inference func for Mamdani's rule of implication;*/
double mu_flip_pt_fun(double, double, double);

/* the mu values for each fun */
double mu_sum_error_sm,mu_sum_error_med, mu_sum_error_lg;
double mu_index_num_vsmall, mu_index_num_small, mu_index_num_med;
double mu_index_num_high, mu_index_num_vhigh;
double mu_flip_pt_bad, mu_flip_pt_ok, mu_flip_pt_good;
double mu_flip_pt_exc, mu_flip_pt_vexc;

```

```

double flip_pt_x=0.0; /* ranges 0 to 40, @ 0.1 increment; see flip_pt_rating */

/* membership func: func return a mu for a given x value (flip_pt_x) */
double mu_flip_pt_bad_fun(double), mu_flip_pt_ok_fun(double),
mu_flip_pt_good_fun(double);
double mu_flip_pt_exc_fun(double), mu_flip_pt_vexc_fun(double);

/* these are the membership values of the function; used to calculate mu of the func */
/* at the value of a given x (flip_pt_x) from the above func */
double mu_flip_pt_bad_slope, mu_flip_pt_ok_slope, mu_flip_pt_good_slope;
double mu_flip_pt_exc_slope, mu_flip_pt_vexc_slope;

/* func returns the mu_value_at_flip_pt_x given the following (in this order) */
/* mu_flip_pt_*1, mu_flip_pt_*1_slope, mu_flip_pt_*2, mu_flip_pt_*2_slope */
double centroid_defuzz_fun(double, double, double, double);

/* these sum for centroid defuzzification method */
double mu_value_at_flip_pt_x=0.0, flip_pt_rating=0.0;
double sum_mu_times_x=0.0, sum_mu=0.0;
/* maximize the flip_pt_rating, higher rating--> better choice for flip point */

/***** !!!!!!!!!NOTE: index_num>=dead_zone !!!!!!!!!*****/
/* Check if index_num<index_num_one_freq ==> return flip_pt_rating=0 & exit */
if (index_num<index_num_one_freq){
    printf(" ERROR IN CODE!!! Index Number < Dead Zone (Number of points of
one period of freq) \n");
    flip_pt_rating=0;
    return(flip_pt_rating);
}

/* Call the membership functions */

/* sum_error_one_freq & index_num_one_freq are set by the values of each */
/* variable at one complete period of the natural frequency of signal (sensor noise) */

mu_sum_error_sm = mu_sum_error_sm_fun(sum_error, sum_error_one_freq);
mu_sum_error_med = mu_sum_error_med_fun(sum_error, sum_error_one_freq);
mu_sum_error_lg = mu_sum_error_lg_fun(sum_error, sum_error_one_freq);

/*printf(" mu sum error sm = %lf\n", mu_sum_error_sm);
printf(" mu sum error med = %lf \n", mu_sum_error_med);
printf(" mu sum error lg = %lf \n", mu_sum_error_lg);
printf(" \n"); */

```

```

mu_index_num_vsmall = mu_index_num_vsmall_fun(index_num,
index_num_one_freq);
mu_index_num_small = mu_index_num_small_fun(index_num, index_num_one_freq);
mu_index_num_med = mu_index_num_med_fun(index_num, index_num_one_freq);
mu_index_num_high = mu_index_num_high_fun(index_num, index_num_one_freq);
mu_index_num_vhigh = mu_index_num_vhigh_fun(index_num, index_num_one_freq);

```

```

/*printf("  mu index num vsmall = %lf \n", mu_index_num_vsmall);
printf("  mu index num small = %lf \n", mu_index_num_small);
printf("  mu index num med = %lf \n", mu_index_num_med);
printf("  mu index num high = %lf \n", mu_index_num_high);
printf("  mu index num vhigh = %lf \n", mu_index_num_vhigh);
printf(" \n"); */

```

```

/* initialize all mu_flip_pt_* to 0 */
mu_flip_pt_bad=0;
mu_flip_pt_ok=0;
mu_flip_pt_good=0;
mu_flip_pt_exc=0;
mu_flip_pt_vexc=0;

```

```

/* Each rule calls the rule base function */
/* max-min inference func: returns max(min(mu_sum_error_*,
mu_index_num_*),mu_flip_pt_*) for each rule in rule base */
/* part_1=min(mu_sum_error_*, mu_index_num_*) for Mamdani's rule of implication;
max-min inference; */
/* the max( part_1, mu_flip_pt_*) for the else (union) to pick largest mu_flip_pt_* for
each */

```

```

/* RULE ONE */
if (mu_sum_error_sm>0 && mu_index_num_vsmall>0) {
    mu_flip_pt_vexc=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_vsmall,mu
_flip_pt_vexc);
}

```

```

/* RULE TWO */
if (mu_sum_error_sm>0 && mu_index_num_small>0){
    mu_flip_pt_good=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_small,mu_
flip_pt_good);
}

```

```

/* RULE THREE */
if (mu_sum_error_sm>0 && mu_index_num_med>0){

```

```

mu_flip_pt_exc=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_med,mu_flip_pt_ex
c);
}

/* RULE FOUR */
if (mu_sum_error_sm>0 && mu_index_num_high>0){

mu_flip_pt_good=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_high,mu_flip_pt_
good);
}

/* RULE FIVE */
if (mu_sum_error_sm>0 && mu_index_num_vhigh>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_vhigh,mu_flip_pt_o
k);
}

/* RULE SIX */
if (mu_sum_error_med>0 && mu_index_num_vsmall>0){

mu_flip_pt_good=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_vsmall,mu_flip_
pt_good);
}

/* RULE SEVEN */
if (mu_sum_error_med>0 && mu_index_num_small>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_small,mu_flip_pt_
ok);
}

/* RULE EIGHT */
if (mu_sum_error_med>0 && mu_index_num_med>0){

mu_flip_pt_good=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_med,mu_flip_pt
_good);
}

/* RULE NINE */
if (mu_sum_error_med>0 && mu_index_num_high>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_high,mu_flip_pt_o
k);
}

```

```

}

/* RULE TEN */
if (mu_sum_error_med>0 && mu_index_num_vhigh>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_vhigh,mu_flip_pt
_bad);
}

/* RULE ELEVEN */
if (mu_sum_error_lg>0 && mu_index_num_vsmall>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_vsmall,mu_flip_pt_o
k);
}

/* RULE TWELVE */
if (mu_sum_error_lg>0 && mu_index_num_small>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_small,mu_flip_pt_b
ad);
}

/* RULE THIRTEEN */
if (mu_sum_error_lg>0 && mu_index_num_med>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_med,mu_flip_pt_ok);
}

/* RULE FOURTEEN */
if (mu_sum_error_lg>0 && mu_index_num_high>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_high,mu_flip_pt_ba
d);
}

/* RULE FIFTEEN */
if (mu_sum_error_lg>0 && mu_index_num_vhigh>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_vhigh,mu_flip_pt_b
ad);
}

/*printf("  mu_flip_pt_bad = %lf\n", mu_flip_pt_bad);

```

```

printf("  mu_flip_pt_ok = %lf \n", mu_flip_pt_ok);
printf("  mu_flip_pt_good = %lf\n", mu_flip_pt_good);
printf("  mu_flip_pt_exc = %lf \n", mu_flip_pt_exc);
printf("  mu_flip_pt_vexc = %lf \n", mu_flip_pt_vexc);*/

/* print only for diagnostic checking */
/*printf("  mu_flip_pt_bad_slope = %lf \n", mu_flip_pt_bad_slope);
printf("  mu_flip_pt_ok_slope = %lf \n", mu_flip_pt_ok_slope);
printf("  mu_flip_pt_good_slope = %lf \n", mu_flip_pt_good_slope);
printf("  mu_flip_pt_exc_slope = %lf \n", mu_flip_pt_exc_slope);
printf("  mu_flip_pt_vexc_slope = %lf \n", mu_flip_pt_vexc_slope);*/

/* for loop for summing up centroid areas of output membership functions */
/* output member func is split into three segments: (bad,ok),(ok,good),(good,exc)*/
/* for each part of the for loop, the centroid_defuzz_fun is called to calculate */
/* the max{min(mu_flip_pt_*1_slope, mu_flip_pt_*1), min(mu_flip_pt_*2_slope,
mu_flip_pt_*2)} */
/* the resulting max (mu_value_at_flip_pt_x) is calculated into the centroid
defuzzification method */

flip_pt_x=0;
for (flip_pt_x=0; flip_pt_x<40.1;) {

    if (flip_pt_x<=10){ /* the first part of the ouput membership functions */
        mu_flip_pt_bad_slope = mu_flip_pt_bad_fun(flip_pt_x);
        mu_flip_pt_ok_slope = mu_flip_pt_ok_fun(flip_pt_x);
        mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_bad,
mu_flip_pt_bad_slope, mu_flip_pt_ok, mu_flip_pt_ok_slope);
        sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
        sum_mu = mu_value_at_flip_pt_x + sum_mu;
    }

    if (flip_pt_x>10 && flip_pt_x<20.1){ /* the second part of the ouput
membership functions */
        mu_flip_pt_ok_slope = mu_flip_pt_ok_fun(flip_pt_x);
        mu_flip_pt_good_slope = mu_flip_pt_good_fun(flip_pt_x);
        mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_ok,
mu_flip_pt_ok_slope, mu_flip_pt_good, mu_flip_pt_good_slope);
        sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
        sum_mu = mu_value_at_flip_pt_x + sum_mu;
    }
}

```



```

        if (flip_pt_x>20 && flip_pt_x<30.1){ /* the third part of the ouput membership
functions */
            mu_flip_pt_good_slope = mu_flip_pt_good_fun(flip_pt_x);
            mu_flip_pt_exc_slope = mu_flip_pt_exc_fun(flip_pt_x);
            mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_exc,
mu_flip_pt_exc_slope, mu_flip_pt_good, mu_flip_pt_good_slope);
            sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
            sum_mu = mu_value_at_flip_pt_x + sum_mu;
        }

        if (flip_pt_x>30 && flip_pt_x<40.1){ /* the fourth part of the ouput
membership functions */
            mu_flip_pt_exc_slope = mu_flip_pt_exc_fun(flip_pt_x);
            mu_flip_pt_vexc_slope = mu_flip_pt_vexc_fun(flip_pt_x);
            mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_vexc,
mu_flip_pt_vexc_slope, mu_flip_pt_exc, mu_flip_pt_exc_slope);
            sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
            sum_mu = mu_value_at_flip_pt_x + sum_mu;
        }

        flip_pt_x=flip_pt_x+0.1;

/* end of for loop summing all the centroid areas */
}

flip_pt_rating=sum_mu_times_x/sum_mu;

/*
printf("  sum_mu_times_x = %lf  AND  sum_mu = %lf \n", sum_mu_times_x,
sum_mu);
printf("  flip point rating = %lf  (from fuzzlets function) \n", flip_pt_rating);
*/

if (sum_mu_times_x<=0.1){ /* Error Check of sum_mu_times_x */
    printf("  ERROR IN CALCULATION: Deffuzification Error with Sum of (mu
multiplied by x) \n");
    flip_pt_rating=0;
    return(flip_pt_rating);
}

if (sum_mu<=0.001){ /* Error Check of sum_mu */

```

```

        printf(" ERROR IN CALCULATION: Deffuzification Error with Sum of mu
\n");
        flip_pt_rating=0;
        return(flip_pt_rating);
    }

```

```

return(flip_pt_rating);
/* end of fuzzlets.c function called to calculate rating of flip point */
}

```

```

/***** THE MEMBERSHIP FUNCTIONS
*****/

```

```

/*****the mu_sum_error membership functions *****/
/* For all the mu_sum_error membership functions are adjusted back to
sum_error_one_period */
/* which is the sum_error @ one period of freq. */

```

```

/***** mu_sum_error_sm_fun *****/ sum_error_small decreases linearly such
that */
/* trapazoid membership function without left leg */
/* the mu=1 (max value) is sum_error=(4/5)*sum_error_one_per, and mu=0 (min value)
is (9/5)*sum_error_one_period */

```

```

double mu_sum_error_sm_fun(double x, double sum_error_one_period)
{

```

```

    double slope=1/(sum_error_one_period);
    double mu, x_intercept_down=slope*(9*sum_error_one_period/5);
    /* x_intercept_down=the x-intercept of the membership function with x-axis */

```

```

    if(x<4*sum_error_one_period/5) mu = 1;
    else if(x>=4*sum_error_one_period/5 && x<=9*sum_error_one_period/5) mu = -
slope*x + x_intercept_down;
    else mu=0;

```

```

    return(mu);
    /* end of mu_sum_error_sm_fun */
}

```

```

/***** mu_sum_error_med_fun *****/ sum_error_med increase & decreases
linearly such that */
/* the mu=0 (min value-left) is (4/5)*sum_error_one_period */
/* and mu=1 (max value) is (9/5)*sum_error_one_period */

```

```

/* the mu=0 (min value-right) is (14/5)*sum_error_one_period */

double mu_sum_error_med_fun(double x, double sum_error_one_period)
{

double slope=1/(sum_error_one_period);
double mu, x_intercept_up=slope*(4*sum_error_one_period/5);
double x_intercept_down=slope*(14*sum_error_one_period/5);
/* x_intercept_up=the x-intercept of the increasing membership function with x-axis */
/* x_intercept_down=the x-intercept of the decreasing membership function with x-axis */

if (x>4*sum_error_one_period/5 && x<=9*sum_error_one_period/5) mu=slope*x-
x_intercept_up;
else if (x>9*sum_error_one_period/5 && x<=14*sum_error_one_period/5) mu=-
slope*x+x_intercept_down;
else mu=0;

return(mu);
/* end of mu_sum_error_med_fun */
}

/***** mu_sum_error_lg_fun *****/ sum_error_lg increases as 1-1/(x/x0) */
/* the mu=0 (min value) is (9/5)*sum_error_one_period;
x0=(9/5)*sum_error_one_period */
/* the function is 1-1/(x/x0) which increases quadratic to infinity */

double mu_sum_error_lg_fun(double x, double sum_error_one_period)
{
double mu;

if(x>9*sum_error_one_period/5) mu=1-1/(x/(9*sum_error_one_period/5));
else mu=0;

return(mu);
/* end of mu_sum_error_lg_fun */
}

/***** The mu_index_num_membership functions
*****/
/* start_index_num is the index at one period of freq and period=number of points for a
period */

```

```

/***** mu_index_num_vsmall_fun *****/ linearly decreasing membership
function */
/* mu=1 (max value) is start_index_num ; the mu=0 (min value-right) is
1.5*start_index_num */

double mu_index_num_vsmall_fun(double x, double start_index_num)
{
double mu, slope_of_line=1/(0.5*start_index_num); /* slope of line: slope of
membership func */
double x_intercept_down=slope_of_line*(1.5*start_index_num);
/* x-axis intercept for decreasing (right) side of trapazoid membership func */

if (x>=start_index_num && x<=1.5*start_index_num) mu = -
slope_of_line*x+x_intercept_down;
else mu=0;

return(mu);
/* end of double mu_index_num_vsmall_fun */
}

/***** mu_index_num_small_fun *****/ triangle membership function */
/* the mu=0 (min value-left) is start_index_num, and mu=1 (max value) is
1.5*start_index_num */
/* the mu=1 (min value-right) is 2*start_index_num */

double mu_index_num_small_fun(double x, double start_index_num)
{
double mu, slope_of_line=1/(0.5*start_index_num); /* slope of line: slope of
membership func */
double x_intercept_up=slope_of_line*(start_index_num);
double x_intercept_down=slope_of_line*(2*start_index_num);
/* x-axis intercept for increasing (left) and decreasing (right) side of triangle membership
func */

if (x>start_index_num && x<=1.5*start_index_num) mu = slope_of_line*x-
x_intercept_up;
else if (x>1.5*start_index_num && x<=2*start_index_num) mu = -
slope_of_line*x+x_intercept_down;
else mu=0;

return(mu);

/* end of mu_index_num_small_fun */
}

```

```

/***** mu_index_num_med_fun *****/ triangle membership function */
/* the mu=0 (min value-left) is 1.5*start_index_num, and mu=1 (max value) is
2*start_index_num */
/* the mu=1 (min value-right) is 2.5*start_index_num */

double mu_index_num_med_fun(double x, double start_index_num)
{
double mu, slope_of_line=1/(0.5*start_index_num); /* slope of line: slope of
membership func */
double x_intercept_up=slope_of_line*(1.5*start_index_num);
double x_intercept_down=slope_of_line*(2.5*start_index_num);
/* x-axis intercept for increasing (left) and decreasing (right) side of triangle membership
func */

if (x>1.5*start_index_num && x<=2*start_index_num) mu = slope_of_line*x-
x_intercept_up;
else if (x>2*start_index_num && x<=2.5*start_index_num) mu = -
slope_of_line*x+x_intercept_down;
else mu=0;

return(mu);

/* end of mu_index_num_med_fun */
}

```

```

/***** mu_index_num_high_fun *****/ triangle membership function */
/* the mu=0 (min value-left) is 2*start_index_num, and mu=1 (max value) is
2.5*start_index_num */
/* the mu=1 (min value-right) is 3*start_index_num */

double mu_index_num_high_fun(double x, double start_index_num)
{
double mu, slope_of_line=1/(0.5*start_index_num); /* slope of line: slope of
membership func */
double x_intercept_up=slope_of_line*(2*start_index_num);
double x_intercept_down=slope_of_line*(3*start_index_num);
/* x-axis intercept for increasing (left) and decreasing (right) side of triangle membership
func */

```

```

if (x>2*start_index_num && x<=2.5*start_index_num) mu = slope_of_line*x-
x_intercept_up;
else if (x>2.5*start_index_num && x<=3*start_index_num) mu = -
slope_of_line*x+x_intercept_down;
else mu=0;

return(mu);

/* end of mu_index_num_high_fun */
}

/***** mu_index_num_vhigh_fun *****/ mu_index_num_high_fun increases as 1-
1/sqrt(x-x0) */
/* the mu=0 (min value) is 2.5*start_index_num; */
/* x0=2.5*start_index_num-1 (move the index to left by one) */
/* the function is 1-1/sqrt(x-x0) which increases quadratically to infinity */

double mu_index_num_vhigh_fun(double x, double start_index_num)
{
double mu, dummy_var=2.5*start_index_num-1;
/* dummy_var=x0 which adjust so mu=0 @ x=2.5*start_index_num */

if(x>2.5*start_index_num) mu = 1-1/sqrt(x-dummy_var);
else mu=0;

return(mu);

/* end of mu_index_num_vhigh_fun */
}

/***** The flip_pt_membership functions *****/
double mu_flip_pt_bad_fun(double x)
{
double mu;

if (x>0.0 && x<10) mu = -0.1*x + 1;
else mu=0;

return(mu);
/* end of mu_flip_pt_bad_fun */
}

double mu_flip_pt_ok_fun(double x)

```

```

{
double mu;

if (x>0 && x<=10) mu=0.1*x;
else if (x>=10 && x<20) mu=-0.1*x+2;
else mu=0;

return(mu);
/* end of mu_flip_pt_ok_fun */
}

```

```

double mu_flip_pt_good_fun(double x)
{
double mu;

if (x>=10 && x<=20) mu=0.1*x-1;
else if (x>=20 && x<30) mu=-0.1*x+3;
else mu=0;

return(mu);
/* end of mu_flip_pt_good_fun */
}

```

```

double mu_flip_pt_exc_fun(double x)
{
double mu;

if (x>=20 && x<=30) mu=0.1*x-2;
else if (x>=30 && x<40) mu=-0.1*x+4;
else mu=0;

return(mu);
/* end of mu_flip_pt_exc_fun */
}

```

```

double mu_flip_pt_vexc_fun(double x)
{
double mu;

if (x>=30 && x<=40) mu=0.1*x-3;
else mu=0;

return(mu);
/* end of mu_flip_pt_vexc_fun */
}

```

```

}

/***** THE MAX-MIN INFERENCE ENGINE MAMDANI'S RULE OF
IMPLICATION *****/
/* called from each of the nine rules in the main progrsm */
/* max(min(mu_sum_error, mu_index_num), mu_flip_pt) */

double mu_flip_pt_fun(double mu_sum_error, double mu_index_num, double
mu_flip_pt)
{
if (mu_sum_error <= mu_index_num && mu_sum_error > mu_flip_pt) mu_flip_pt =
mu_sum_error;
if (mu_index_num <= mu_sum_error && mu_index_num > mu_flip_pt) mu_flip_pt =
mu_index_num;
else mu_flip_pt = mu_flip_pt;

return(mu_flip_pt);

/* end of rule function */
}

/***** Centroid Defuzzification Funtions *****/
/* used in for loop, output member func is split into three segments */
/* decrease func is the member func sloping down while increase is sloping up. */
/* the mu_flip_decrease (or increase) is mu_value calculated from the fired rulebase. */
/* mu_flip_decrease_slope is calculated based on the membership rules */
/* for a particular value of flip_pt_x_values along the horizontal axis */
double centroid_defuzz_fun(double mu_flip_dec,double mu_flip_dec_slope,double
mu_flip_inc,double mu_flip_inc_slope)
{
double mu_value;

if (mu_flip_dec<=mu_flip_dec_slope && mu_flip_inc<=mu_flip_inc_slope){
    if (mu_flip_dec>=mu_flip_inc) mu_value=mu_flip_dec;
    else if (mu_flip_inc>mu_flip_dec) mu_value=mu_flip_inc;
}

if (mu_flip_dec_slope<mu_flip_dec && mu_flip_inc<=mu_flip_inc_slope){

```



```

        if (mu_flip_dec_slope>= mu_flip_inc) mu_value=mu_flip_dec_slope;
        else if (mu_flip_inc>mu_flip_dec_slope) mu_value=mu_flip_inc;
    }

if (mu_flip_dec<=mu_flip_dec_slope && mu_flip_inc_slope<mu_flip_inc){

    if (mu_flip_dec>=mu_flip_inc_slope) mu_value=mu_flip_dec;
    else if (mu_flip_inc_slope>mu_flip_dec) mu_value=mu_flip_inc_slope;
}

if (mu_flip_dec_slope<mu_flip_dec && mu_flip_inc_slope<mu_flip_inc) {

    if (mu_flip_dec_slope>=mu_flip_inc_slope) mu_value=mu_flip_dec_slope;
    else if (mu_flip_inc_slope>mu_flip_dec_slope) mu_value=mu_flip_inc_slope;
}

return (mu_value);
}

/*****END OF FUZZLETS PROGRAM *****/
/*****
***/

```

Return vector a[] with 1024 points - this is the extended signal.

B.5 Computer Code for Weighted Fuzzy Extension Technique

This is the computer code for the Weighted Fuzzy extension technique. The windowed signal is the vector represented by a[] with 768 points. The return vector a[] with 1024 can be smoothed with the wavelet smoothing code.

```

for (k=0;k<700+1;k++){
    tmp[k]=a[n-k];
}

/* this calculates the sum of error at one freq */
for(i=0;i<index_num_one_freq; i++){

```

```

        mean_one_freq += tmp[i];
    }
    sum_one_freq=mean_one_freq;
    mean_one_freq=mean_one_freq/index_num_one_freq;

    for(i=0;i<index_num_one_freq; i++){
        sum_error_one_freq += (mean_one_freq-tmp[i])*(mean_one_freq-tmp[i]);
    }

    sum_error_one_freq=sum_error_one_freq/index_num_one_freq; /*sum_sq_dev/n */

    for (k=index_num_one_freq;k<2.0*index_num_one_freq;k++) {
/* k=length of skip window to how far back*/
/* k=the point index in the for loop */

        for (kk=0;kk<k+1;kk++) { /* calculate the mean */
            mean1 +=tmp[kk];
        }

        mean1=mean1/(k+1); /*remember index_num=k+1 */

        for (kk=0;kk<k+1;kk++) { /* calculate the sum_error deviation */
            sum_error += (mean1-tmp[kk])*(mean1-tmp[kk]);
        }

/* sum_error = sum_error; /* this is sum_sq_dev */
sum_error = sum_error/(k+1); /* this is sum_sq_dev/n */
/* sum_error = sum_error/sqrt(k+1); /* this is sum_sq_dev/n */

        index_num=k+1;

rating=weightfuzz(sum_error, index_num, sum_error_one_freq, index_num_one_freq);

        /*if (rating >=max_rating){
            flip_mean=mean1;
            max_rating=rating;
        }*/

        printf("tmp[%i]= %le sum_error= %le mean= %lf rating= %lf \n ", k, tmp[k],
sum_error, mean1, rating);

        sum_rating_times_value+=tmp[k]*rating;
        sum_rating+=rating;

```



```

/* sum_error (really sum_sq_deviation/n) & index_num increases as go back away from
end of data */
/* double sum_error, index_num; */
/* sum_error_one_freq is the sum deviation sq at one period of freq/n*/
/* index_num_one_freq is the number of pts (index) for one period of freq */
/* double index_num_one_freq, sum_error_one_freq; */

/* membership func: func return a mu for a given x value (index_num_* or sum_error_* )
*/
double mu_sum_error_sm_fun(double,double), mu_sum_error_med_fun(double,double),
mu_sum_error_lg_fun(double,double);
double mu_index_num_med_fun(double,double),
mu_index_num_mhigh_fun(double,double), mu_index_num_high_fun(double,double);

/* max-min inference func for Mamdani's rule of implication;*/
double mu_flip_pt_fun(double, double, double);

/* the mu values for each fun */
double mu_sum_error_sm,mu_sum_error_med, mu_sum_error_lg;
double mu_index_num_med, mu_index_num_mhigh, mu_index_num_high;
double mu_flip_pt_bad, mu_flip_pt_ok, mu_flip_pt_good, mu_flip_pt_exc;

double flip_pt_x=0.0; /* ranges 0 to 30, @ 0.1 increment; see flip_pt_rating */

/* membership func: func return a mu for a given x value (flip_pt_x) */
double mu_flip_pt_bad_fun(double), mu_flip_pt_ok_fun(double),
mu_flip_pt_good_fun(double),mu_flip_pt_exc_fun(double);

/* these are the membership values of the function; used to calculate mu of the func */
/* at the value of a given x (flip_pt_x) from the above func */
double mu_flip_pt_bad_slope, mu_flip_pt_ok_slope, mu_flip_pt_good_slope,
mu_flip_pt_exc_slope;

/* func returns the mu_value_at_flip_pt_x given the following (in this order) */
/* mu_flip_pt_*1, mu_flip_pt_*1_slope, mu_flip_pt_*2, mu_flip_pt_*2_slope */
double centroid_defuzz_fun(double, double, double, double);

/* these sum for centroid defuzzification method */
double mu_value_at_flip_pt_x=0.0, flip_pt_rating=0.0;
double sum_mu_times_x=0.0, sum_mu=0.0;
/* maximize the flip_pt_rating, higher rating--> better choice for flip point */

/***** !!!!!!!!!NOTE: index_num>=dead_zone !!!!!!!!!*****/
/* Check if index_num<index_num_one_freq ==> return flip_pt_rating=0 & exit */

```

```

if (index_num<index_num_one_freq){
    printf("  ERROR IN CODE!!! Index Number < Dead Zone (Number of points of
one period of freq) \n");
    flip_pt_rating=0;
    return(flip_pt_rating);
}

/* Call the membership functions */

/* sum_error_one_freq & index_num_one_freq are set by the values of each */
/* variable at one complete period of the natural frequency of signal (sensor noise) */

mu_sum_error_sm = mu_sum_error_sm_fun(sum_error, sum_error_one_freq);
mu_sum_error_med = mu_sum_error_med_fun(sum_error, sum_error_one_freq);
mu_sum_error_lg = mu_sum_error_lg_fun(sum_error, sum_error_one_freq);

/*
printf("  mu sum error sm = %lf\n", mu_sum_error_sm);
printf("  mu sum error med = %lf\n", mu_sum_error_med);
printf("  mu sum error lg = %lf\n", mu_sum_error_lg);
printf(" \n");
*/

mu_index_num_med = mu_index_num_med_fun(index_num, index_num_one_freq);
mu_index_num_mhigh = mu_index_num_mhigh_fun(index_num,
index_num_one_freq);
mu_index_num_high = mu_index_num_high_fun(index_num, index_num_one_freq);

/*
printf("  mu index num med = %lf\n", mu_index_num_med);
printf("  mu index num mhigh = %lf\n", mu_index_num_mhigh);
printf("  mu index num high = %lf\n", mu_index_num_high);
printf(" \n");
*/

/* initialize all mu_flip_pt_* to 0 */
mu_flip_pt_bad=0;
mu_flip_pt_ok=0;
mu_flip_pt_good=0;
mu_flip_pt_exc=0;

/* Each rule calls the rule base function */
/* max-min inference func: returns max(min(mu_sum_error_*,
mu_index_num_*),mu_flip_pt_*) for each rule in rule base */

```

```

/* part_1=min(mu_sum_error_*, mu_index_num_*) for Mamdani's rule of implication;
max-min inference; */
/* the max( part_1, mu_flip_pt_*) for the else (union) to pick largest mu_flip_pt_* for
each * */

/* RULE ONE */
if (mu_sum_error_sm>0 && mu_index_num_med>0) {
    mu_flip_pt_exc=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_med,mu_fli
p_pt_exc);
}

/* RULE TWO */
if (mu_sum_error_sm>0 && mu_index_num_mhigh>0){
    mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_mhigh,mu_fl
ip_pt_ok);
}

/* RULE THREE */
if (mu_sum_error_sm>0 && mu_index_num_high>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_sm,mu_index_num_high,mu_flip_pt_b
ad);
}

/* RULE FOUR */
if (mu_sum_error_med>0 && mu_index_num_med>0){

mu_flip_pt_good=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_med,mu_flip_pt
_good);
}

/* RULE FIVE */
if (mu_sum_error_med>0 && mu_index_num_mhigh>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_mhigh,mu_flip_pt
_ok);
}

/* RULE SIX */
if (mu_sum_error_med>0 && mu_index_num_high>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_med,mu_index_num_high,mu_flip_pt_
bad);
}

```

```

/* RULE SEVEN */
if (mu_sum_error_lg>0 && mu_index_num_med>0){

mu_flip_pt_ok=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_med,mu_flip_pt_ok);
}

/* RULE EIGHT */
if (mu_sum_error_lg>0 && mu_index_num_mhigh>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_mhigh,mu_flip_pt_b
ad);
}

/* RULE NINE */
if (mu_sum_error_lg>0 && mu_index_num_high>0){

mu_flip_pt_bad=mu_flip_pt_fun(mu_sum_error_lg,mu_index_num_high,mu_flip_pt_ba
d);
}

/*
printf("  mu_flip_pt_bad = %lf\n", mu_flip_pt_bad);
printf("  mu_flip_pt_ok = %lf\n", mu_flip_pt_ok);
printf("  mu_flip_pt_good = %lf\n", mu_flip_pt_good);
printf("  mu_flip_pt_exc = %lf\n", mu_flip_pt_exc);
printf(" \n");
*/

/* print only for diagnostic checking */
/*
printf("  mu_flip_pt_bad_slope = %lf\n", mu_flip_pt_bad_slope);
printf("  mu_flip_pt_ok_slope = %lf\n", mu_flip_pt_ok_slope);
printf("  mu_flip_pt_good_slope = %lf\n", mu_flip_pt_good_slope);
printf("  mu_flip_pt_exc_slope = %lf\n", mu_flip_pt_exc_slope);
*/

/* for loop for summing up centroid areas of output membership functions */
/* output member func is split into three segments: (bad,ok),(ok,good),(good,exc)*/
/* for each part of the for loop, the centroid_defuzz_fun is called to calculate */
/* the max { min(mu_flip_pt_*1_slope, mu_flip_pt_*1), min(mu_flip_pt_*2_slope,
mu_flip_pt_*2)} */
/* the resulting max (mu_value_at_flip_pt_x) is calculated into the centroid
defuzzification method */

flip_pt_x=-0.142857;

```

```

for (flip_pt_x=-0.142857; flip_pt_x<=1.142857;) {

    if (flip_pt_x<=0.2857){ /* the first part of the ouput membership functions */
        mu_flip_pt_bad_slope = mu_flip_pt_bad_fun(flip_pt_x);
        mu_flip_pt_ok_slope = mu_flip_pt_ok_fun(flip_pt_x);
        mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_bad,
mu_flip_pt_bad_slope, mu_flip_pt_ok, mu_flip_pt_ok_slope);
        sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
        sum_mu = mu_value_at_flip_pt_x + sum_mu;
    }

    if (flip_pt_x>=0.2857 && flip_pt_x<=0.7143){ /* the second part of the ouput
membership functions */
        mu_flip_pt_ok_slope = mu_flip_pt_ok_fun(flip_pt_x);
        mu_flip_pt_good_slope = mu_flip_pt_good_fun(flip_pt_x);
        mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_ok,
mu_flip_pt_ok_slope, mu_flip_pt_good, mu_flip_pt_good_slope);
        sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
        sum_mu = mu_value_at_flip_pt_x + sum_mu;
    }

    if (flip_pt_x>=0.7143 && flip_pt_x<=1.142857){ /* the third part of the ouput
membership functions */
        mu_flip_pt_good_slope = mu_flip_pt_good_fun(flip_pt_x);
        mu_flip_pt_exc_slope = mu_flip_pt_exc_fun(flip_pt_x);
        mu_value_at_flip_pt_x = centroid_defuzz_fun(mu_flip_pt_exc,
mu_flip_pt_exc_slope, mu_flip_pt_good, mu_flip_pt_good_slope);
        sum_mu_times_x = mu_value_at_flip_pt_x * flip_pt_x +
sum_mu_times_x;
        sum_mu = mu_value_at_flip_pt_x + sum_mu;
    }
    flip_pt_x=flip_pt_x+0.001;

/* end of for loop summing all the centroid areas */
}

flip_pt_rating=sum_mu_times_x/sum_mu;

/*
printf(" sum_mu_times_x = %lf AND sum_mu = %lf \n", sum_mu_times_x,
sum_mu);

```



```

printf(" flip point rating = %lf (from weightfuzz function) \n", flip_pt_rating);
*/

if (sum_mu_times_x<0.001){ /* Error Check of sum_mu_times_x */
    printf(" ERROR IN CALCULATION: Deffuzification Error with Sum of (mu
multiplied by x) \n");
    flip_pt_rating=0;
    return(flip_pt_rating);
}

if (sum_mu<=0.001){ /* Error Check of sum_mu */
    printf(" ERROR IN CALCULATION: Deffuzification Error with Sum of mu
\n");
    flip_pt_rating=0;
    return(flip_pt_rating);
}

return(flip_pt_rating);
/* end of newfwave.c function called to calculate rating of flip point */
}

/***** THE MEMBERSHIP FUNCTIONS
*****/
/*****the mu_sum_error membership functions *****/
/* For all the mu_sum_error membership functions are adjusted back to
sum_error_one_period */
/* which is the sum_error @ one period of freq. */

/***** mu_sum_error_sm_fun *****/ sum_error_small decreases linearly such
that */
/* the mu=1 (max value) is sum_error=(4/5)*sum_error_one_per, and mu=0 (min value)
is (9/5)*sum_error_one_period */

double mu_sum_error_sm_fun(double x, double sum_error_one_period)
{

double slope=1/(sum_error_one_period);
double mu, x_intercept_down=slope*(9*sum_error_one_period/5);
/* x_intercept_down=the x-intercept of the membership function with x-axis */

if(x<4*sum_error_one_period/5) mu = 1;

```

```

else if(x>=4*sum_error_one_period/5 && x<=9*sum_error_one_period/5) mu = -
slope*x + x_intercept_down;
else mu=0;

return(mu);
/* end of mu_sum_error_sm_fun */
}

/***** mu_sum_error_med_fun *****/ sum_error_med increase & decreases
linearly such that */
/* the mu=0 (min value-left) is (4/5)*sum_error_one_period */
/* and mu=1 (max value) is (9/5)*sum_error_one_period */
/* the mu=0 (min value-right) is (14/5)*sum_error_one_period */

double mu_sum_error_med_fun(double x, double sum_error_one_period)
{
double slope=1/(sum_error_one_period);
double mu, x_intercept_up=slope*(4*sum_error_one_period/5);
double x_intercept_down=slope*(14*sum_error_one_period/5);
/* x_intercept_up=the x-intercept of the increasing membership function with x-axis */
/* x_intercept_down=the x-intercept of the decreasing membership function with x-axis
*/

if (x>4*sum_error_one_period/5 && x<=9*sum_error_one_period/5) mu=slope*x-
x_intercept_up;
else if (x>9*sum_error_one_period/5 && x<=14*sum_error_one_period/5) mu=-
slope*x+x_intercept_down;
else mu=0;

return(mu);
/* end of mu_sum_error_med_fun */
}

/***** mu_sum_error_lg_fun *****/ sum_error_lg increases as 1-1/(x/x0) */
/* the mu=0 (min value) is (9/5)*sum_error_one_period;
x0=(9/5)*sum_error_one_period */
/* the function is 1-1/(x/x0) which increases quadratic to infinity */

double mu_sum_error_lg_fun(double x, double sum_error_one_period)
{
double mu;

if(x>9*sum_error_one_period/5) mu=1-1/(x/(9*sum_error_one_period/5));
else mu=0;

```

```

return(mu);
/* end of mu_sum_error_lg_fun */
}

/***** The mu_index_num_membership functions
*****/
/* start_index_num is the index at one period of freq */

/***** mu_index_num_med_fun *****/ linearly decreasing membership function
*/
/* linearly decreasing membership function such that */
/* the mu=1 (max value-right) is start_index_num; the mu=0 (min value-left) is
1.33*num_points one period */

double mu_index_num_med_fun(double x, double start_index_num)
{
double mu, slope_of_line=1/(0.333*start_index_num); /* slope of line: slope of
membership func */
double x_intercept_down=slope_of_line*(1.333*start_index_num);
/* x-axis intercept for decreasing (right) side of trapazoid membership func */

if (x>=start_index_num && x<=1.333*start_index_num) mu = -
slope_of_line*x+x_intercept_down;
else mu=0;

return(mu);
/* end of mu_index_num_med_fun */
}

/***** mu_index_num_mhigh_fun *****/ triangle membership function */
/* the mu=0 (min value-left) is 1.0*start_index_num, and mu=1 (max value-middle) is
1.33*start_index_num */
/* the mu=0 (min value-right) is 1.667*start_index_num Triangle membership function
*/

double mu_index_num_mhigh_fun(double x, double start_index_num)
{
double mu, slope_of_line=1/(0.333*start_index_num); /* slope of line: slope of
membership func */
double x_intercept_up=slope_of_line*(1.00*start_index_num);
double x_intercept_down=slope_of_line*(1.667*start_index_num);

```

```

/* x-axis intercept for increasing (left) and decreasing (right) side of triangle membership
func */

if (x>=1.00*start_index_num && x<=1.333*start_index_num) mu = slope_of_line*x-
x_intercept_up;
else if(x>1.333*start_index_num && x<=1.667*start_index_num) mu = -
slope_of_line*x+x_intercept_down;
else mu=0;

return(mu);

/* end of mu_index_num_mhigh_fun */
}

/***** mu_index_num_high_fun *****/ mu_index_num_high_fun increases as 1-
1/sqrt(x-x0) */
/* period = start_index_num; the mu=0 (min value) is 1.333*start_index_num; */
/* x=2.0*start_index_num-1 (move the index to left by one) */
/* the function is 1-1/(x-x0) which increases quadraticly to infinity */

double mu_index_num_high_fun(double x, double start_index_num)
{
double mu, dummy_var=1.333*start_index_num-1;
/* dummy_var=x0 which adjust so mu=0 @ x=1.333*start_index_num */

if(x>1.333*start_index_num) mu = 1-1/sqrt(x-dummy_var);
else mu=0;

return(mu);

/* end of mu_index_num_high_fun */
}

/***** The flip_pt_membership functions *****/
double mu_flip_pt_bad_fun(double x)
{
double mu;

if (x>=-0.142857 && x<=0.2857) mu =-7*x/3+0.666667;
else mu=0;

return(mu);
/* end of mu_flip_pt_bad_fun */
}

```

```

double mu_flip_pt_ok_fun(double x)
{
double mu;

if (x>=-0.142857 && x<=0.2857) mu=7*x/3+0.33333;
else if (x>=0.2857 && x<0.7142857) mu=-7*x/3+1.66667;
else mu=0;

return(mu);
/* end of mu_flip_pt_ok_fun */
}

double mu_flip_pt_good_fun(double x)
{
double mu;

if (x>=0.2857 && x<=0.7142857) mu=7*x/3-0.66667;
else if (x>=0.7142857 && x<=1.142857) mu=-7*x/3+2.66667;
else mu=0;

return(mu);
/* end of mu_flip_pt_good_fun */
}

double mu_flip_pt_exc_fun(double x)
{
double mu;

if (x>=0.7142857 && x<=1.142857) mu=7*x/3-1.66667;
else mu=0;

return(mu);
/* end of mu_flip_pt_exc_fun */
}

/***** THE MAX-MIN INFERENCE ENGINE MAMDANI'S RULE OF
IMPLICATION *****/
/* called from each of the nine rules in the main progrsm */
/* max(min(mu_sum_error, mu_index_num), mu_flip_pt) */

double mu_flip_pt_fun(double mu_sum_error, double mu_index_num, double
mu_flip_pt)
{

```

```

if (mu_sum_error <= mu_index_num && mu_sum_error > mu_flip_pt) mu_flip_pt =
mu_sum_error;
if (mu_index_num <= mu_sum_error && mu_index_num > mu_flip_pt) mu_flip_pt =
mu_index_num;
else mu_flip_pt = mu_flip_pt;

return(mu_flip_pt);

/* end of rule function */
}

/***** Centroid Defuzzification Funtions *****/
/* used in for loop, output member func is split into three segments */
/* decrease func is the member func sloping down while increase is sloping up. */
/* the mu_flip_decrease (or increase) is mu_value calculated from the fired rulebase. */
/* mu_flip_decrease_slope is calculated based on the membership rules */
/* for a particular value of flip_pt_x_values along the horizontal axis */
double centroid_defuzz_fun(double mu_flip_dec,double mu_flip_dec_slope,double
mu_flip_inc,double mu_flip_inc_slope)
{
double mu_value;

if (mu_flip_dec<=mu_flip_dec_slope && mu_flip_inc<=mu_flip_inc_slope){

    if (mu_flip_dec>=mu_flip_inc) mu_value=mu_flip_dec;
    else if (mu_flip_inc>mu_flip_dec) mu_value=mu_flip_inc;
}

if (mu_flip_dec_slope<mu_flip_dec && mu_flip_inc<=mu_flip_inc_slope){

    if (mu_flip_dec_slope>= mu_flip_inc) mu_value=mu_flip_dec_slope;
    else if (mu_flip_inc>mu_flip_dec_slope) mu_value=mu_flip_inc;
}

if (mu_flip_dec<=mu_flip_dec_slope && mu_flip_inc_slope<mu_flip_inc){

    if (mu_flip_dec>=mu_flip_inc_slope) mu_value=mu_flip_dec;
    else if (mu_flip_inc_slope>mu_flip_dec) mu_value=mu_flip_inc_slope;
}
}

```

```
if (mu_flip_dec_slope < mu_flip_dec && mu_flip_inc_slope < mu_flip_inc) {  
    if (mu_flip_dec_slope >= mu_flip_inc_slope) mu_value = mu_flip_dec_slope;  
    else if (mu_flip_inc_slope > mu_flip_dec_slope) mu_value = mu_flip_inc_slope;  
}  
  
return (mu_value);  
}
```

Return vector a[] with 1024 points - this is the extended signal.

VITA

Jonathan Mark Hynson

Candidate for the Degree of

Master of Science

**Thesis: IMPROVED SIGNAL EXTENSION METHODS FOR
WAVELET SMOOTHING**

Major Field: Chemical Engineering

Biographical:

Personal Data: Born in Knoxville, Tennessee, on February 20, 1971; the son of Larry and Kathy Hynson.

Education: Graduated from Stillwater High School, Stillwater, Oklahoma, in May 1989. Received Bachelor of Science degree in Chemical Engineering, Stillwater, Oklahoma in December, 1993. Completed the requirements for the Master of Science degree in Chemical Engineering at Oklahoma State University, Stillwater, Oklahoma in December 1996.

Professional Experience: Internships with Mobil Chemical Company, Beaumont, Texas summer of 1991; Koch Refining Company, St. Paul, Minnesota, summer of 1992; Southwest Refining Company (subsidiary of Kerr-McGee) Corpus Christi, Texas summer of 1993; Teaching Assistant at Oklahoma State University, Stillwater, Oklahoma, January 1994 - December 1994 and September 1995 - December 1995; Research Assistant at Oklahoma State University, Stillwater, Oklahoma, January 1995 - August 1995.

Professional Memberships: Registered Engineer Intern in the State of Oklahoma; American Institute of Chemical Engineers, Tau Beta Pi, Omega Chi Epsilon.