ECG Compression Algorithms Comparisons among EZW, Modified EZW and Wavelet Based Linear Prediction

by

Xiaohua Zhou

A Thesis

Presented to the Faculty of

Binghamton University

in Partial Fulfillments of the Requirements of the Degree of

Master of Science in Mechanical Engineering

February 3, 2004

Approved:

Advisor

Department Chairperson

(Date: Month and Year)

DEDICATION

To My Parents, My Sister and My Husband

Acknowledgements

I must first express my sincere appreciation to my thesis advisor, Dr. Mark Fowler, without his teaching on data compression theories and advices and supports through the course of this research, I could not have been able to complete my thesis.

I am also indebted to many faculty members, staff members, and students at Binghamton University who helped me to enrich my academic experiences.

Finally, I want to thank my parents, sibling and my husband for their love and supports through the time I am studying at Binghamton University and writing my thesis.

Contents

List of Tables

List of Figures

List of Abbreviations

ECG	Electrocardiography
PAN	Period and Amplitude Normalization [2]
WBLP	Wavelet Based Linear Prediction [2]
EZW	Embedded Zerotrees of Wavelet [1]
MEZW	Modified Embedded Zerotrees of Wavelet

Abstract

This thesis is to study the compression of ECG samples by EZW and Modified EZW (MEZW) algorithms, and compare their effectiveness with a wavelet based linear prediction (WBLP) method published in a journal paper.

In EZW algorithm, 3-level decomposition is performed to the original ECG samples, and the wavelet coefficients at different sub-band representing the same spatial location in the ECG samples are loaded into a spanning tree. Through several dominant passes and subordinate passes, significant coefficients are selected and refined progressively and encoded following the spanning tree structure.

MEZW is a method derived from EZW method. The difference between them is that only one dominant pass and one subordinate pass are performed in MEZW. The dominant pass is to select all the significant coefficients above the threshold at one time and they are uniformly quantized in the subordinate pass.

This thesis also simulates the compression results from WBLP. In this method, *QRS* detection is first conducted to detect the ECG beats, and each beat is period normalized and amplitude normalized to create the PAN beats. Discrete wavelet transform is performed on the PAN beats and a number of significant coefficients at the same index locations across the beats are selected for every beat. These significant wavelet coefficients are stacked across the beats to minimize the variance of the data sequence before feeding into the linear predication filter. The outputs of the linear prediction filter are the residual sequences with further reduced variances.

The simulation results show that MEZW has the best compression performance among the three. The abnormal ECG signal reconstruction achieved by MEZW at an average rate of 230b/s is of high quality. While WBLP is very effective at compressing sinus-rhythm ECG signals, when compressing abnormal ECG signals, it has to keep the low compression ratios to avoid the big reconstruction errors. MEZW and EZW are good substitutes for WBLP at compressing abnormal ECG signals.

If the computation time is a concern, both EZW and MEZW, especially EZW method, need to be applied to each of the ECG beats in order to break down the computational complexity.

Chapter 1 Introduction

This thesis presents research on applying 1-D EZW [1] and modified EZW (MEZW) algorithms to compress the ECG signals to maximize the compression ratios with minimum distortion rates. The thesis simulates the 1-D EZW algorithms applied to ECG compression in several different ways and gives results comparable with the those from a journal published novel compression method: Wavelet Based Linear Prediction [2] (WBLP). Moreover, it demonstrates the EZW's superior effectiveness to WBLP. This thesis also simulates the results from WBLP algorithm and discusses its advantages and disadvantages.

1.1 Motivation

Compression of electrocardiography (ECG) is necessary for efficient storage and transmission of the digitized ECG signals. A typical ECG monitoring device generates a large amount of data in the continuous long-term (24-48 hours) ambulatory monitoring tasks. For good diagnostic quality, up to 12 different streams of data may be obtained from various sensors placed on the patient's body. The sampling rates of ECG signals are from 125Hz to 500Hz, and each data sample may be digitized into 8 to 12 bits binary number. Even with one sensor at the lowest sampling rate of 125 Hz and 8-bit encoding, it generates data at a rate of 7.5KB per minute and 450KB per hour. For a sampling rate of 500Hz and 12-bit encoding recording, it generates data at a rate of 540KB per minute and 30MB per hour. The data rate from 12 different sensors totally will generate 12 times amount of data and it is enormously big. Besides, recording over a period of time as long as 24 hours maybe needed for a patient with irregular heart rhythms. The monitor device such as Holter must have a memory capacity of about 400-800 MB for a 12-lead recording, but such a big memory cost may render a solid-state commercial Holter device impossible. Thus, efficient ECG data compression to dramatically reduce the data storage capacity is a necessary solution. On the other hand, it makes possible to transmit ECG data over a telephone line from one cardiac doctor to another cardiac doctor to get opinions.

1.2 Research Goals

The primary goal of this research is to contribute to the fundamental understanding of applying of 1-D Embedded Zero Tree and modified 1-D Embedded Zero Tree algorithm to compress the ECG signals, and compare their compression effectiveness and computational complexities with WBLP method, a known method published in a journal paper.

It will take four steps to accomplish the goals: 1) To detect the *QRS* complex accurately. 2) To simulate the results from WBLP method. 3) To apply the 1-D EZW and modified 1-D EZW algorithm to compress the ECG signals. 4) To compute the computational complexities of the WBLP method and EZW based method.

1.3 Research Background

The electrocardiogram (ECG) essentially reads the electrical impulses that stimulate the heart to contract, it is probably the most useful tool to determine whether the heart has been injured or how it is functioning.

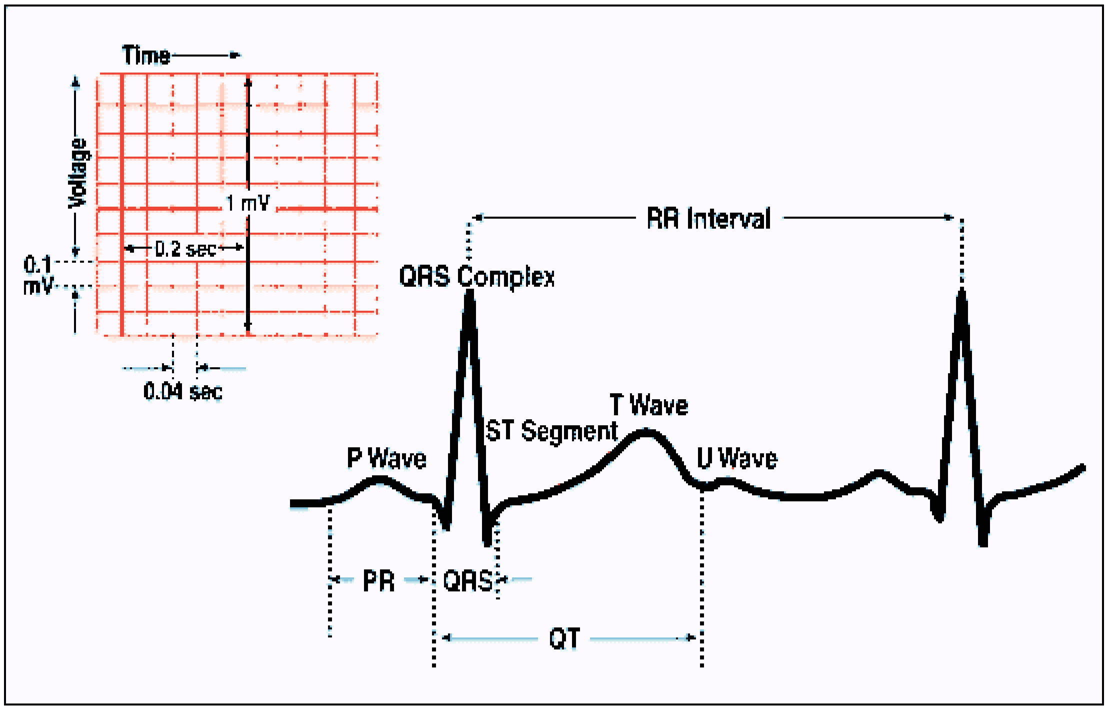


Figure 1.1 Standard ECG waves and intervals [7].

As figure 1 shows, the ECG is made up of a number of segments or waves: P wave, *QRS* wave, and T wave, they are diagnostic critical waves. The P wave represents the atrial depolarization where the blood is squeezed from the atria to the ventricles. The *QRS* segment is when the ventricles depolarize and squeeze the blood from the right ventricle to the aorta; The T wave represents the period of time when the ventricles repolarize (get ready for the next heart beat).

Exiting data compression techniques for ECG signals fall in three categories: 1) the direct data transformation methods, 2) transformation methods, and 3) parameter extraction techniques.

This thesis uses one of the transformation methods to analyze the spectral and energy distributions of the ECG signals and to detect and eliminate the redundancies. The transformation is conducted to each of the ECG beats. ECG beats are delineated by RR waves, and one RR wave starts from the peak sample of one *QRS* complex, and ends to the sample right before the peak of the next *QRS* complex, as figure 1 shows.

The basic concept of the cycle-to-cycle compression is to represent a periodic signal by one cycle period and a count of the total number of cycles, and it is only valid for strict periodic signals where all the signal cycles are the same. Though the ECG waveforms do not bear such characteristic, ECG is considered a quasi-periodic signal [8], especially for a normal ECG as shown in figure 1.

It is noted that it is very important to accurately detect the *QRS* complex for cycle to cycle compression because it guarantees the adjacent cycles are statistically dependent to one another. While it is easier to detect the *QRS* complex from the normal ECG signals, it can be very difficult to accurately detect it from the abnormal ECG signals. This thesis uses the technique reported in [3] for the *QRS* detection, and it applies three steps to process the original ECG signals: 1) linear digital filtering, 2) nonlinear transformation, 3) decision rule algorithm.

After the delineation of the *QRS* complex, both WBLP algorithm and EZW algorithm can use the results to conduct cycle-to-cycle compression.

The WBLP method normalizes the period of each beat by resampling each beat so that each beat contains the same number of samples. It also normalizes the amplitude of each beat by scaling a factor of maximum amplitude of the beat. Thus, the generated period and amplitude normalized (PAN) beats bear more correlations. Next, the Daubechies-4 (D4) wavelet is used for representing each PAN beat, and Mallat's pyramidal DWT algorithm is used to compute the wavelet coefficients. To increase the compression ratio, only the residual sequence obtained after linear prediction of the significant wavelet coefficients is transmitted to the decoder [2]. At the decoder, the original signals are reconstructed by the significant wavelet coefficients.

In the 1-D EZW algorithm, each ECG beat will be decomposed into several sub-bands by wavelet transform, and there are wavelet coefficients in different sub-bands that represent the same spatial location in the ECG beat. Each coefficient at a lower band will have two coefficients from its next higher band as its descents, so it is easy to visualize it as a tree structure. The coefficients closer to the root of the tree normally have higher magnitudes than coefficients further away from the root. The large compression ratio is achieved by conducting dominant passes to select the wavelet coefficients bigger than the threshold and encode their positions following the tree's structure, and the recording of the selected significant wavelet coefficients are refined by the subordinate passes.

Modified the 1-D EZW algorithm is to only conduct one dominant pass, and in the subordinate pass, the selected wavelet coefficients are uniformly quantized.

This thesis also applies 1-D EZW and 1-D modified EZW algorithms directly to the ECG samples without the *QRS* detection. While they bypass the hassle to delineate the beats, their worst case computational complexity, especially for 1-D EZW, to too high to put them into use if the computation time is a concern.

Chapter 2 Algorithms

This section will discuss the algorithms to compress the ECG signals. One is the journal published algorithm: WBLP [2], the other one is 1-D EZW method. Both of them employ wavelet transform to compress the digitized ECG signals.

From the generic EZW algorithm, this section derives modified 1-D EZW (MEZW) method in order to on one hand reduce the computational complexity of the generic EZW method, and on the other hand, achieve higher compression ratios with comparable distortion rates. Because the beat detections involve a lot of extra pre-processing work, especially it can be very difficult to delineate beats from ECG signals with abnormal waves, this section explores direct application of EZW on compressing ECG samples without *QRS* detections.

The structure of this section is to first talk about the detections algorithm since it is the pre-processing step required for all the cycle-to-cycle compression algorithms. Next, this section will lay out all the compression algorithms one by one in details.

2.1 *QRS* Detections

QRS detection is the most important pre-processing step for cycle-to-cycle compression algorithms. Inaccurate delineations of ECG beats will detriment the inter-cycle correlations between the beats and result in more information redundancies in the compression process.

An ECG beat is defined as the signal sample from one *R*-wave to the next. The technique documented in [3] uses non-syntactic approach to detect *QRS* complex: rely on characteristic features of *QRS* complexes to perform the simple detection.

The beat detection algorithm can be broken down into three sections: linear digital filtering, nonlinear transformation, and decision rule algorithms as the figure 2.1 shows.

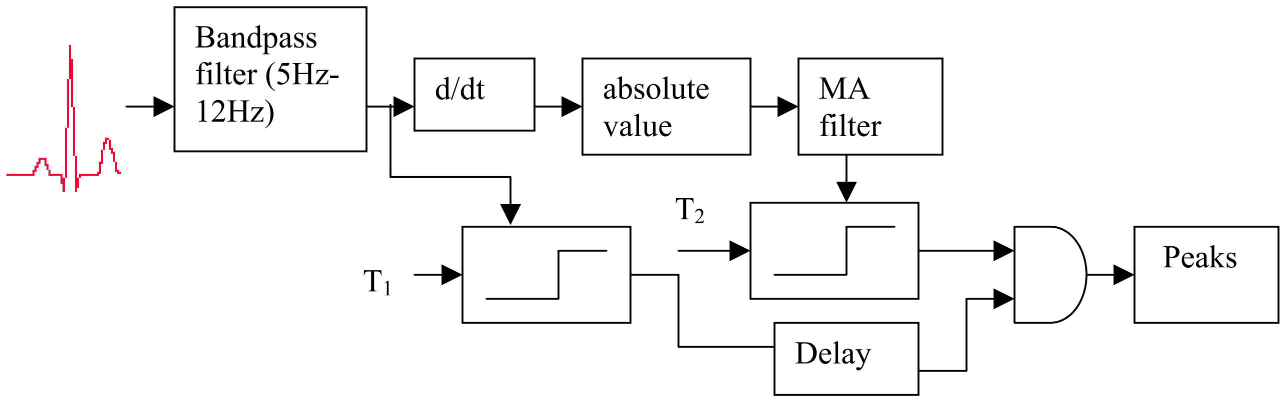


Figure 2.1: Block diagram of *QRS* detections.

2.1.1 Band-pass filter

The band-pass filter shown in figure 2.2 is two filters cascaded, one is a low pass filter, and the other one is a high pass filter as seen in figure 2.2.

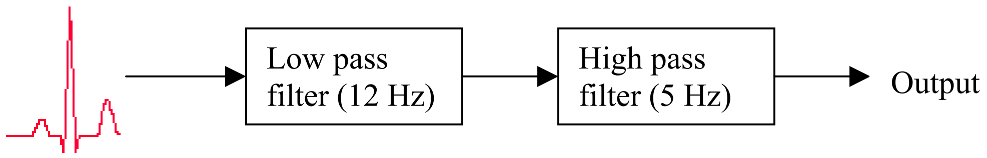


Figure 2.2: Block diagram of the band pass filter

The transfer function of the second-order low pass filter is [3]

$$H(z) = \frac{(1 - z^{-6})^2}{(1 - z^{-1})^2}. \tag{2.1}$$

and its difference equation [3] is

$$\begin{aligned} y(nT) = 2y(nT - T) - y(nT - 2T) + x(nT) \\ - 2x(nT - 6T) + x(nT - 12T). \end{aligned} \tag{2.2}$$

where T is the sampling period. The cutoff frequency is about 12Hz and the gain is 36. The filter processing delay is 6 samples.

The transfer function of the high pass filter is given by [3]

$$H(z) = \frac{(-1 + 32z^{-16} + z^{-32})}{(1 + z^{-1})}. \tag{2.3}$$

Its difference equation is [3]

$$\begin{aligned} & y(nT) = 32x(nT - 16T) \\ & -[y(nT - T) + x(nT) - x(nT - 32T)]. \end{aligned} \tag{2.4}$$

The low cutoff frequency of this filter is about 5 Hz, the gain is 32.

2.1.2 Derivatives

Differentiation of the filtered signals is to provide the slope information of *QRS* complex since there are quick rise and fall times of the *QRS* complex in the ECG signals, taking the derivative of the ECG would make it easier to detect when the *QRS* complex occur.

The transfer function of the five-point differentiation equation is given by [2]

$$H(z) = (\frac{1}{8}T)(-z^{-2} - 2z^{-1} + 2z^{1} + z^{2}). \tag{2.5}$$

Its difference equation is given by [2]

$$y(nT) = (\frac{1}{8}T)[-x(nT-2T)-(2x(nT-T) \\ +2x(nT+T)+x(nT+2T)]. \tag{2.6}$$

It has 2 samples delay.

2.1.3 Nonlinear transform

Instead of squaring the output signal from the derivative filter point by point as stated in [3], this thesis rectifies the signals by taking their absolute values [9], thus reducing the gain sensitivities to improve the performance of the decision rule algorithm described in section 2.1.5.

The equation of this operation is

$$y(nT) = \sqrt{x(nT)^2}. \tag{2.7}$$

2.1.4 Moving-window integration

The window size has to be taken properly, neither so wide that merges the *QRS* complex and *T* wave together, nor so narrow that produces several peaks in the integration waveform. The proper window size will give waveform feature information besides the *R* wave. It is calculated from [3]

$$y(nT) = (\frac{1}{N})[x(nT-(N-1)T)+x(nT-(N-2)T)+........+x(nT)]. \tag{2.8}$$

where *N* is the width of the integration window [3]. This thesis takes *N* as 30.

2.1.5 Decision rule algorithm

As seen in figure 2.1, the algorithm sets two thresholds T_1 and

T_2 to make decisions. T_1 is set for the filtered ECG, and T_2 is set for the signals produced by the moving window integration.

Thresholds T_1 and T_2 are running estimated from beat-to-beat in order to adapt to the rises and falls of *R*-waves' peak amplitudes. Processing delays are considered to estimate the average *R*-*R* intervals. If an *R*-wave is not detected after the maximum time interval, the algorithm will go back to a certain time interval to search for possible *R* wave candidate by using a new set of lower thresholds.

Paper [3] gives more details about how to set up the two thresholds and outlines the steps to implement the decision rules algorithms through Matlab simulation.

2.1.6 Simulation Results

This thesis uses the ECG signals from MIT-BIH ECG Compression Test Database (cdb) and MIT-BIH Normal Sinus Rhythm Database (nsrdb) hosted in PhysioBank [4]. Figure 2.3 shows the original ECG signals from 16773, one of the sinus-rhythm ECG records, and a sequence of linear filtering and nonlinear transform results.

ECG 12431_03 is the ECG signals containing some abnormal beats. Figure 2.4 shows part of the original signal samples, filtering and detection results.

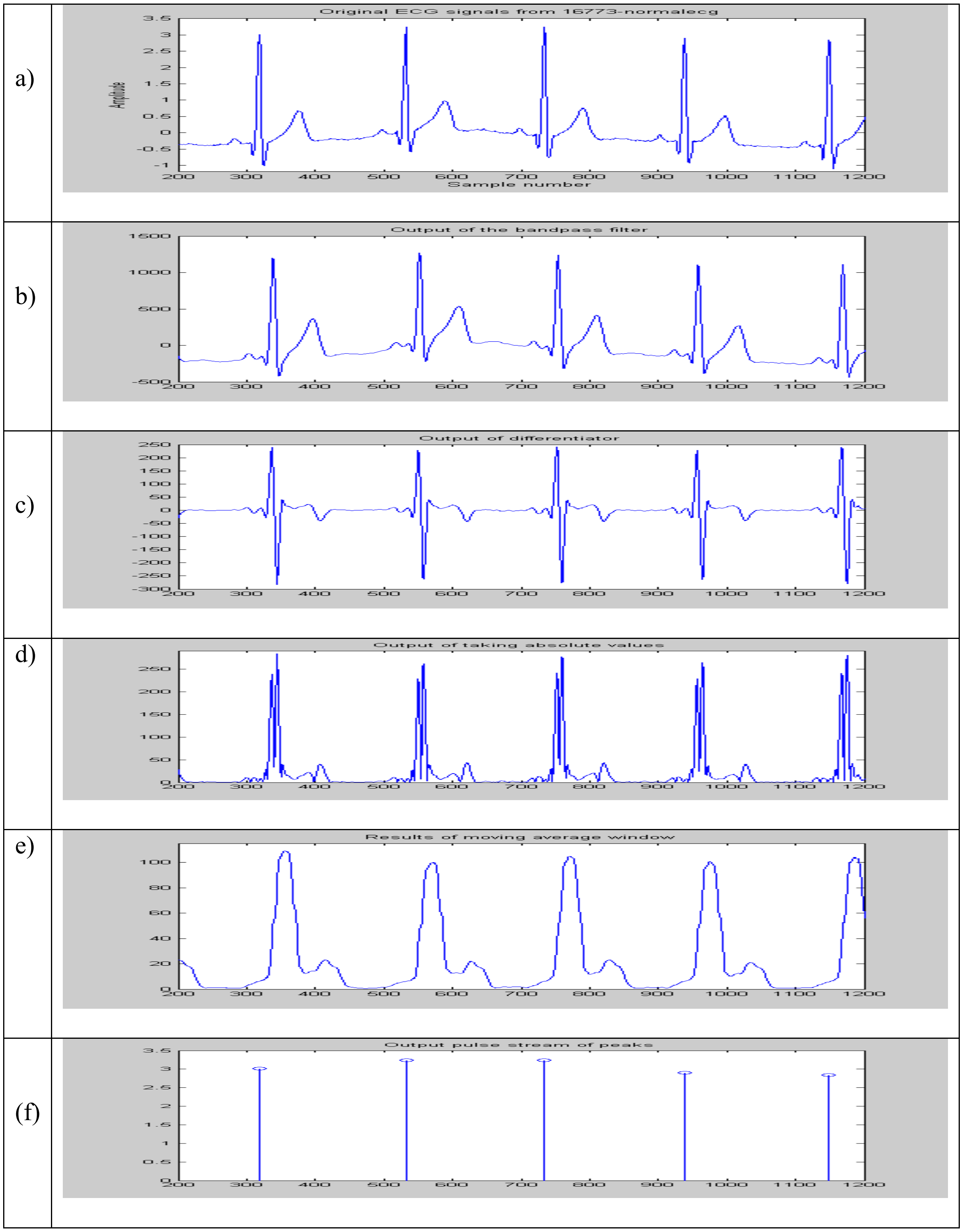


Figure 2.3: (a) Original ECG signal 16773. (b) Output of band-pass filter. (c) Output of differentiator. (d) Output of absolute values. (e) Results of moving-window integration. (f) Output pulse stream of detected peaks.

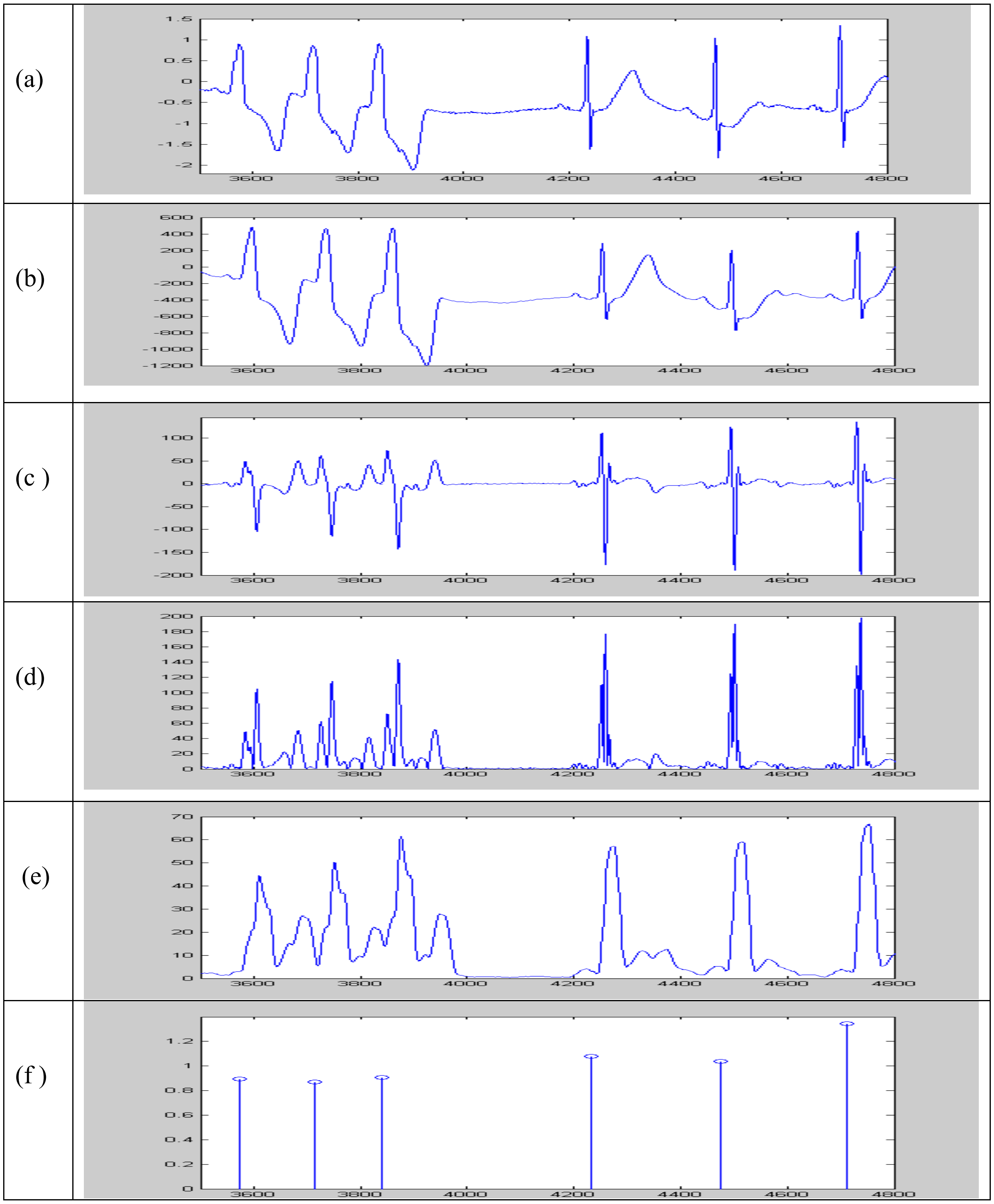


Figure 2.4: (a) Original ECG signal from 12431_03 (b) Output of band-pass filter. (c) Output of differentiator. (d) Output of absolute values. (e) Results of moving-window integration. (f) Output pulse stream of detected peaks.

2.2 Wavelet Based Linear Prediction Algorithm

After the beats detections, this algorithm conducts the cycle-to-cycle compressions.

2.2.1 Period and Amplitude Normalizations

In order to bring more inter-cycle and intra-cycle correlation among the beats, the algorithm re-samples each beat to convert beats originally having different periods into beats with a constant period. The selected sampling rate should be always higher than each beat's original sampling rate to guarantee no distortion, and the modified sampling rate should satisfy Nyquist criteria. The mean beat period (MBP) is estimated from some initial cycles of the data being coded [2].

Thus, at the encoder side, each original beat will send to an interpolation filter with its individual up-sampling factor L_i, and at the decoder side, each PAN beat will bring back to its original sampling rate by a decimation filter with down-sampling factor M_i, and M_i should be equal to L_i. Figure 2.5 given by [2] shows the diagram of this process.

$$\begin{tikzcd}
x_i(n) \arrow[r] & \boxed{\uparrow L_i} \arrow[r, "x_1(n)"] & \boxed{H(z)} \arrow[r, "x_2(n)"] & \boxed{\downarrow M_i} \arrow[r] & y_i(n)
\end{tikzcd}$$

Figure 2.5: Period Normalization Diagram.

Through dividing every signal sample within the beat by the maximum amplitude value of that beat, amplitude normalization makes every beat having the highest amplitude as unity. If the MBP is bigger than every original beat's period, the near perfect reconstruction from PAN is guaranteed as Figure 2.6 shows.

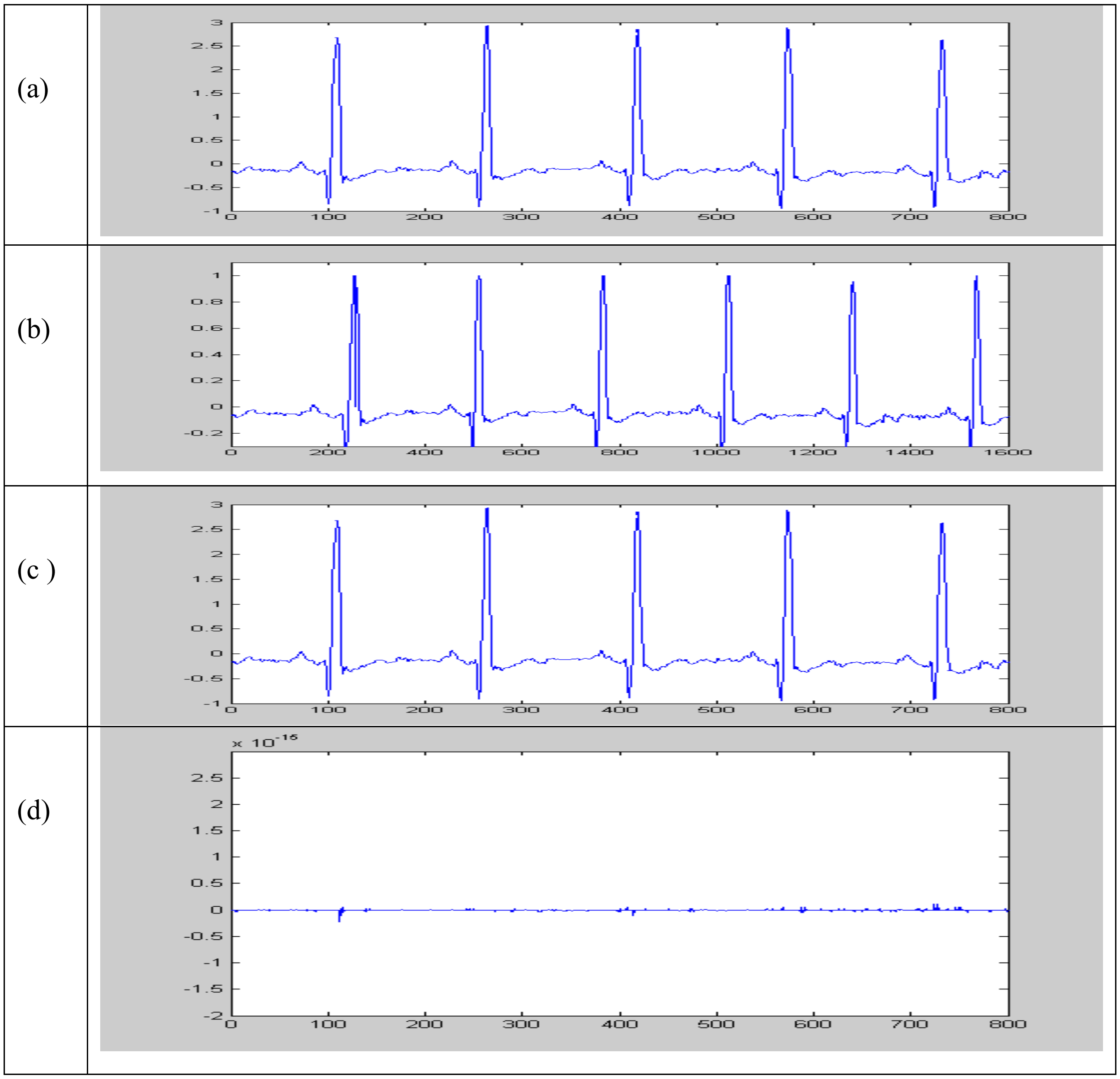


Figure 2.6 (a) Original ECG beats (b) PAN beats (c) Reconstructed from PAN beats (d) Reconstruction errors.

It is clear from Figure 2.6(d) that the recovery errors from PAN beat to the original beats are in the level of 10^{-15}, so it is negligible and the process can be considered perfectly reversible.

But for some ECG signals, when some beats have original periods bigger than MBP, the reconstructions from PAN beats will not be nearly perfect. For example, in

ECG record 12431_03, the 18th and 9th beats originally contain 394 samples and 258 samples, both of them are bigger than MBP which is 256 samples. The period normalization will bring in distortions to these two beats as figure 2.7 (c) shows.

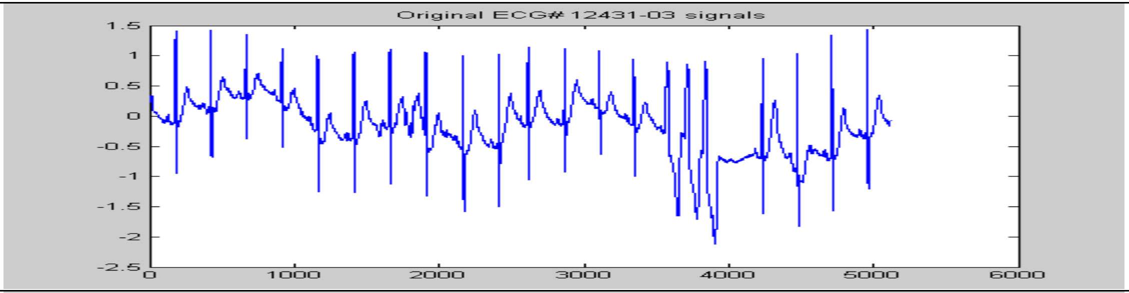
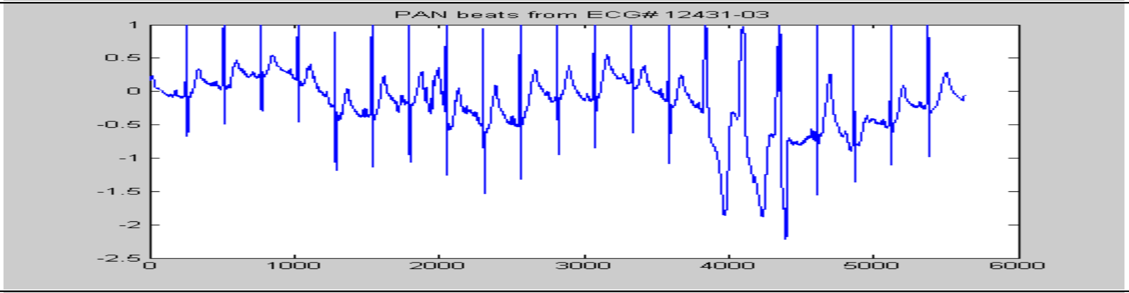
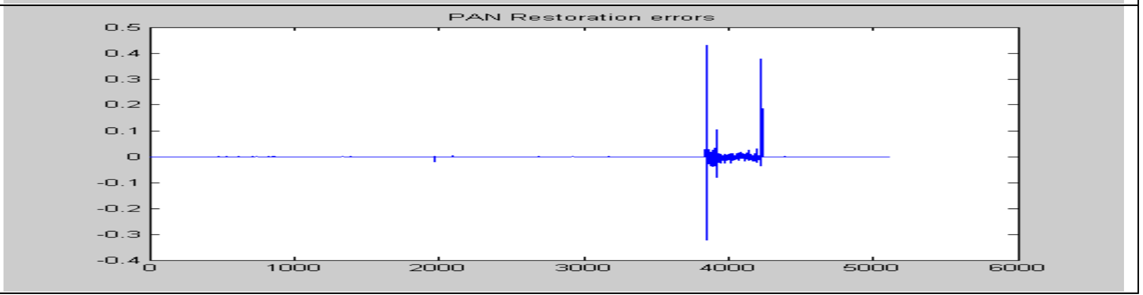
(a)	
(b)	
(c)	

Figure 2.7 (a) Original ECG 12431_03; (b) PAN beats; (c) PAN restoration errors.

2.2.2 Significant Wavelet Coefficients

The algorithm uses Daubechies-4 (D4) wavelet for representing each PAN beat, and uses Mallat's pyramidal DWT algorithm to compute the wavelet coefficients for the signal. The algorithm normalizes the beat to be 256 samples long as Mallat's algorithm requires that the number of samples in the sequence be a power of 2.

Because not all the wavelet coefficients are significant, only a fixed set of significant coefficients which can keep important rhythm and morphological information need to be retained for the beats reconstruction. The algorithm chooses first K beats and sorts the absolute values of each beat's wavelet coefficients in the descending order. The final set of N_R number of significant coefficients is the union of the N_m number of significant coefficients from each beat. The paper [2] reports the N_R is 50, K is15 and N_m is 30.

The simulation result shows that for ECG signal 16265, the locations of the retained approximation wavelet coefficients are:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 47 116 117 119 120 122 123 124 125 126 127 128 129 130 131

and the locations of the retained detail wavelet coefficients are:

1 2 5 125 126 127 129 130

Thus, for every PAN beat, only wavelet coefficients at these locations will retain their values, all the rest will simply be set as zeros.

Figure 2.8(a) plots one of the PAN beats having 256 samples in the sequence. 2.8(b) and 2.8(c) show respectively the plot of approximate coefficients and detail coefficients from one PAN beat. Each wavelet transformed PAN beat consists of 131 approximate and 131 detail coefficients.

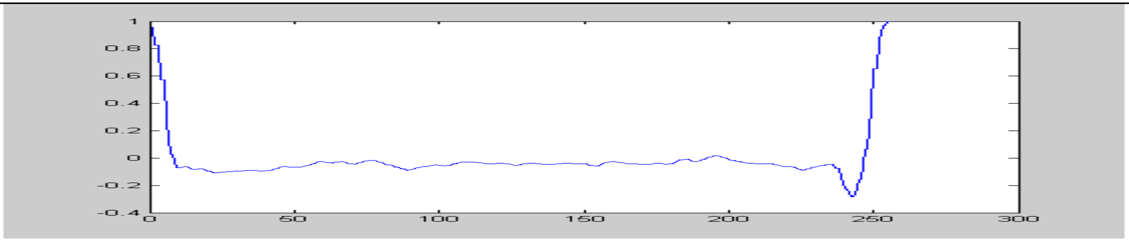
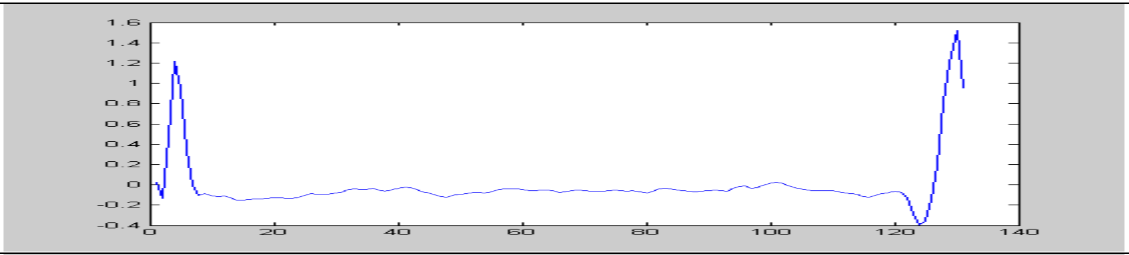
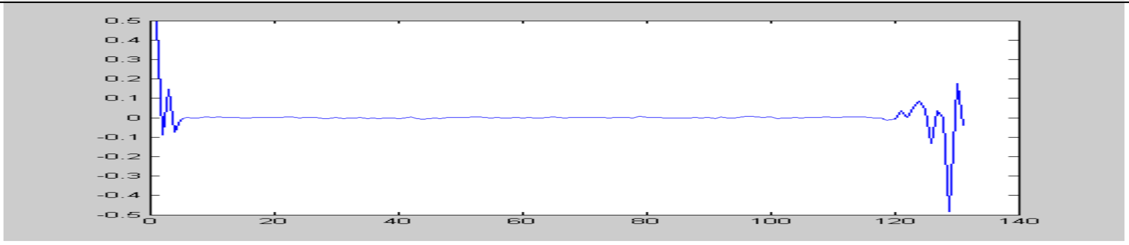
(a)	
(b)	
(c)	

Figure 2.8 (a) One PAN beat. (b) Approx. coeffs. (c) Detail coeffs.

Figure 2.9(a) shows the 42 retained approximate coefficients, and all the other approximate coefficients are set as zeros. 2.9(b) shows the 8 retrained detail coefficients, and the rest detail coefficients are set zeros.

2.2.3 Linear Prediction

In order to further reduce the bit cost to encode the wavelet coefficients, wavelet coefficients across the PAN beats at the corresponding scales and locations are stacked to form a near-cyclo-stationary sequence.

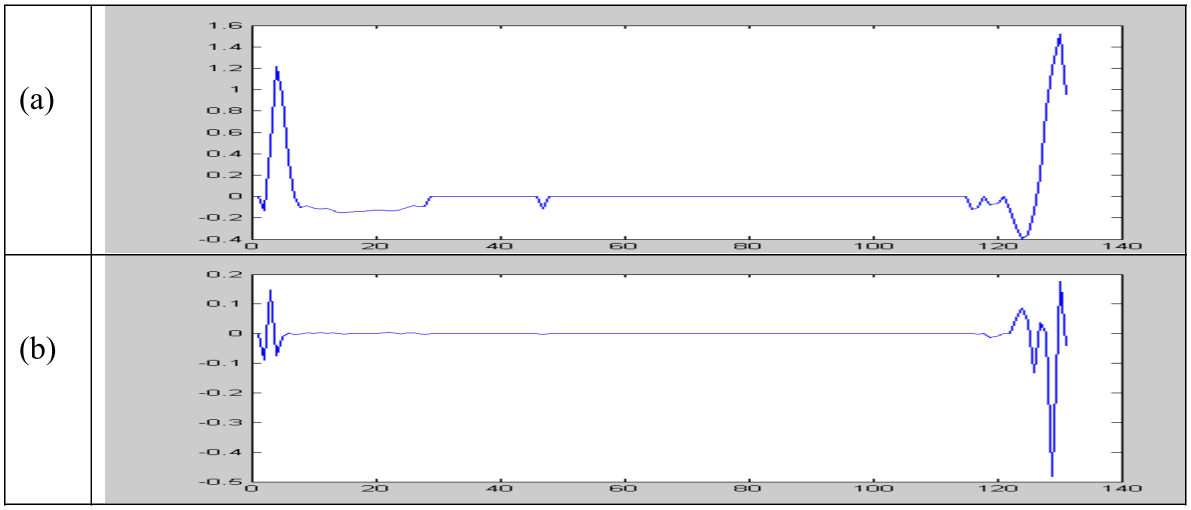


Figure 2.9 (a) Retained 42 approx. coeffs. (b) Retained 8 detail coeffs.

The expression of the grouped vector sequence is given in [2], figure 2.10 shows one group of stacked vectors, they are formed by stacking every PAN beat's wavelet coefficients at scale 4 and location 9. The PAN beats are from original ECG record 16265.

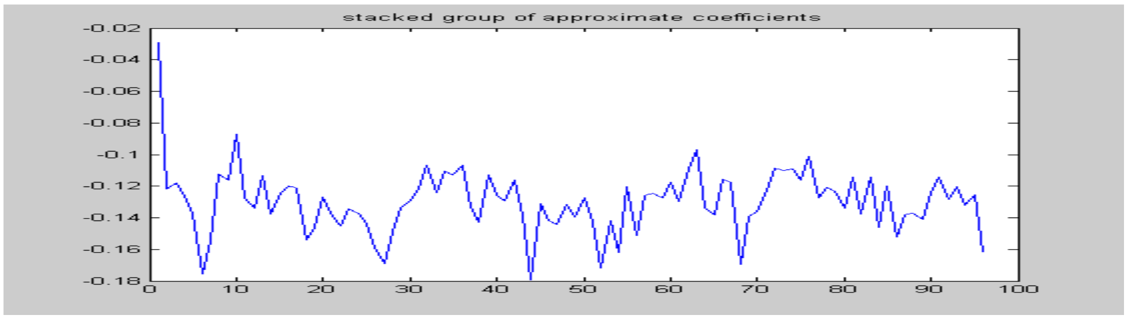


Figure 2.10 Stacked Wavelet Coeffs at scale 4 and location 9 across ECG 16265's PAN beats.

Figure 2.11 shows another beat from ECG record 12431_02.

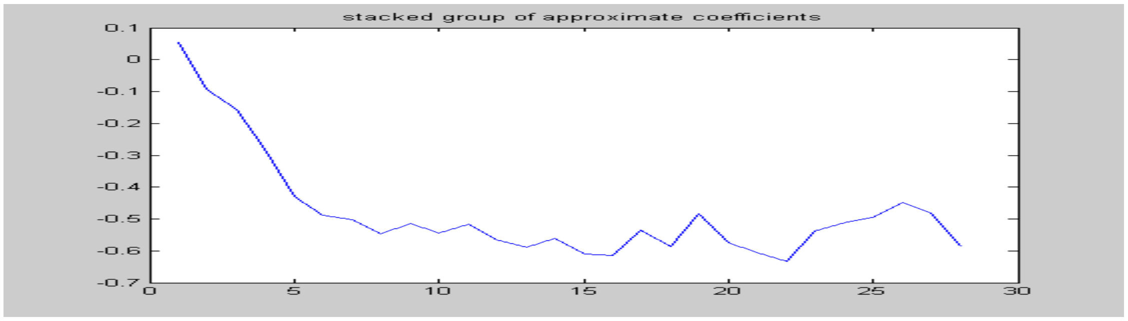


Figure 2.11 Stacked Wavelet Coeffs. at scale 4 and location 9 across ECG 12431_02's PAN beats.

The variance of the sequence in figure 2.10 is only 4.0255e-004, and the variance of the sequence in figure 2.11 is 0.0263.

The average variance of the stacked sequences for ECG record 16265 from totally 96 PAN beats is 0.0084, and if not performing stacking, the average variance is 0.0711. The later one is almost 10 times bigger than the former. Stacking the wavelet coefficients does help to reduce the variances of the sequence of data processed.

For the ECG record 12341_02, the average variances are 0.0669 and 0.1205 respectively from performing stacking and non-stacking. The later one is almost double the former one.

Thus, it is obvious that stacking the coefficients at corresponding scales and locations will help reduce the variance of the data sequence, and help reduce the bit cost to encode them.

Differential encoding is performed on the stacked sequences. Autocorrelation method is employed to find the linear predictor coefficients [5], and only the residues, or the difference between the samples are encoded [6]. The differencing operation is performed without the accumulation of the quantization noise. [5]

Figure 2.12 shows the residues of the sequence in figure 2.10. Its variance is 4.4146e-004.

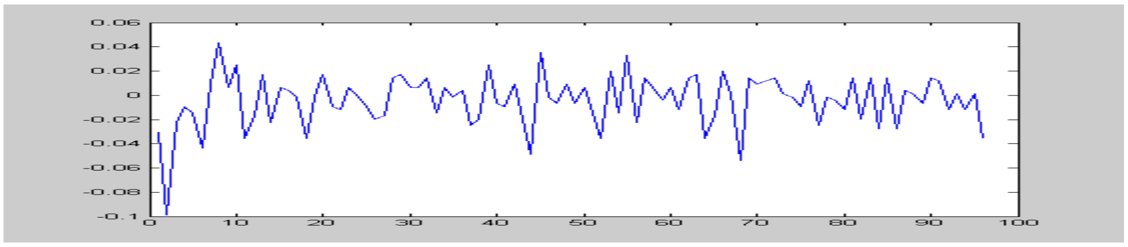


Figure 2.12: The residues of the sequence in figure 2.10.

Figure 2.13 shows the residues of the sequence in figure 2.11, its variance is 0.0041, almost 50 times less than that of the data sequence in figure 2.11.

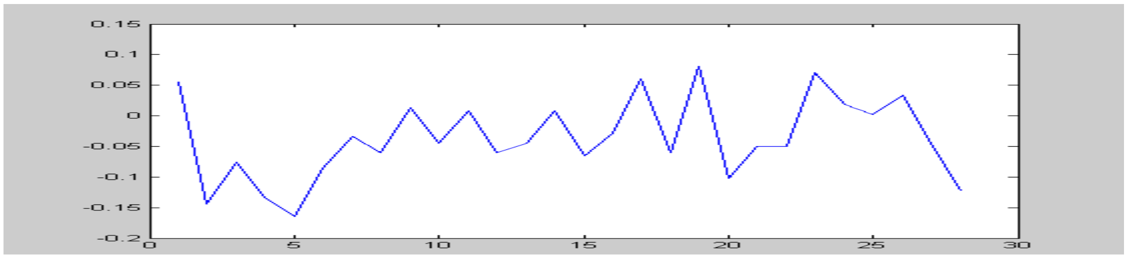


Figure 2.13: The residues of the sequence in figure 2.11.

The variance comparison table 2.1 and 2.2 show a clearer picture of the effect of linear prediction method on reducing the variances of the sequences from stacked wavelet coefficients of ECG 16265 and ECG 12341_02.

ECG 16265	Non-stacked sequence	Stacked sequence	Residues from stacked-sequence
Average variances	0.0711	0.0084	0.0107

Table 2.1 Variances comparisons (ECG 16265).

ECG 12341_02	Non-stacked sequence	Stacked sequence	Residues from stacked-sequence
Average variances	0.1205	0.0669	0.0614

Table 2.2 Variances comparisons (ECG 12341_02).

The result from table 2.1 indicates for some ECG signals residues may not always have smaller variances compared with original stacked data sequence. If the variance from the original data sequence is already small enough, the linear prediction method will not help to further reduce the encoding cost. But for many other ECG signals in the opposite situations, the linear prediction method does generate residues with reduced variance as seen in table 2.2.

Linear prediction method is effective for compressing ECG records, such as ECG 12490_02, ECG 12531_03, and ECG 12936_01. Generally speaking, linearly prediction is effective on ECG signals containing abnormal beats or non-periodic waveforms.

Average variance	Data sequence of stacked coefs. across the beats	Residues from data sequence of stacked coefs. across the beats
ECG 12490_02	0.2238	0.1740
ECG 12531_03	1.7345	1.4806
ECG 12936_01	0.8682	0.2870

Table 2.3 Variances comparisons from testing three objects.

2.2.4 Beat Reconstruction

After passing through the inverse linear prediction filter, the residues are processed, and the reconstructed wavelet coefficients across the beats are re-ordered to get the DWT coefficients for each beat. The reconstructed PAN beats are computed by inverse DWT those coefficients, and the original ECG beats' periods and amplitudes are brought back by re-sampling and scaling the parameters discussed in the section 2.2.1. The detail schematic diagram of the encoding and decoding is provided in [3].

2.3 1-D Embedded Zerotree Wavelet (EZW)

The EZW (Embedded Zerotree Wavelet) algorithm is a simple, yet remarkably effective lossy image compression algorithm. [6]. It was designed by J.M.Shapiro in 1993. It generates a progressive compressed bit stream and is one of the highly attractive data compression algorithms.

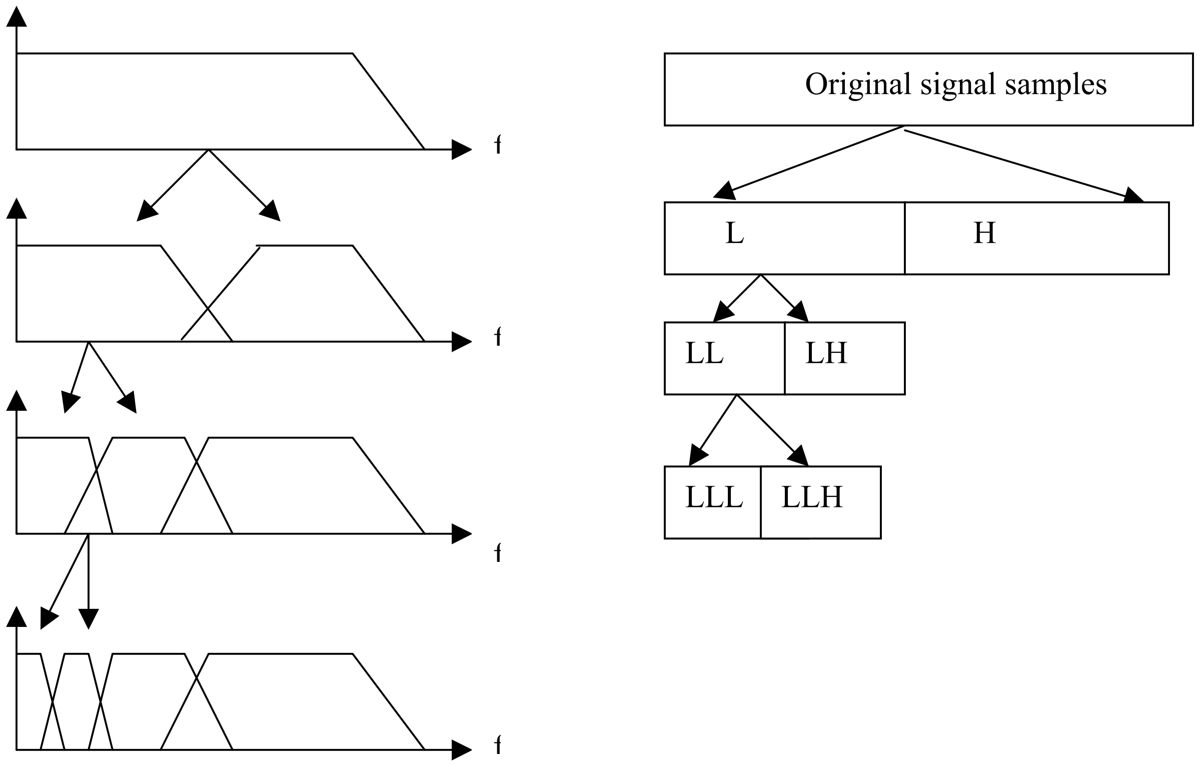


Figure 2.14: 1-D wavelet decomposition

EZW is one of the transform based data compression algorithms. Discrete wavelet transform is the transform used in EZW algorithm, and it transforms the original signal to a joint time-scale domain. The natural ECG signal in general has a low pass spectrum and a high pass spectrum, and the high-pass spectrum contains the smallest details. The low pass spectrum with greater details can be further split into a low pass spectrum and a high pass spectrum. This process can be continued again and again as figure 2.14 shows until it reaches the desired number of decomposition bands.

Considering the data compression efficiency and computational complexity, this thesis uses 3-level decomposition as seen in figure 2.14. The energy in the sub-band decreases, thus for a wide class of ECG signals the wavelet coefficients tend to decrease in absolute magnitude as ECG signals going to the finer scales, or the higher frequency sub-bands.

Because larger coefficients in the coarse scales contain more information and thus are more important, EZW algorithm encodes the larger wavelet coefficients first, and through a zero-tree structure, to encode the less significant coefficients. A zero tree structure is a spanning tree that each parent object has 2 children, and each of the children in turn acts as a parent to have two children, and so on. The zero tree built from the 3-level wavelet discomposed 1-D discrete ECG signal shown in the figure 2.15 is not a strict spanning tree structure, because the spanning only starts from the second layer of the tree. As seen from figure 2.15, the elements at the very top level of the tree are approximate wavelet coefficients from the coarsest scale LLL filtering, followed by its counterpart detail coefficients after the LLH filtering. It starts to span from the coefficients from the LH filtering and the bottom level resides detail coefficients after the most refined filtering.

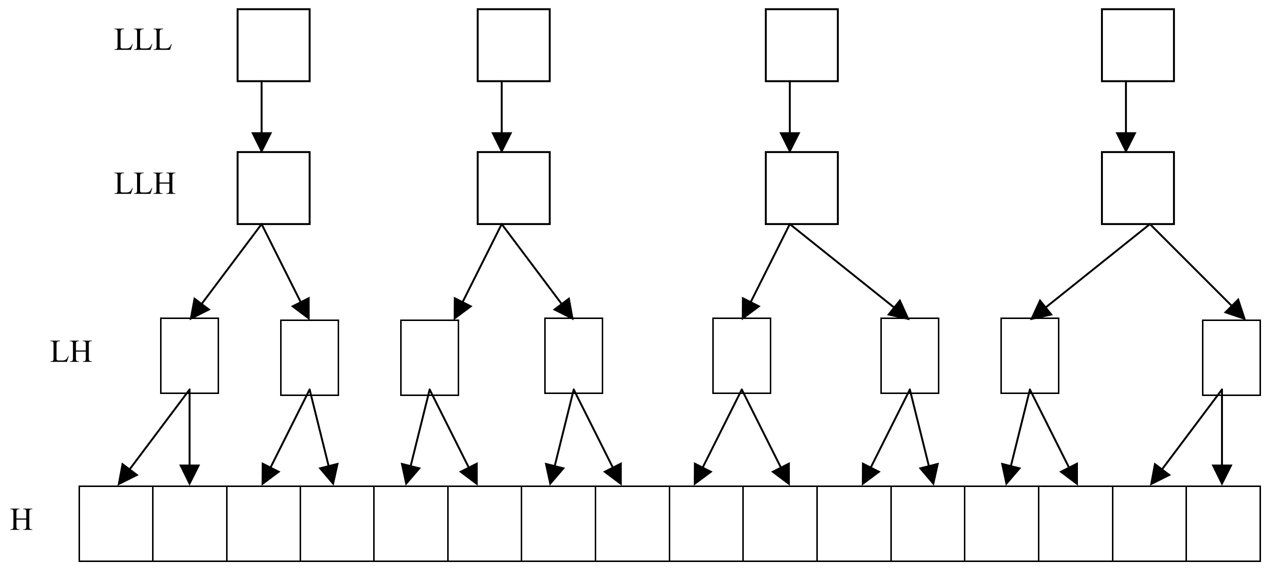


Figure 2.15: Wavelet coefficients in the tree structure.

If the original discrete ECG samples are interpolated into the length of its next 2's power and the interpolated length is *N*, then the length of the wavelet coefficients from H, the most refined scale, is *N/2*. Accordingly, the lengths for LH, LLH and LLL are *N/4*, *N/8* and *N/8* respectively. The total number of coefficients in the spanning tree is *N*. Wavelet decomposition does not increase the number of data to compress.

The EZW algorithm includes two passes, dominant pass and subordinate pass. In the dominant pass, all the coefficient are scanned once by the raster scan sequence shown in figure 2.16, and the coefficients whose absolute values are bigger than the threshold given in (2.9) is coded either as P or N for its positive or negative sign.

$$T_{initial} = 2^{\lfloor \log_2 C_{\max} \rfloor} \tag{2.9}$$

where T_{initial} is the initial value of the threshold, and $C_{\max}$ is the largest absolute value of the wavelet coefficients in the zero tree. For each dominant pass afterwards, the threshold T_i will be half of its old value in the previous pass:

$$T_i = \frac{1}{2} T_{i-1}. \tag{2.10}$$

If a coefficient and all its children's absolute values are less than the threshold, the parent coefficient is coded as ZR, meaning zero-root. Once a coefficient is coded as ZR, the information on all of its children in the tree will be ignored, from which EZW algorithm gains its coding efficiency. On the other hand, if an insignificant parent coefficient has a significant descendant, and even this descendant is not its immediate child but several generations downward, the parent coefficient will be coded as IZ, meaning isolated zero. The EZW algorithm gains its coding advantage from the fact that the probability of ZR's occurrence in the tree is much higher than the IZ's, which is because the magnitudes of the wavelet coefficients decrease with finer scales so that in the tree structure, the coefficients in the top layers have bigger magnitudes than the layers below them.

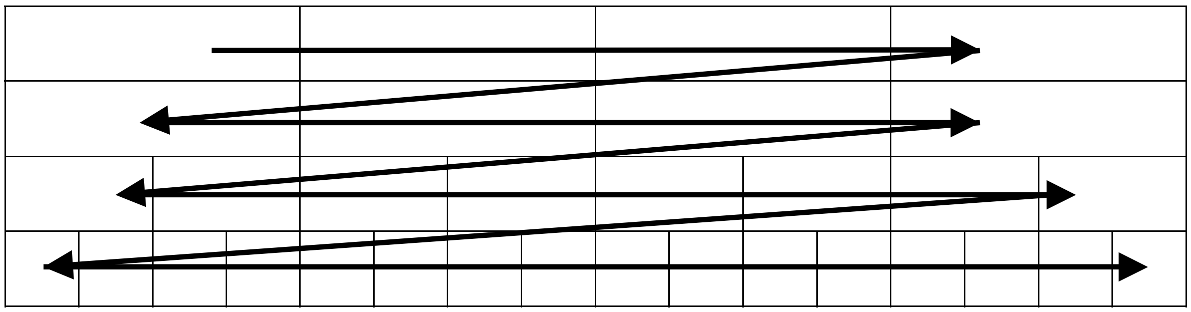


Figure 2.16: Raster scan sequence of wavelet coefficients in the tree.

The performance of the dominant pass will stop only when T_i is less than T_{min}, the minimum coefficient value desired to be transmitted.

The subdominant pass is to refine the absolute values of the significant coefficients. If they are bigger than 1.5 times of the threshold T_i, they are coded as 1, otherwise, they are coded as 0.

This thesis uses 1-D EZW algorithm to compress ECG signals in two ways. One way is to EZW encode the wavelet coefficients from each ECG beats after the *QRS* detections, and the other way is to EZW encode the wavelet coefficients from the entire set of discrete ECG samples without *QRS* detections. The cycle-by-cycle 1-D EZW compression can be implemented in two different ways: setting a constant desired threshold for all the beats, or setting a same number of significant coefficients for all the beats by putting different desired threshold for each beat.

2.3.1 After *QRS* detections

Following the *QRS* detection by the algorithm given in section 2.1, each beat is interpolated to its next 2's power. For example, if the original beat's length is 115, the interpolated length is 128, likewise, 256 for 220, and 512 for 394, etc. Then, 3-level wavelet decomposition is conducted to each beat using the Daubechies-4 function. Because ECG signals are micro-electrical signals, their discrete wavelet transform coefficients have values of 3 to 4 digits after the decimal points, thus, those wavelet coefficients have to be 10000 times amplified and their ceiling integer values are used to apply EZW algorithm.

2.3.1.1 Constant Threshold

A constant minimum threshold is set for wavelets coefficients from every beats loaded in every tree structure. Each beat will have different number of significant wavelet coefficients. The diagram in figure 2.17 tells how to apply the algorithm.

The compression ratio (CR) has for this method has been computed as follows:

$$CR = \frac{K\sum_{i=1}^{N_T} P_i}{N_T\left(b_{EZW} + b_L + b_{Num} + b_{T_{initial}}\right) + b_{T_{\min}}}. \tag{2.11}$$

where K is the number of b/sample in the original signal, P_{i} is the period of ith beat, N_T is the total number of beats, b_{EZW} is the total number of bits from EZW coder for one ECG beat, b_L, b_{Num}, $b_{Tinitial}$ and b_{Tmin} are the number of bits used for transmitting the original length of one beat, the number of significant coefficients kept for each beat, the value of the initial threshold, and the desired minimum threshold for all the beats.

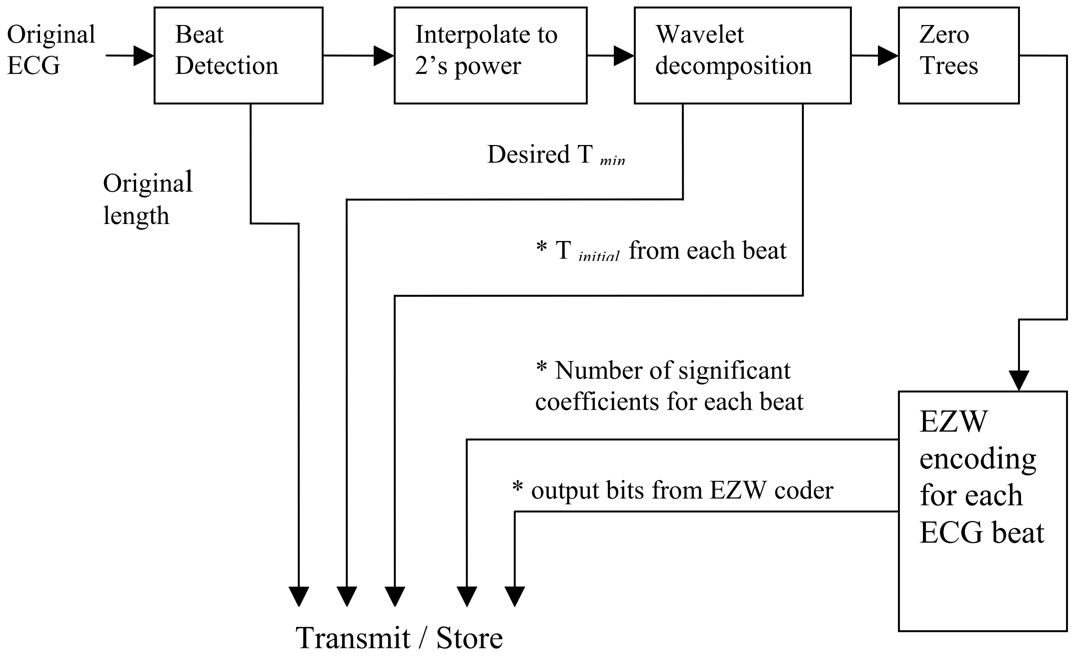

Original ECG
Beat Detection
Interpolate to 2's power
Wavelet decomposition
Zero Trees
Original length
Desired T min
* T initial from each beat
* Number of significant coefficients for each beat
EZW encoding for each ECG beat
* output bits from EZW coder
Transmit / Store

Figure 2.17: Block diagram of the encoder (constant threshold).

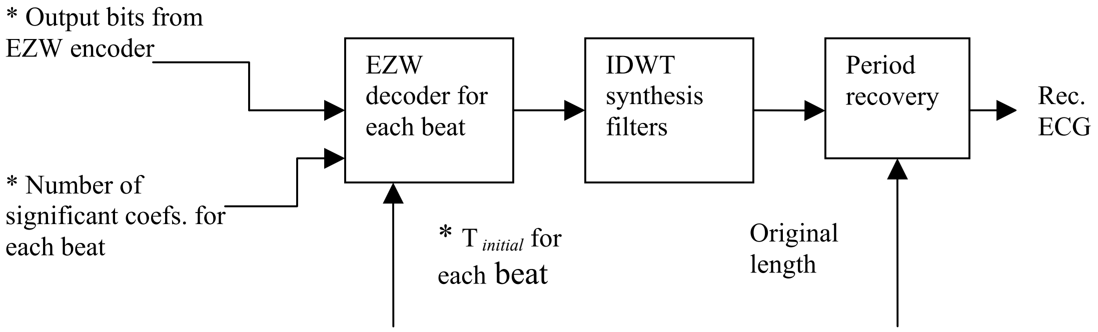

* Output bits from EZW encoder
* Number of significant coefs. for each beat
EZW decoder for each beat
IDWT synthesis filters
Period recovery
Rec. ECG
* T initial for each beat
Original length

Figure 2.18: Block diagram of the decoder (constant threshold).

2.3.1.2 Fixed Number of Coefficients

This way of applying the EZW to each detected beat allows a fixed number of significant coefficients be chosen for each beat. The WBLP algorithm described in second 2.2 also chooses a fixed number of significant coefficients across all the beats, but the difference is in WBLP, they are chosen from every beat at the same index locations, while using EZW, they do not have to be from the same index locations.

To guarantee the same number of significant coefficients be selected for each beat, the desired minimum thresholds for each beat's decomposed wavelet coefficients will be different. Figure 2.19 shows the block diagram of how to implement it, suppose 55 coefficients is the target.

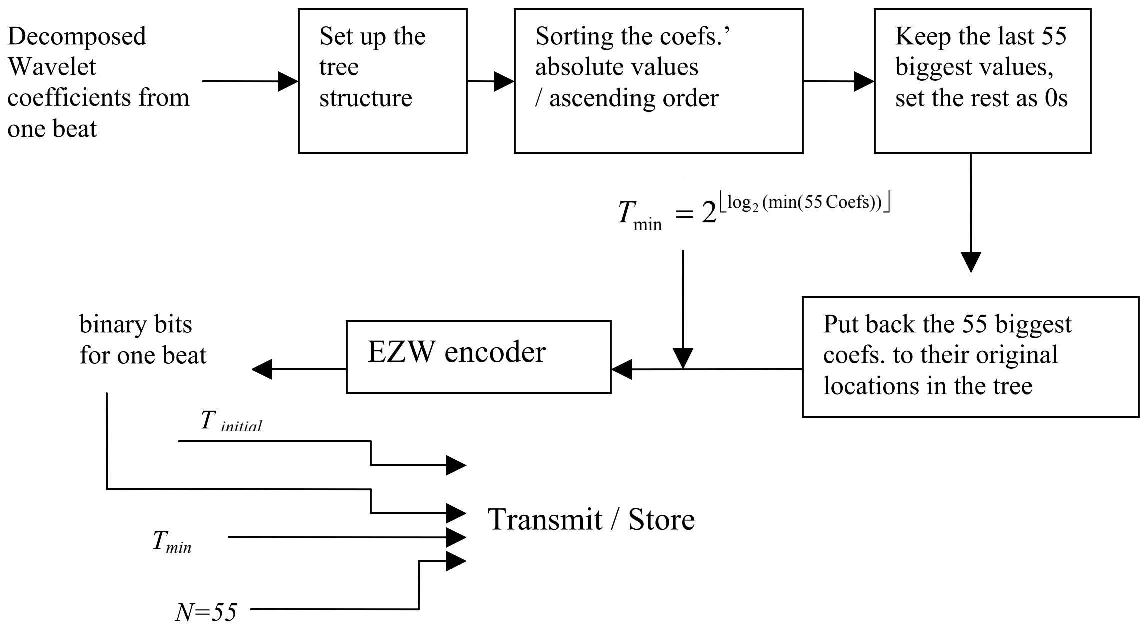

Decomposed Wavelet coefficients from one beat
Set up the tree structure
Sorting the coefs.' absolute values / ascending order
Keep the last 55 biggest values, set the rest as 0s
$T_{\min} = 2^{\lfloor \log_2(\min(55\,\mathrm{Coefs})) \rfloor}$
Put back the 55 biggest coefs. to their original locations in the tree
EZW encoder
binary bits for one beat
$T_{initial}$
T_{min}
$N=55$
Transmit / Store

Figure 2.19 Block diagram for the encoder (fixed number of coefficients).

The decoder diagram is pretty much the same as what shows in figure 2.19, but the desired minimum threshold from each beat is what needed for EZW decoding each beat, instead of the number of coefficients kept from each beat.

The compression ratio (CR) for this method has been computed as follows:

$$CR = \frac{K\sum_{i=1}^{N_T} P_i}{N_T\left(b_{EZW} + b_L + b_{T_{\min}} + b_{T_{initial}}\right) + b_{Num}}. \tag{2.12}$$

where all the symbols in equation 2.12 stand for the same parameters as those in the equation 2.11.

2.3.2 No *QRS* Detections

To avoid the hassles of the *QRS* detections, this method decomposes the entire set of discrete ECG samples, and then applies the EZW coding strategy.

For this method, two wavelet functions are used to represent the ECG signal, one is Daubechies, and the other one is Symlets. Symlets are only the modified Daubechies to increase the symmetries. Section 3 of this thesis will show the different compression effects from employing these two wavelet functions to decompose the original ECGs.

The compression ratio (CR) for this method has been computed as follows:

$$CR = \frac{K \times N}{b_{EZW} + b_{T_{initial}} + b_{T_{\min}} + b_L}. \tag{2.13}$$

where N is the total samples in the discrete ECG samples, and the rest symbols stand for the same parameters appear in the equation 2.11.

2.4 Modified 1-D Embedded Zero-tree Wavelet

While the EZW algorithm can use several dominant passes to progressively select the significant coefficients, and use the subordinate passes corresponding to every dominant pass to refine the selected values, the modified EZW algorithm only uses one dominant pass to aggressively select all the significant coefficients bigger than the desired minimum threshold and code their relative locations within the tree structure. In the subordinate pass, all these significant coefficients are one time uniform encoded. In the modified EZW algorithm, the initial threshold is set as:

$$T_{initial} = T_{desired}. \tag{2.14}$$

where $T_{desired}$ is the desired threshold. Note the $\mathrm{T}_{desired}$ doesn't have to be a power of 2's, and it can be any arbitrary integer.

The step size Δ is the only parameter needed to be decided for the uniform quantizer, and it is given as:

$$\Delta = \frac{2 \times 4\sigma}{2^b}. \tag{2.15}$$

where σ is the standard deviation of the data samples need to be quantized, *b* is the total number of bits used in the quantization.

Likewise, algorithm can be applied to every ECG beat after the *QRS* detections, or directly applied to the entire discrete ECG samples.

2.4.1 After *QRS* detections

Again, it can be performed with one constant desired threshold for all the ECG beats, or setting different initial thresholds to gain a fixed number of significant coefficients for each beat.

2.4.1.1 Constant Threshold

Figure 2.20 shows the diagram of how the algorithm works.

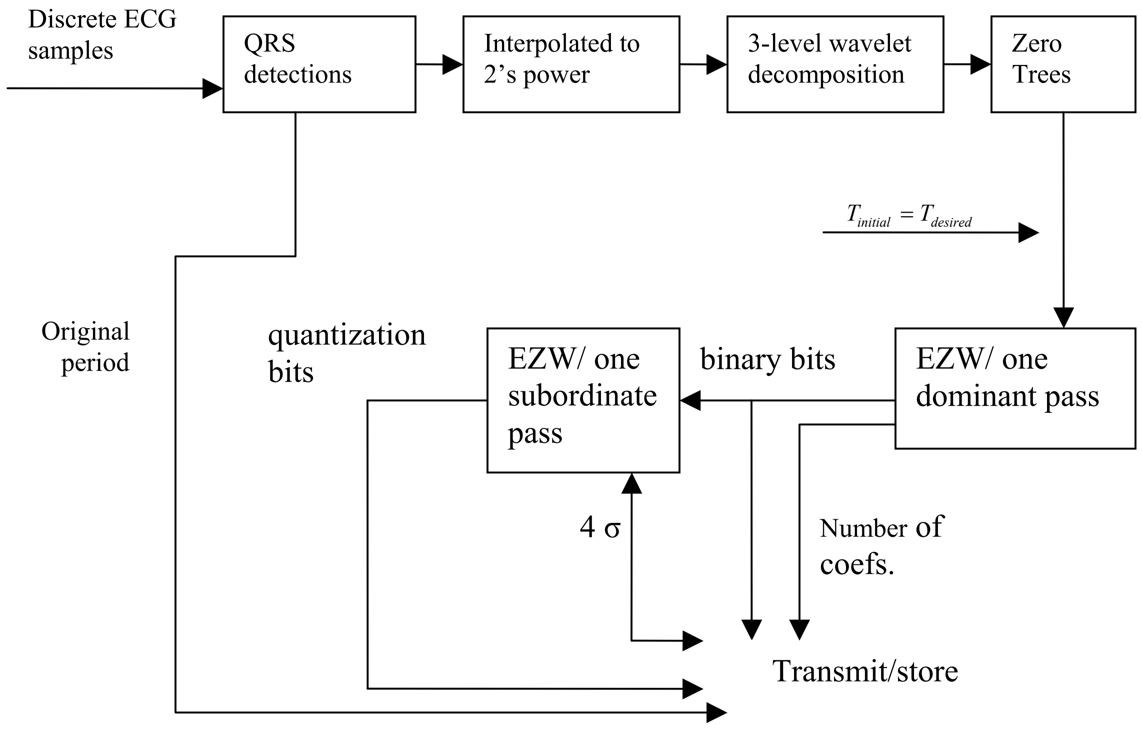


Figure 2.20: Block diagram of the encoder for modified EZW (constant T)

The compression ratio is computed as:

$$CR = \frac{K\sum_{i=1}^{N_T} P_i}{N_T(b_{MEZW} + b_{N_i} + b_L)}. \tag{2.16}$$

where K is the number of b/sample in the original signal, P_i is the period of ith beat, N_T is the total number of beats, b_{MEZW} is the total number of bits output from modified EZW encoder, which includes the number of bits used to encode the relative locations of the significant coefficients in the zero tree structure, and the total bits used for uniform quantize those coefficients. b_T *and* b_L are respectively the number of bits used for encode the initial threshold for all the ECG beats and their original periods.

2.4.1.2 Fixed Number of Wavelet Coefficients

The procedure of defining the initial thresholds is the same that described in section of 2.3.1.2., except the value of initial thresholds do not have to be the power of 2's. After setting the individual initial threshold for every ECG beat, modified EZW algorithm is applied in the same way as that stated in section 2.4.1.2, except that instead of transmitting the number of significant coefficients, it transmits the values of the initial thresholds for every beat.

The compression ratio of this method is computed as:

$$. \qquad CR = \frac{K\sum_{i=1}^{N_T} P_i}{N_T(b_{MEZW} + b_{T_i} + b_L)} \tag{2.17}$$

where b_{T_i} is the number of bits used to encode each initial threshold, and the rest symbols stand for the same parameters as stated in equation 2.16.

2.4.2 No QRS Detections

This is to implement the modified EZW algorithm directly to the entire discrete ECG samples. It does not need QRS detections. The compression ratio of this method is given as:

$$CR = \frac{K \times N}{b_{MEZW} + b_{T_{initial}} + b_L}. \tag{2.18}$$

where N is the total number of discrete ECG samples, and the rest symbols stand for the same parameters appear in (2.17) and (2.13).

Once more, it uses both of the Daubechies-4 wavelet and Symlet-4 wavelet functions to do the 3- level wavelet decomposition.

Chapter 3 Results

This chapter shows the simulation results from the compression algorithms explained in the previous chapter. The compression effectiveness is measured by the compression ratio, bit rate and the reconstruction error. Chapter 2 has given the equations to compute the compression ratio for each algorithm. The bit rate is the average number of bits used to encode one ECG beat. The reconstruction errors are measured as follows:

A. Normalized Root Mean Square Error (NRMSE) [2]

$$NRMSE = \sqrt{\frac{\sum_{i=0}^{N-1}\left[x_0(i) - x_r(i)\right]^2}{\sum_{i=0}^{N-1} x_0^2(i)}}. \tag{3.1}$$

where N is the total number of samples, and $x_0(i)$ and $x_r(i)$ are the *ith* sample of original and reconstructed ECG, respectively.

B. Normalized Maximum Amplitude Error (NMAE) [2]

$$NMAE_i = \frac{\max\left|X_{oi} - X_{ri}\right|}{\max X_{oi} - \min X_{oi}}. \tag{3.2}$$

where $NMAE_i$ is the NMAE for *i*th cycle, and *NMAE* is obtained by averaging over all the cycles.

The discrete ECG samples used for simulation are from MIT-BIH ECG Compression Test Database and from MIT-BIH Normal Sinus Rhythm Database. Most of the ECGs originally are sampled at 250Hz and quantized with 12-b resolution. Some ECGs originally sampled at 125Hz are re-sampled to 250Hz. Totally 12 ECG records are selected to test the algorithm in this thesis, 6 of them are sinus rhythm ECG records:

16265, 18184, 16773, 16272, 17052 and 17453; and the rest are from compression test database: 12431_03, 12936_01, 12247_04, 12621_02, 11950_05 and 11442_03.

This chapter uses a large number of plots to show the simulation results. Of all the figures with subplots, the top, middle and bottom subplots in each figure are respectively the original ECG samples, the reconstructed ECG samples and the reconstruction errors.

3.1 Wavelet Based Linear Prediction

The simulation results show that for some sinus-rhythm ECG signals linear prediction method does not help reducing the normalized root mean reconstruction errors, but does help for compressing ECG signals containing abnormal beats. The reason lies at how the uniform quantizer is designed.

The step size Δ of the uniform quantizer is designed by [5]

$$\Delta = \frac{4\sigma}{M}. \tag{3.1}$$

where σ is the standard deviation of the stacked wavelet coefficients feeding into the uniform quantizer, and M is the number of equally sized intervals. The mean square error (*msqe*) of the quantization noise is [5]

$$msqe = \frac{\Delta^2}{12}. \tag{3.2}$$

substituting (3.1) into (3.2) gives

$$msqe = \frac{4\sigma^2}{3M}. \tag{3.3}$$

As it shows in (3.3), the reconstruction error is proportional to the variance σ^2 when M is constant. When stacked wavelet coefficients from sinus-rhythm ECG beats have smaller variances than those from the DPCM residues, it is useless for DPCM to reduce the reconstruction errors. On the contrary, the stacked wavelet coefficients from abnormal ECG beats have larger variances than those from DPCM residues', thus the overall reconstruction errors are reduced by quantizing data sequences with smaller variances.

Figure 3.1 to figure 3.3 show the compression results for ECG record 16265 at different compression ratios. DPCM is not used in compressing the ECG record 16265.

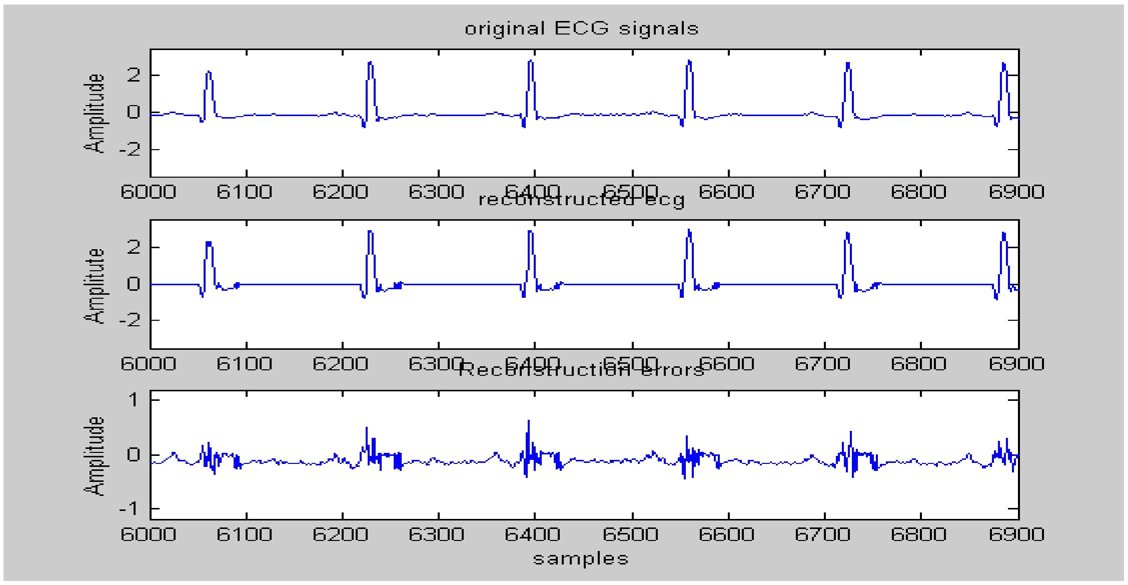


Figure 3.1: ECG 16265 with CR= 22.01, w/o LP

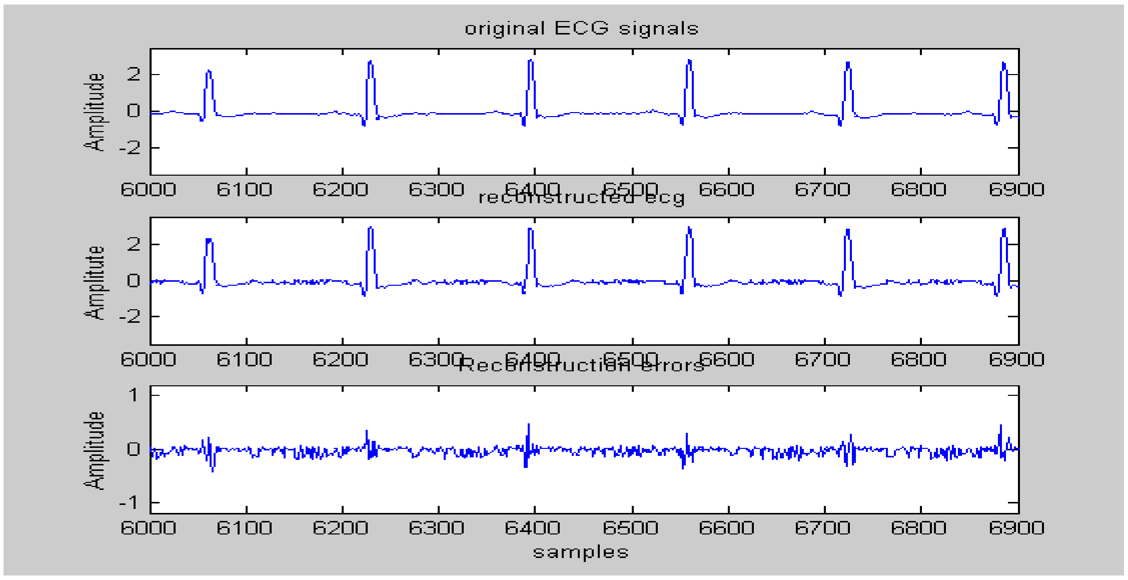


Figure 3.2: ECG 16265 with CR= 8.42, w/o LP

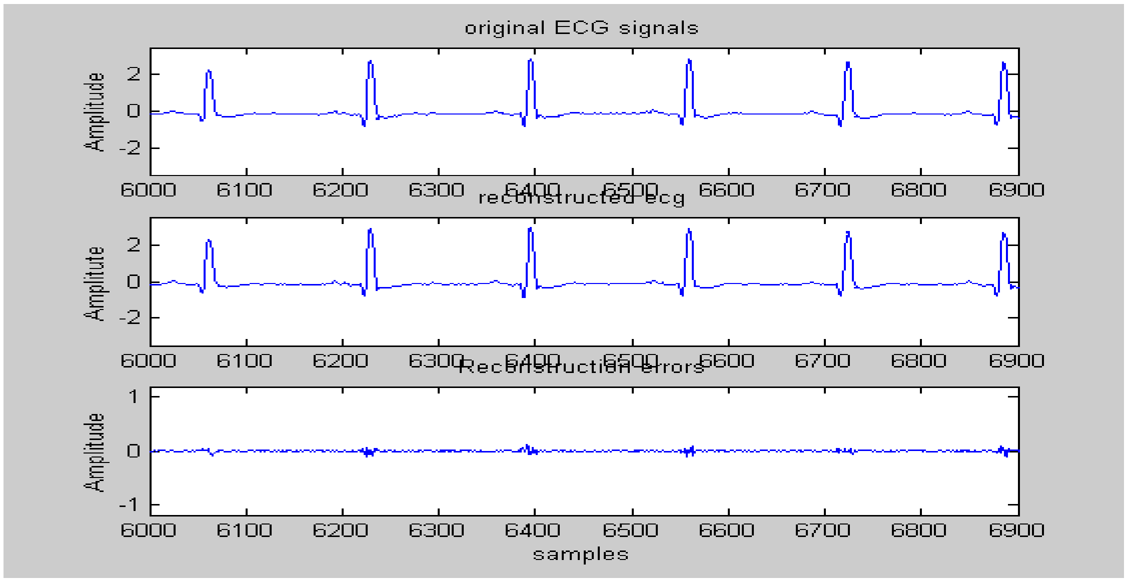


Figure 3.3: ECG 16265 with CR= 2.10, w/o LP

For ECG record 12936_01 which contains abnormal beats, linear prediction plays a good role in reducing the distortion level with the comparable compression ratios as shown from figure 3.4 to figure 3.7. Figure 3.4 shows the result from using LP and keeping 155 coefficients for each beat and quantizing them by 5 bits, the compression ratio is 3.22 and NRMSE is 6.7%. In Figure 3.5, without using LP, though keeping 160 coefficients and quantizing them by 5 bits, the compression ratio is 3.19, and the NRMSE is almost doubled to 13.98%. Figure 3.6 shows the result from keeping 142 coefficients and 4 bits for quantization, and the resulting NRMSE is 13.98%; Figure 3.7 shows the result from keeping 148 coefficients and 4 bits for quantization, and the resulting NRMSE is 27.22%

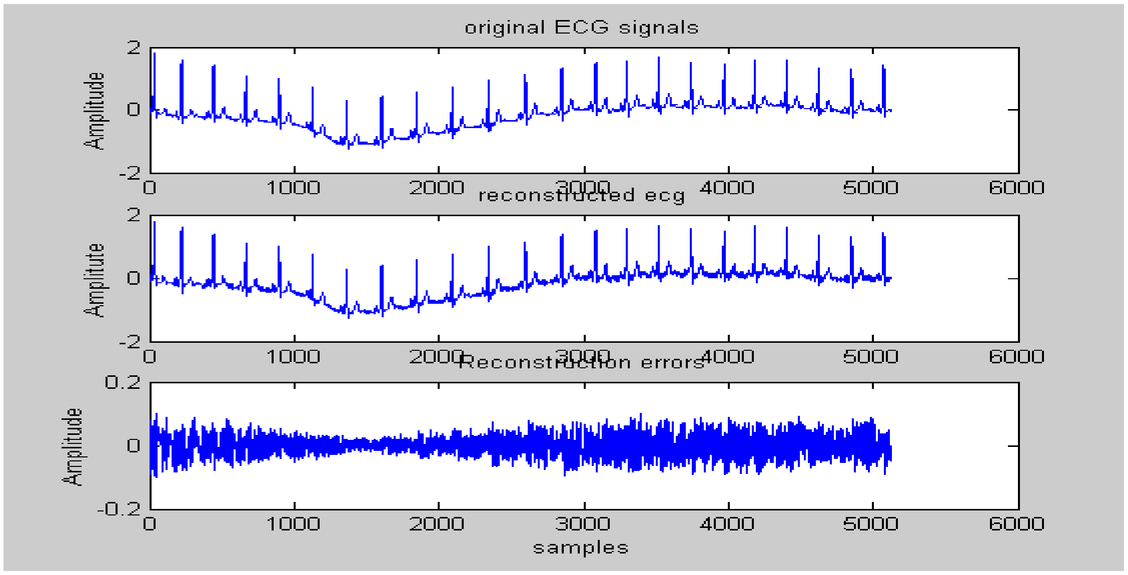


Figure 3.4: ECG 12936_01, CR= 3.22 w/ LP

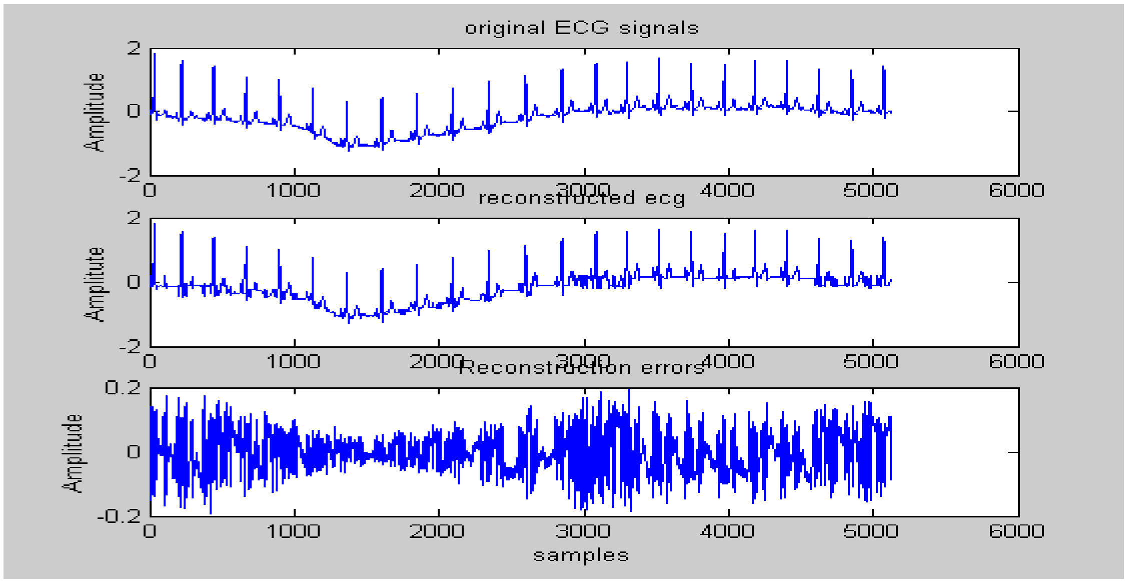


Figure 3.5: ECG 12936_01, CR= 3.19 w/o LP

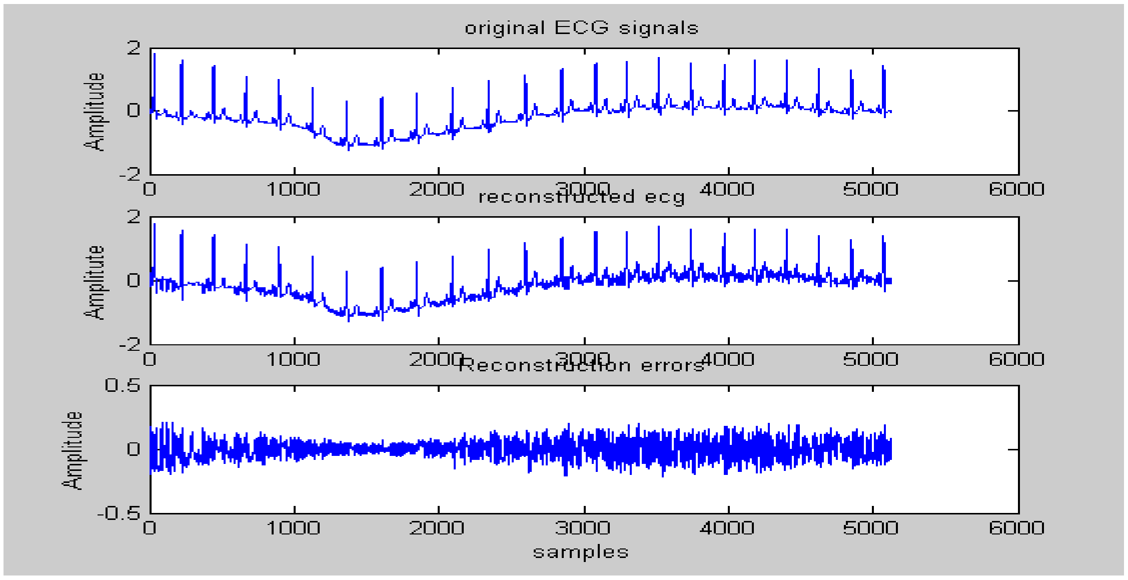


Figure 3.6: ECG 12936_01 with CR= 4.21 w/ LP

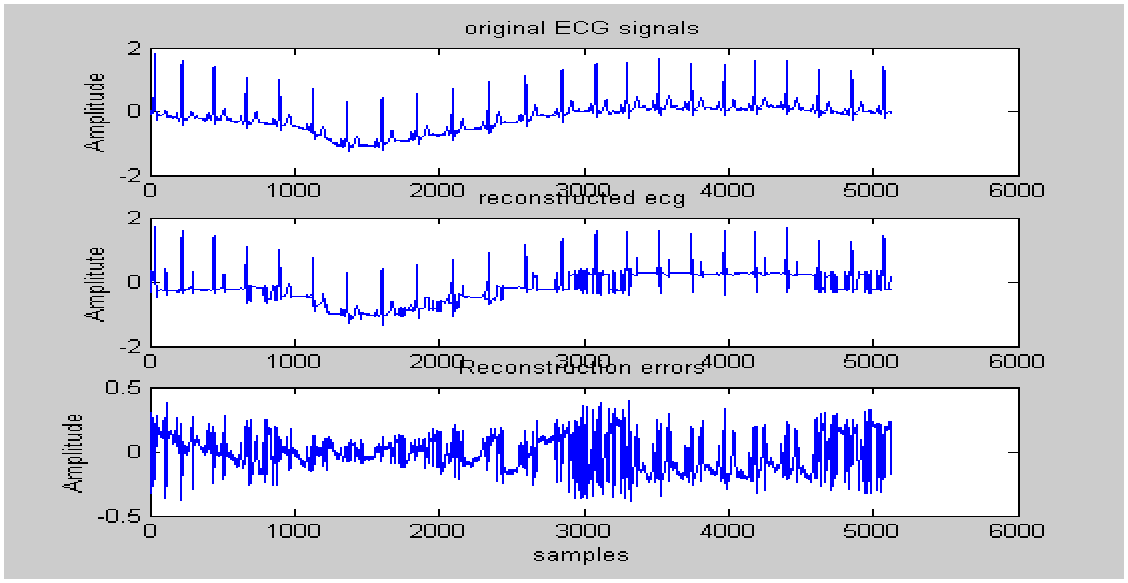


Figure 3.7: ECG 12936_01, CR= 4.21 w/o LP.

On the other hand, the LP method is effective when compressing ECG at high ratios, because to achieve high compression ratios, less quantization bits can be used. Thus, the residues from DPCM with smaller variances can be restored better. Figure 3.8 shows the result from using second order LP and keeping 180 coefficients for each beat and quantizing them by 2 bits, the compression ratio is 6.34% and NRMSE is 59%. In Figure 3.9, without using LP, though keeping 195 coefficients and quantizing them by 2 bits, the compression ratio is 6.34%, and the NRMSE is 68%. Figure 310 shows the result from keeping 250 coefficients and 3 bits for quantization, and the resulting NRMSE is 26.74%; Figure 3.11 shows the result without DPCM from keeping 250 coefficients and 3 bits for quantization, and the resulting NRMSE is 33.92%

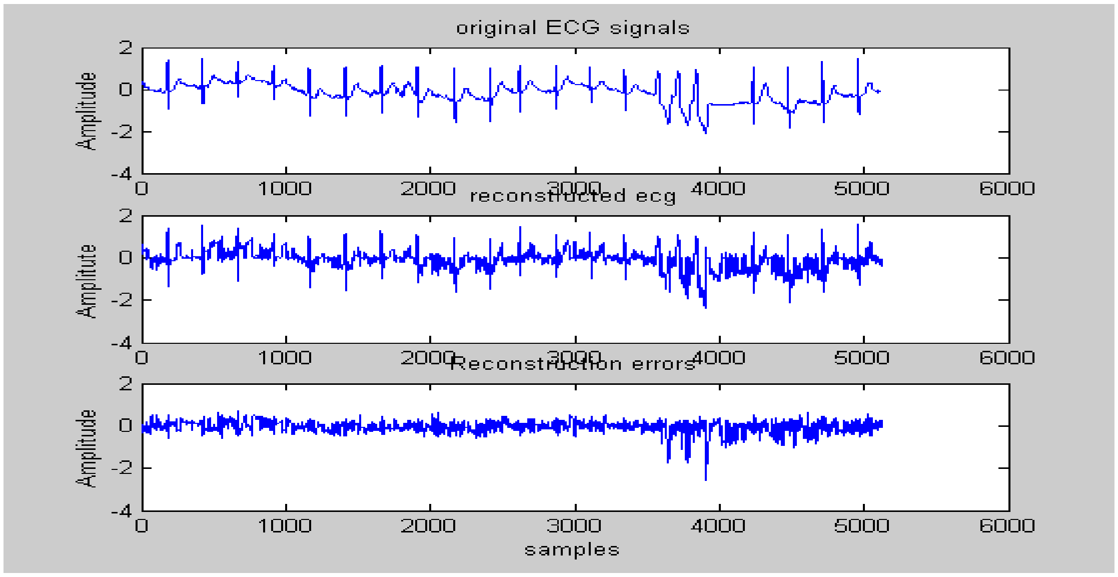


Figure 3.8: ECG record 12431_03, CR= 6.34, w/LP

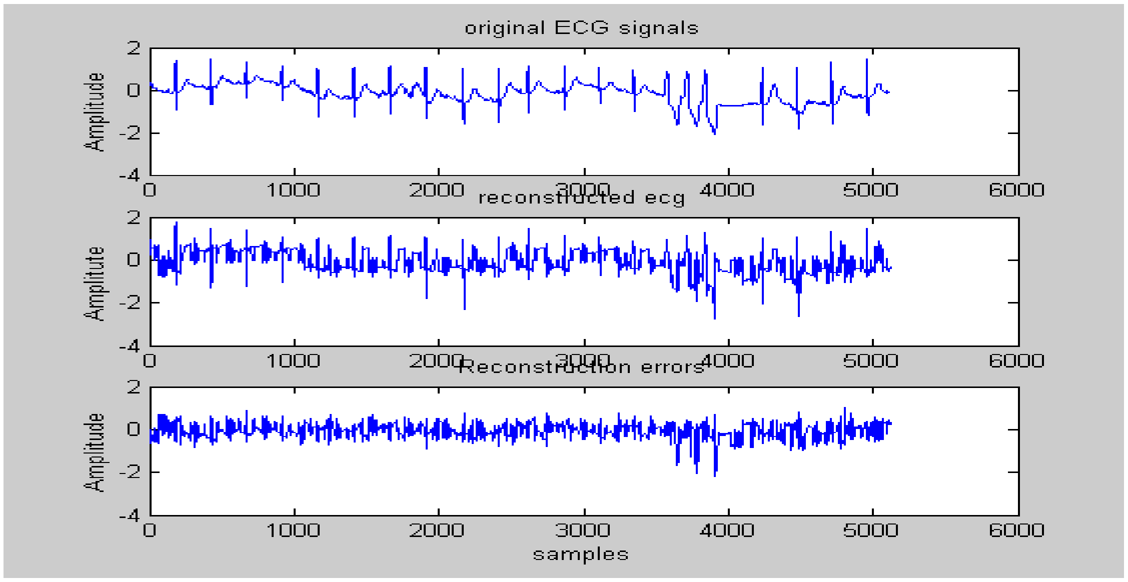


Figure 3.9: ECG 12431_03 with CR= 6.34, w/o LP

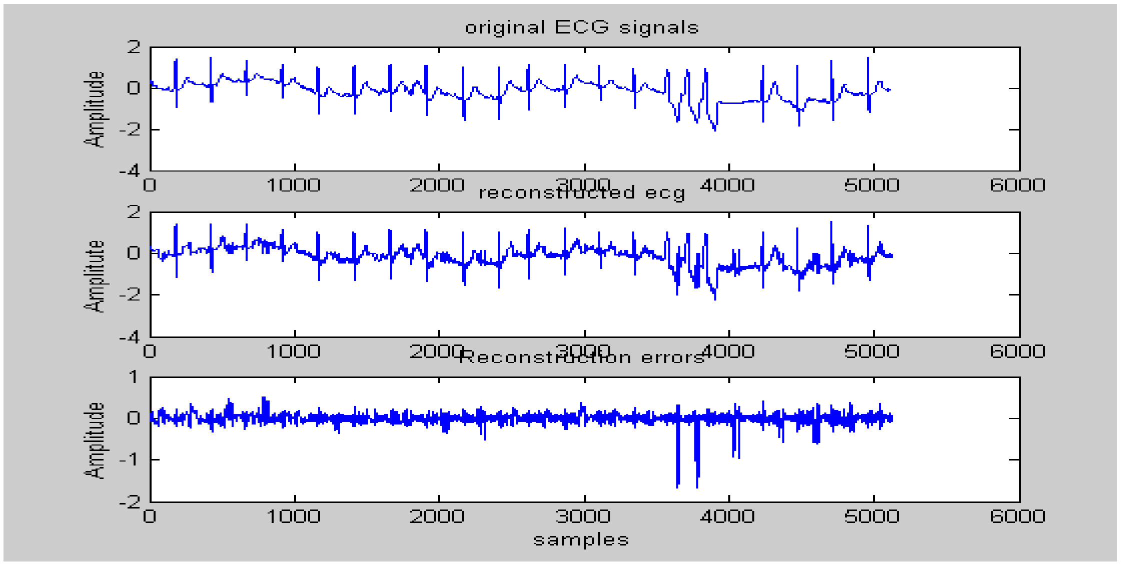


Figure 3.10: ECG 12431_03 with CR= 3.26, w/ LP

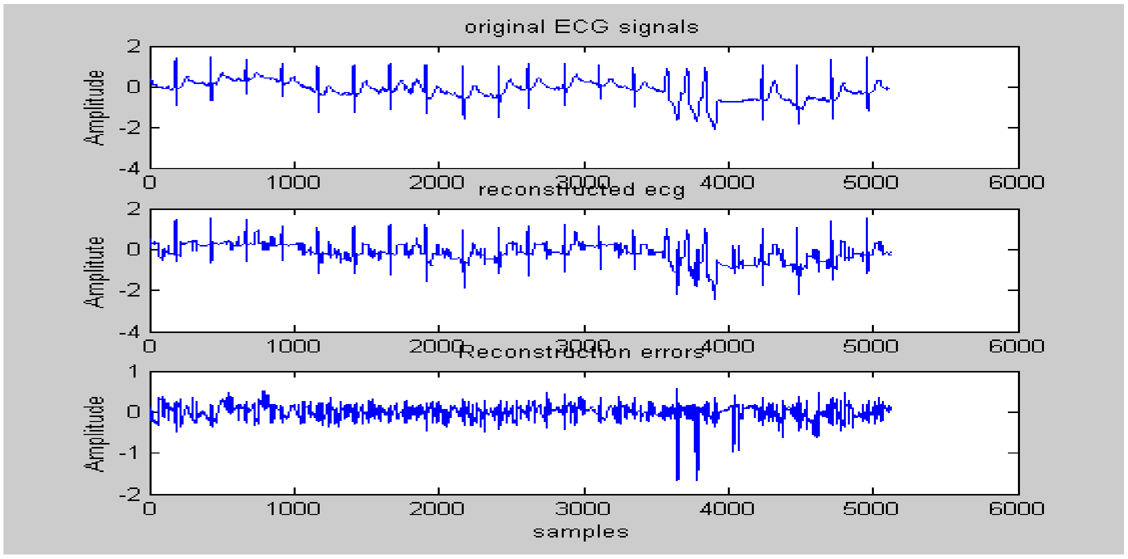


Figure 3.11: ECG 12431_03 with CR= 3.54, w/o LP

Though it is effective for WBLP to compress sinus-rhythm ECG as seen from table 3.1 and table 3.2, which list respectively the best and the average compression results, the same good results do not appear at compressing the ECG signals with abnormal beats. Table 3.3 shows the best compression result using WBLP to compress the abnormal ECG signal, but when the compression ratio only reaches 5.31, the NRMSE is as big as 22.82. Table 3.4 shows the average testing results.

CR	bit-rate	NRMSE	NMAE	Coefs. kept	Qnan. Bits
1.66	1606	1.09	0.65	250	6
2.77	963	4.27	2.45	230	4
3.8	702	8.46	4.83	220	3
4.62	577	8.5	4.83	180	3
5.52	483	8.6	4.91	150	3
6.25	427	11.58	5.46	132	2
8.94	298	18.16	10.89	130	2
10.06	265	22.25	6.56	80	3
13.2	202	23.81	10.3	88	2
16.26	164	27.38	10.29	70	2
26.5	100	35.3	11	70	2
35.43	75	39.73	11.2	28	2

Table 3.1: Best compr. results using WBLP on sinus-rhythm ECG (17052).

CR	Bit Rate	NRMSE	NMAE
2.53	1090	6.08	3.78
5.42	447	13.84	6.68
8.91	261	22.95	10.12
12.44	202	31.81	11.35
16.49	151	38.17	12.34
23.48	109	42.72	12.30

Table 3.2: Average compr. results using WBLP on 6 sinus-rhythm objects.

CR	BIT-RATE	NRMSE	NMAE	Coefs. Kept	Quan. Bits
1.6	1744	2.97	2.04	250	6
2.94	947	5.86	4	160	5
3.61	773	11.87	8.06	160	4
4.17	668	11.99	8.8	142	4
5.31	525	22.82	16.1	140	3
6.59	424	38.47	40.94	112	3
7.34	380	46.45	43.94	100	3

Table 3.3: Best compr. results using WBLP on abnormal ECG (12936_01)

CR	Bit Rate	NRMSE	NMAE
1.69	1483	16.87	11.69
2.43	1141	18.95	13.42
3.35	981	29.86	20.68
4.26	647	36.52	20.05
5.36	433	39.99	19.86
6.52	443	50.54	25.49
8.43	261	54.14	24.23
10.92	271	64.78	26.71

Table 3.4: Average compr. results using WBLP on 6 abnormal objects.

.

3.2 1-D EZW Algorithm

The focus of this thesis is to implement the EZW algorithm on compressing discrete ECG signals, and to compare the results with those getting from WBLP.

As described in section 2.3 and 2.4, 1-D EZW algorithm is conducted after different preprocessing. The first is after the QRS detection and with constant threshold for all the beats, the second is after QRS detections and choosing fixed number of coefficients for all the beats, and the third is without QRS detection and directly apply EZW to the entire ECG discrete samples.

After testing on the 6 sets of sinus-rhythm ECG and 6 sets of abnormal ECG, the simulation results show that 1-D EZW after QRS detections do not outperform

WBLP method in compressing sinus-rhythm ECG signals, only 1-D EZW without QRS detection beats WBLP at higher compression ratios in regarding to NRMSE, but not NMAE. Figure 3.12 and figure 3.13 show these results.

However, the advantage of using EZW lies in compressing abnormal ECG beats as seen in both figure 3.14 and figure 3.15. The plot in figure 3.14 clearly shows that EZW without QRS detections generates smallest reconstruction distortions measured by NRMSE, followed by EZW with QRS detections but with constant threshold for all the beats, which generate comparable NRMSE at lower compression ratios.

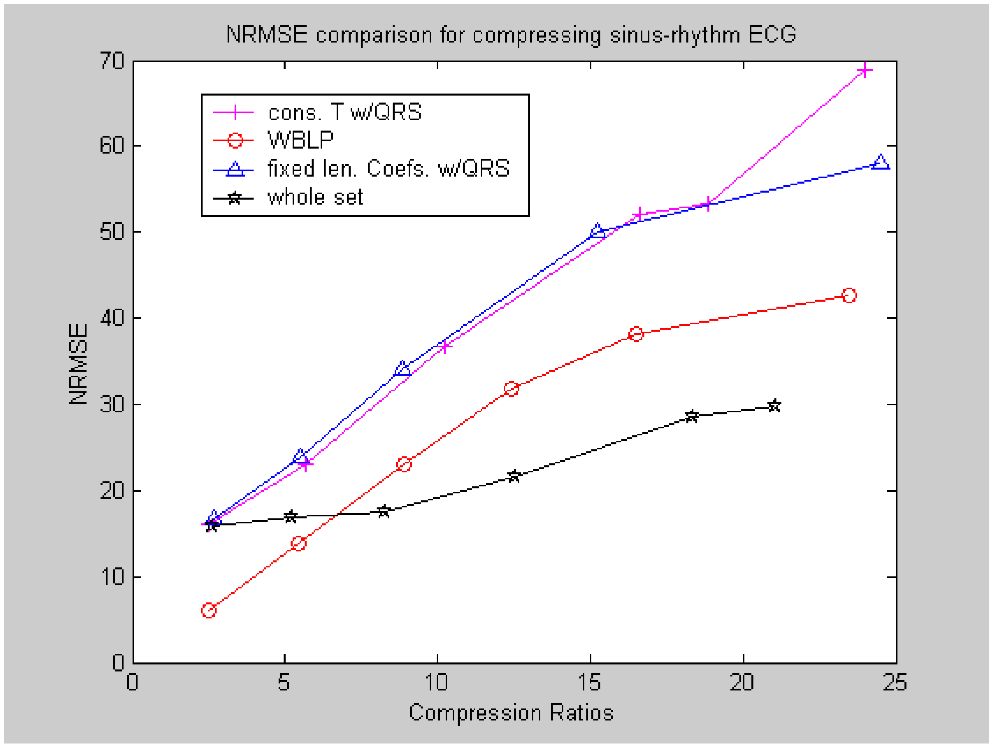


Figure 3.12: NRMSE comparisons for compressing sinus-rhythm ECG.

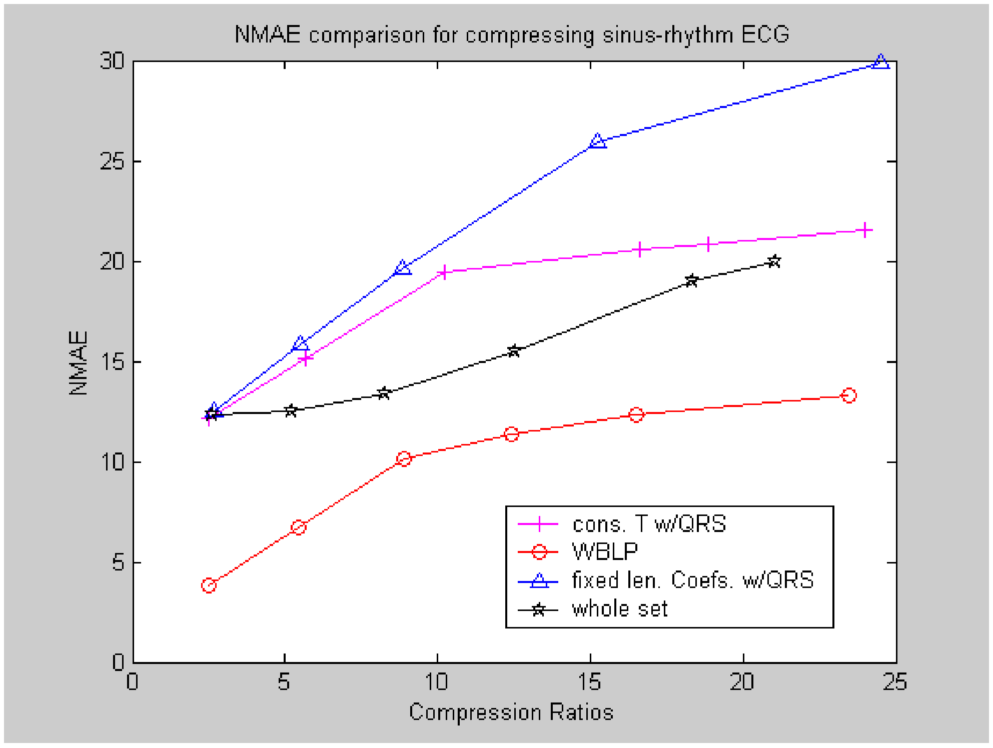


Figure 3.13: NMAE comparisons for compressing sinus-rhythm ECG.

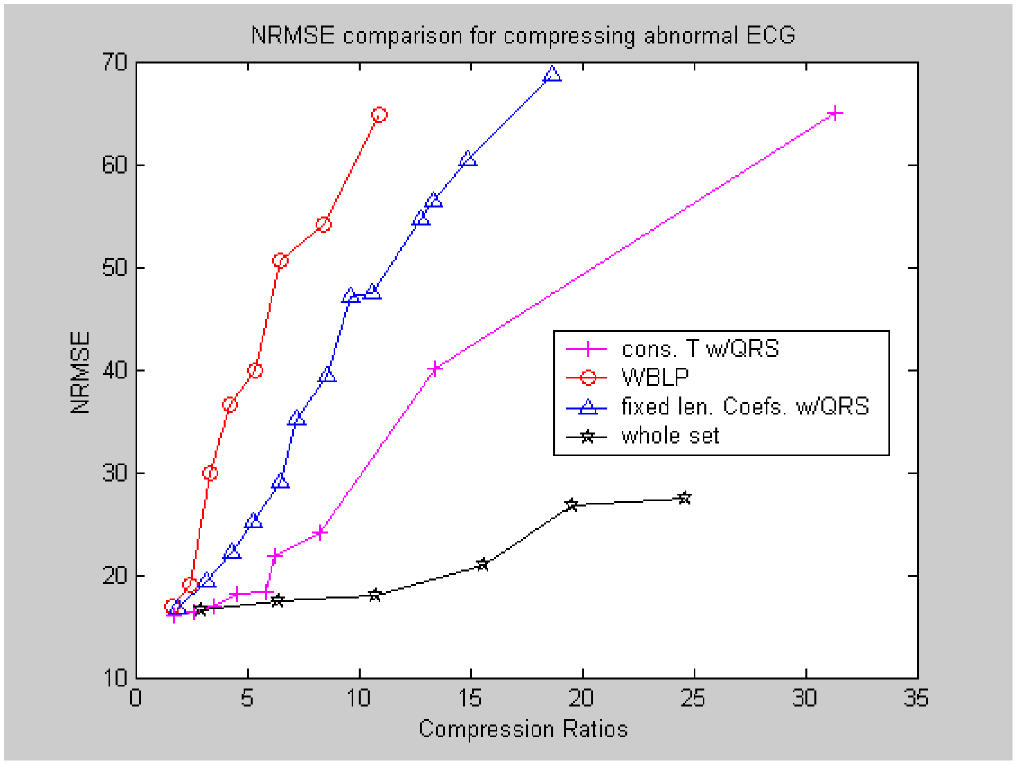


Figure 3.14: NRMSE comparisons for compressing abnormal ECG.

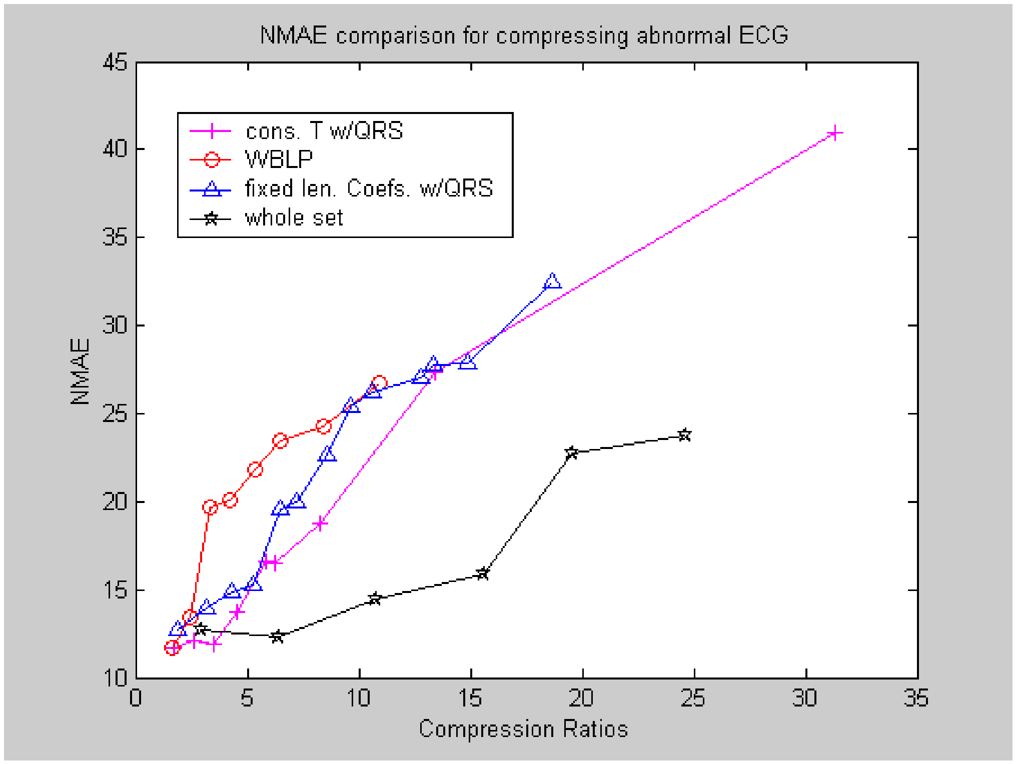


Figure 3.15: NMAE comparisons for compressing abnormal ECG.

The success of 1-D EZW to compress ECG 12431_03 is shown in figure 3.16,

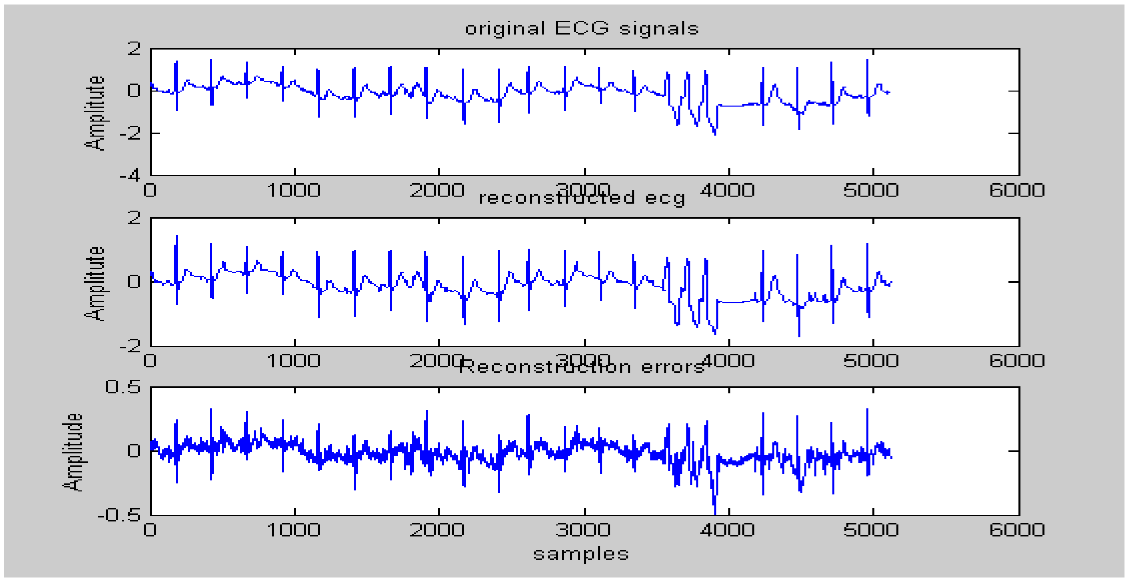


Figure 3.16: EZW w/QRS and constant T on ECG 12431_03, CR=5.88

The second subplot in figure 3.16 demonstrated much better recovery of ECG than that shown in figure 3.10. The NRMSE is only 16.98% with compression ratio CR being 5.88. In figure 3.10, using WBLP, the NRMSE is as high as 26.74% with compression ratio only being 3.26.

The result of compressing ECG 12936_01 using EZW without preprocessing as seen in figure 3.17 is another example to show the success of EZW over WBLP. The compression ratio is 11.98, and the NRMSE and NMAE are respectively 18.99% and 16.23%, versus the simulation result from using WBLP as seen in figure 3.18, the compression ratio is only 5.93 but the corresponding NRMSE and NMAE already hit 24.79% and 24.98%.

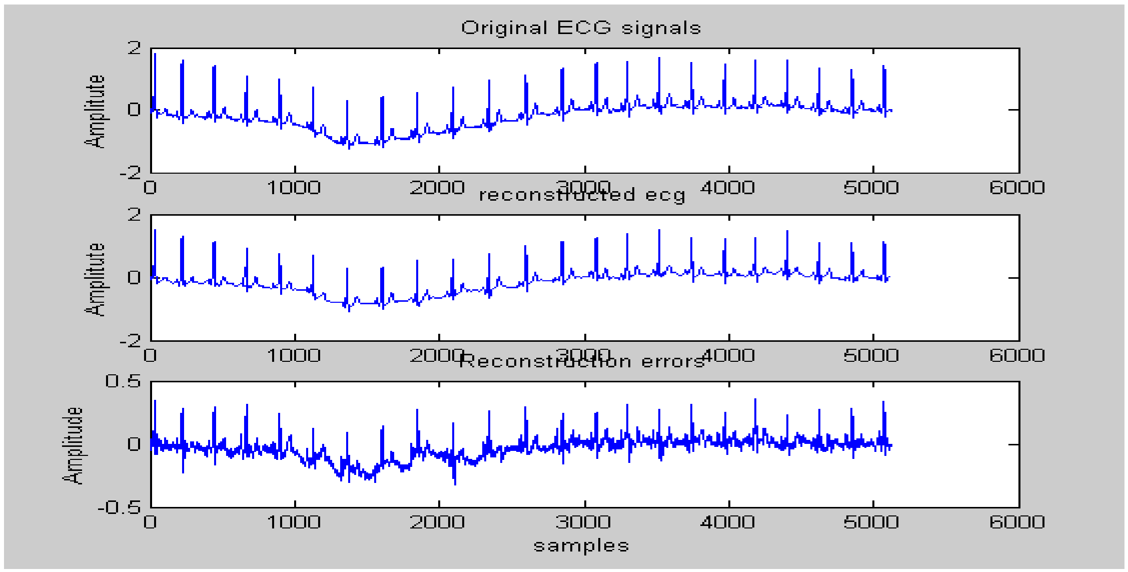


Figure 3.17: Direct EZW on ECG 12936_01, CR=11.98

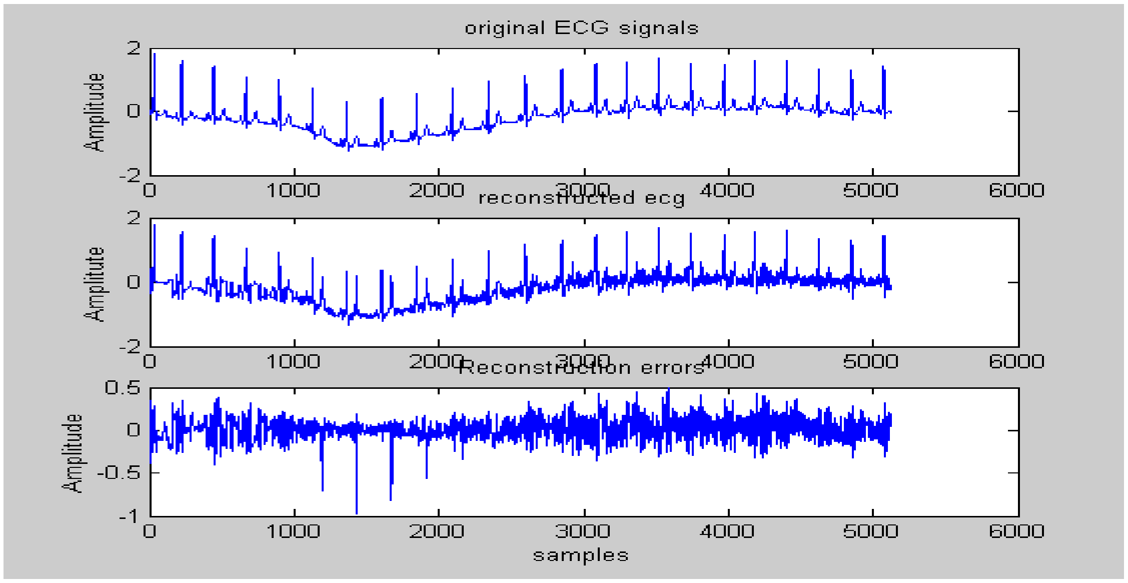


Figure 3.18: Compression ECG 12936_01 by WBLP, CR=5.93.

When using EZW to directly compress the entire set of discrete ECG signals, slightly better results are from decomposing ECG via Symlets wavelet function than via Daubechies. While in applying EZW after delineating beats, employing these two wavelet functions do not generate many differences.

Though directly applying EZW generates good results in compressing ECG, its possible formidable computational complexity at the worst case, as discussed in chapter 4, will prohibit it from being considered if compressing time is a concern.

3.3 Modified 1-D EZW Algorithm

MEZW is an aggressive compression algorithm. Section 2.4 describes how to implement this algorithm. This section shows the simulation results.

Owing to its aggressive nature, it achieves best compression results especially for compressing abnormal ECG beats.

In compressing ECG 12431_03, as seen from figure 3.19, when CR is 25.22, its NRMSE is only 24.39% by setting the threshold as 7500 and using 4 bits to do the quantization.

Using WBLP to compress ECG 11442_03, as seen from figure 3.20, when the CR is only 2.45, the NRMSE and NMAE respectively are as high as 33.13% and 35.16%, while using MEZW with *QRS* detection and constant threshold for all the beats, the NRMSE and NMAE are respectively only 14.39% and 16.20% when CR reaching 7.60 as seen in figure 3.21.

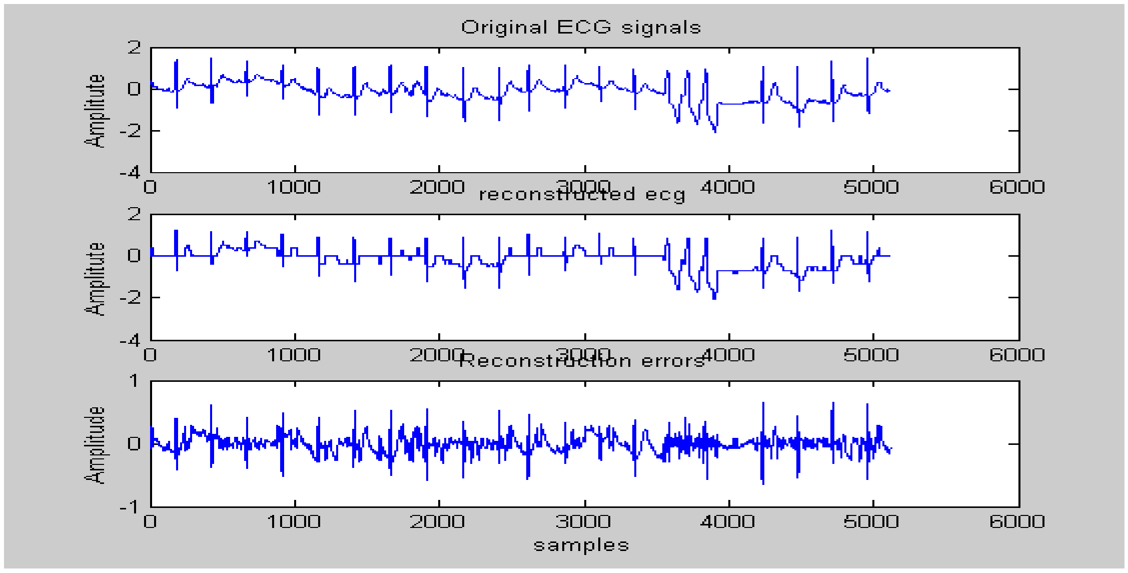


Figure 3.19: MEZW w/o *QRS* on ECG 12431_03. CR=25.22.

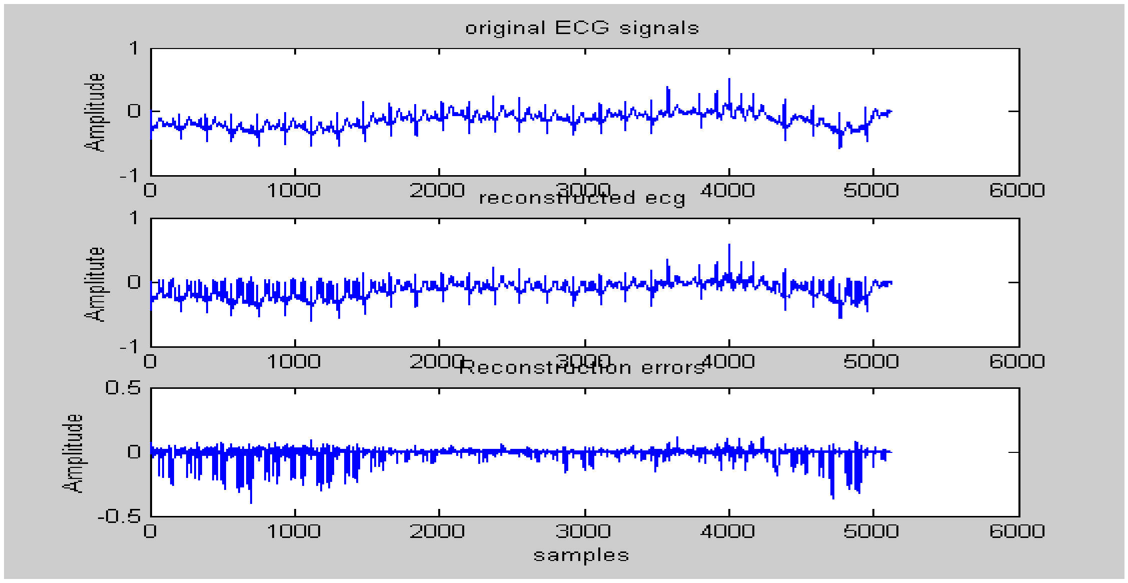


Figure 3.20: WBLP on ECG 11442_03, CR = 2.45.

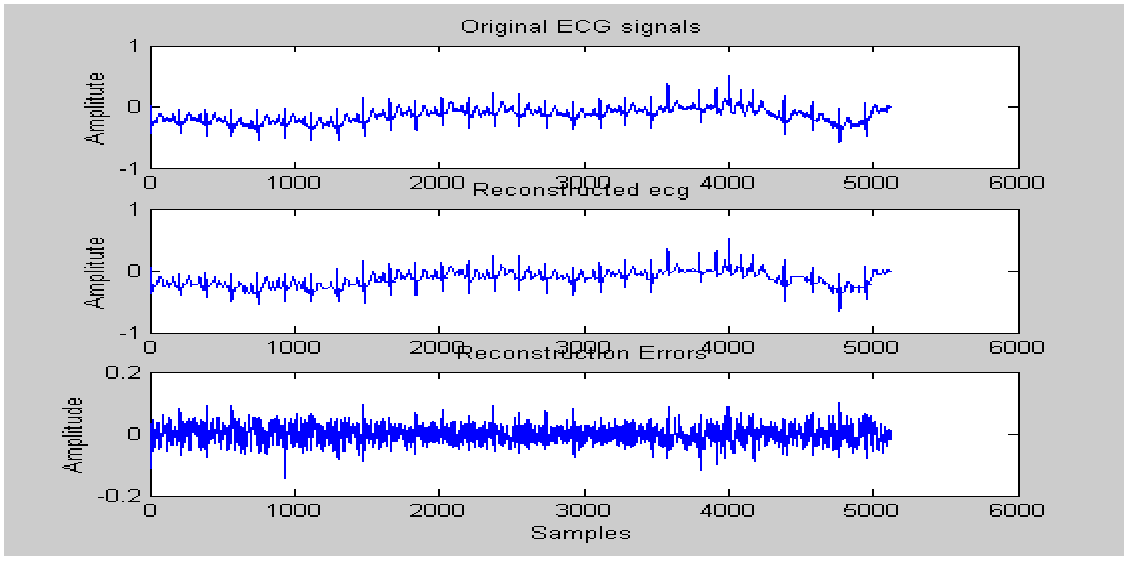
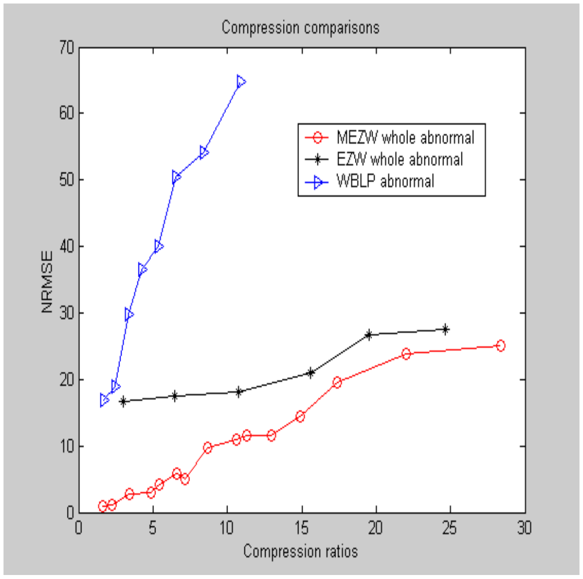


Figure 3.21 MEZW with QRS and cons. T on ECG 11442_03, CR=7.60.

MEZW w/o *QRS* detections outperforms EZW w/o *QRS* and WBLP in compressing both abnormal ECG beats and sinus-rhythm ECG beats as seen in figure 3.22 and figure 3.23 respectively.



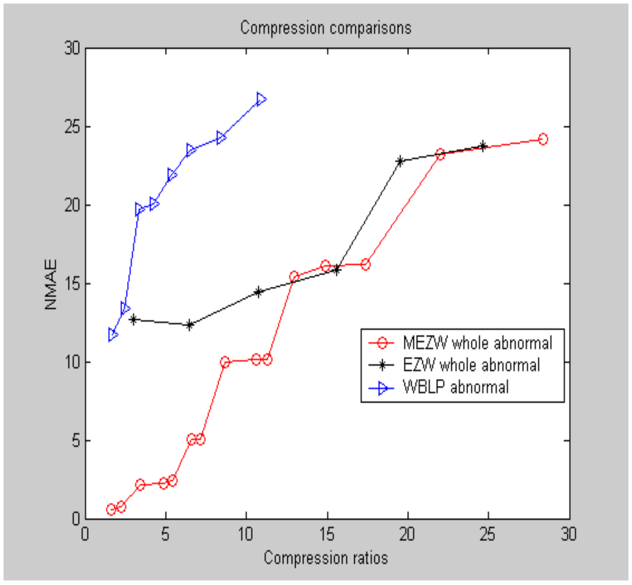
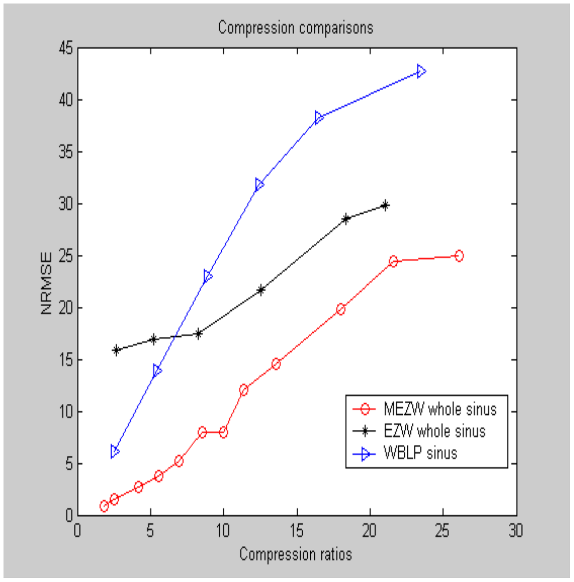


Figure 3.22: Comparisons between three algorithms on ECG abnormal beats.



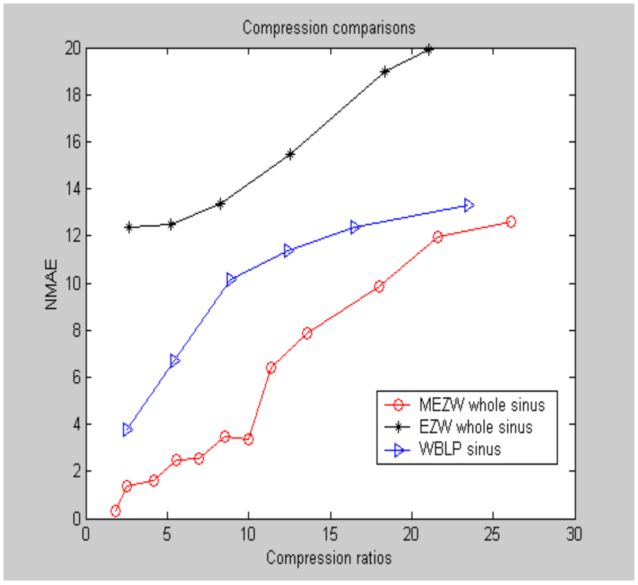


Figure 3.23: Comparisons between three algorithms on ECG sinus-rhythm beats.

MEZW w/o *QRS* detections outperforms MEZW with *QRS* detections on both ECG signals with abnormal beats and sinus-rhythm ECG signals, as seen in figure 3.24 and figure 3.25 respectively.

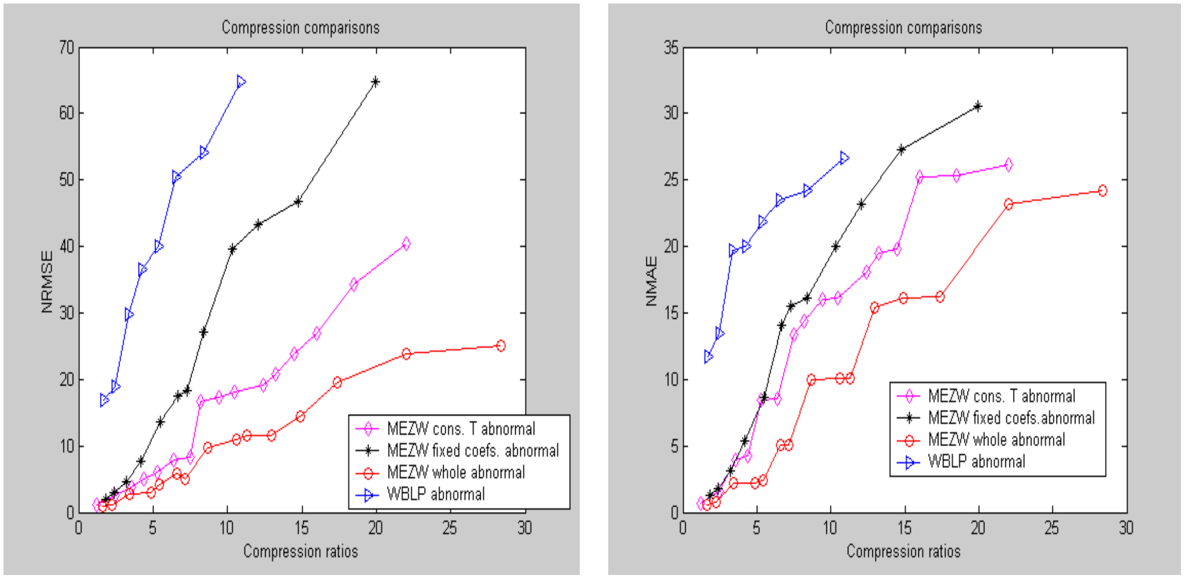


Figure 3.24: Comparisons between four algorithms on abnormal ECG.

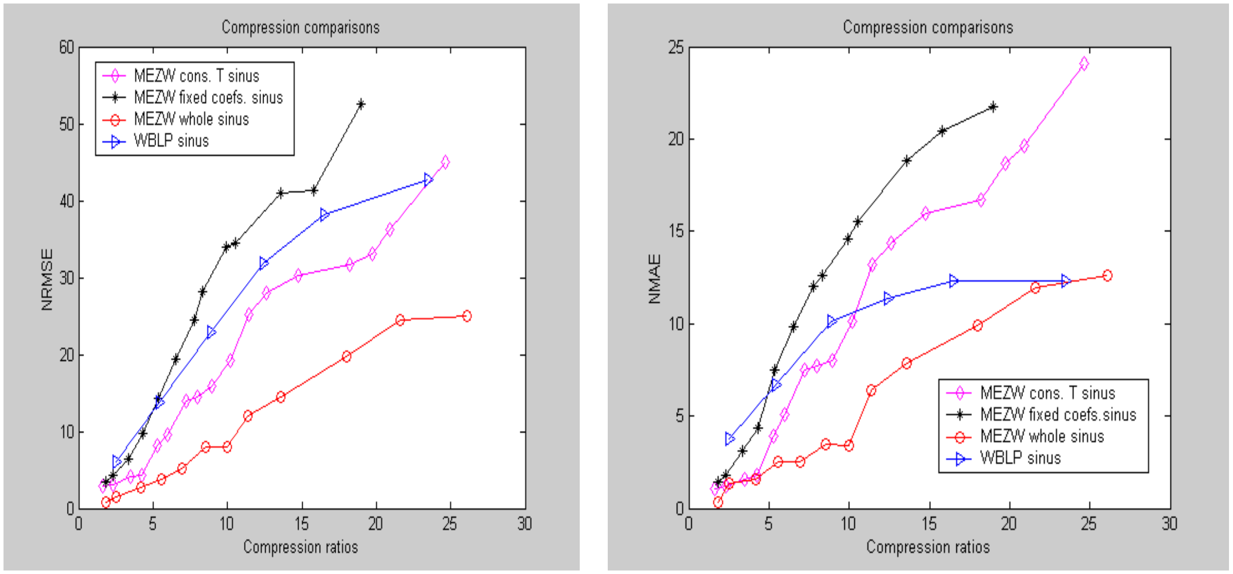


Figure 3.25: Comparisons between four algorithms on sinus rhythm ECG.

Though MEZW without *QRS* detections is the best, it has higher computational complexity than those from MEZW with *QRS* detections as discussed in chapter 4. The next best is MEZW with *QRS* detection and constant T across all the beats as it breaks down the long EZW discrete signals into every ECG beats, thus breaking down the complexity. Between EZW with *QRS* detections and constant T and MEZW with *QRS* detections and constant T, the later one has a better performance as seen in both figure 3.26 and figure 3.27.

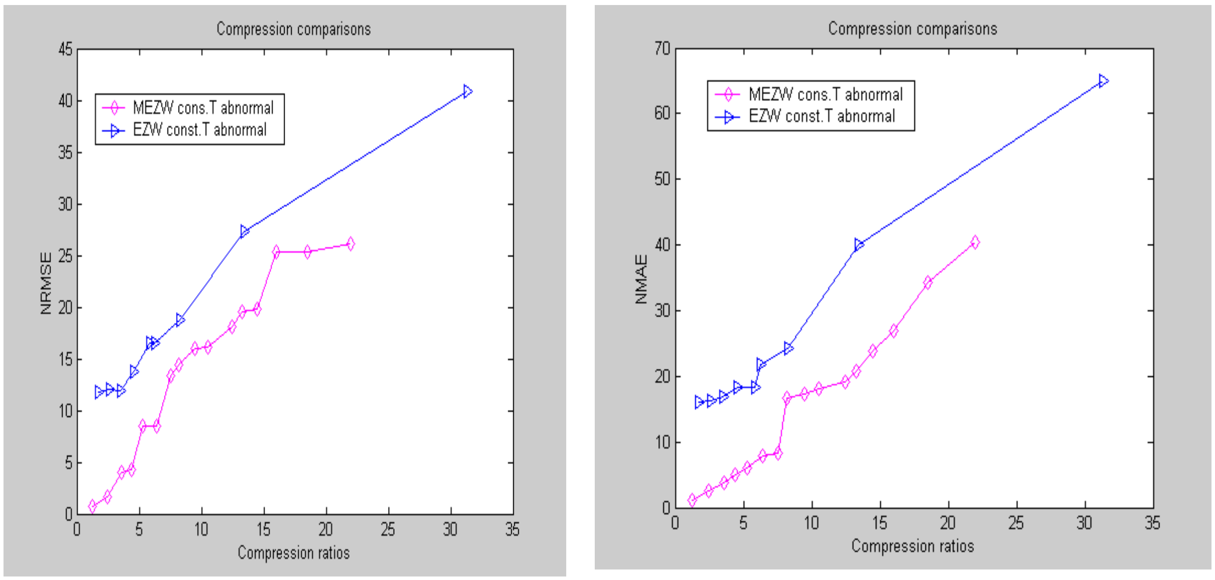


Figure 3.26: Comparisons between two algorithms on abnormal ECG beats.

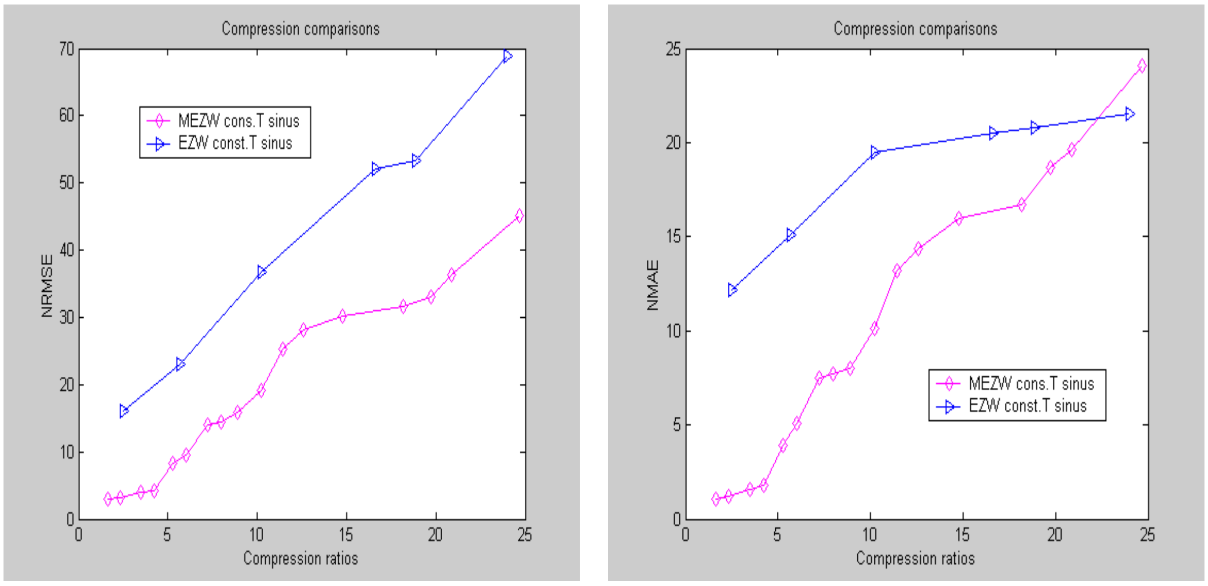


Figure 3.27: Comparisons between two algorithms on sinus-rhythm ECG beats.

Chapter 4 Complexity

4.1 Wavelet Based Linear Prediction

Encoding computational complexity of the WBLP method is as following:

A. Discussions

Suppose totally there are $\frac{N}{R}$ ECG beats, *N* is length of re-sampled ECG discrete samples, and *N* is 2's power. *R* is the length each original beat re-sampled to, in this thesis, *R* is set as 256. In order to find a fixed set of coefficient positions across all the beats, it needs to first sort each beat's wavelet coefficients, and the total number of coefficients from the high pass filter and low pass filter is also *R*. Using the merge sort algorithm, the comparison complexity for sorting is $\frac{N}{R}O(R\log R)$.

For the worst case, all the beats are chosen for consideration, and want to choose *m* coefficients, *m*<*R*, the times of comparisons needed is: $\frac{mN}{R}$.

For the best case, all the *R* coefficients will be chosen, then no comparison is needed. On the other hand, if every beat provide one coefficient's position index till *m* is met, the total comparison is: $\frac{(1+m)m}{2}$. Notice, in the best case, the comparison has nothing to do with the length *N*, which help a lot in reducing the complexity, because *N* is a normally a very big number, such as in the example ECG 16265, N is 24576.

In the amplitude normalization, for each beat, it needs $\frac{N}{R}O(R\log R)$ times of comparisons, and $\frac{N}{R}O(R\log R)$ multiplications.

In the quantization part, since totally $\frac{mN}{R}$ numbers of coefficients need to be quantized, and each quantization needs six multiplication, one 2's power, one adding, so totally we need $\frac{6Nm}{R}$ times multiplications, $\frac{Nm}{R}$ times adding, and $\frac{Nm}{R}$ times 2's power calculations.

To recover from the quantization, we need $\frac{Nm}{R}B$ times comparisons. Where *B* is the number of bits used for the quantization.

In the linear predication, this thesis uses second order LP, thus, there is totally $\frac{N(7m+3)}{R}$ times floating point multiplications, and $\frac{N(5m-2)}{R}$ times floating point adding, should the Yule-Walker method be used.

B: Conclusions:

Table 4.1 and table 4.2 respectively sum up the computational complexity for the worst and for the best case using WBLP method.

Comparison	Multiplication	Adding	2's Power
$\frac{mN}{R}+\frac{2N}{R}O(R\log R)+\frac{mN}{R}B$	$\frac{N}{R}O(R\log R)+\frac{6mN}{R}+\frac{(7m+3)N}{R}$	$\frac{mN}{R}+\frac{(5m-2)N}{R}$	$\frac{mN}{R}$

Table 4.1. Worst case computation complexity for 2nd order WBLP method.

Comparison	Multiplication	Adding	2's Power
$\frac{2m}{m+1}+\frac{2N}{R}O(R\log R)+\frac{Nm}{R}B$	$\frac{N}{R}O(R\log R)+\frac{6mN}{R}+\frac{(7m+3)N}{R}$	$\frac{Nm}{R}+\frac{(5m-2)N}{R}$	$\frac{mN}{R}$

Table 4.2: Best case computation complexity for 2nd order WBLP method.

Thus the computational complexity for LP in the worst case is only $\frac{N}{R}O(R\log R)+\frac{Nm}{R}B$. In most ECG record analyzed in this thesis, $\frac{N}{R}$ ranges from 1 to 100, and R is set as 256, and *m* falls in the range of 150 to 30, and the quantization bits used ranges from 3 to 8. Thus the computational complexity for WBLP even in the worst case is quite small.

4.2 1-D EZW encoding

This algorithm looks at 3-level decomposed wavelet coefficients as a special spanning tree. The coarsest low pass filtered coefficients are at the very top positions of the tree, and its high pass filtered counterparts, with the same number, are right below it. From there on, each coefficient has two descendents, and the very bottom coefficients are the most refined high pass filtered coefficients, whose number is exactly half of the total vertices in this spanning tree.

In the dominant pass, by using the depth first searching strategy, the very bottom coefficients are compared with the threshold and, if they are neither *NEG* nor *POS*, they will be temporarily marked as *ZR*, meanwhile, provide the logical reference for helping decide their parents' attributes from either *ZR* or *IZ*, depending on their parents' absolute values being smaller or bigger than the thresholds.

For example, in the spanning tree listed below:

22							
9							
7				-3			
10		-18		21		-12	
5	6	12	30	60	50	-5	-4

Figure 4.1: Spanning tree structure

The sequence is to first compare 5 and 6 respectively with the threshold, and then as a pair to return a logical value to decide if their parent 10 is a *ZR* or *IZ*, depending on if 10 being below the threshold or not. Then it comes to 12 and 30 to decide for –18 likewise, etc. The whole sequence of scanning is: 5, 6, 10, 12, 30, -18, 7, 60, 50, 21, -5, -4, -12, -3, 9, 22.

In this way, the computational complexity is calculated accordingly.

A. Dominant pass:

A.1 Marking *ZTR, IZ, NEG, POS*. Table 4.3 shows the computational complexity to mark *ZTR, IZ, NEG*, and *POS* for wavelet coefficients.

Pseudo code	Complexity
Function mark_bottoms_temp ( )	Total coefficients at very bottom: $\frac{N}{2}$
{	
if abs(B1) > Threshold then	worst case: N comparisons and
if (B1) < 0 then	$\frac{N}{2}$ absolute value calculations.
B1.mark = NEG;	
Else	
B1.mark = POS;	best case: $\frac{N}{2}$ comparison.
End if	$\frac{N}{2}$ absolute value calculations
Else	
B1.temp_mark = ZTR %temp mark.	
End if	
)	
Function mark_inner_layers_temp()	
{	Total coefficients at very bottom is
if (abs(in_C) > threshold) then.	$\frac{N}{2}-\frac{N}{2^L}$, where L is the decomposition
if ((in_C) < 0) then	levels. In this thesis, L is 3.
in_C.mark = NEG	$3\times(\frac{N}{2}-\frac{N}{2^L})$
else	
in_C.mark = POS	worst case: $3\times(\frac{N}{2}-\frac{N}{2^L})$ comparisons.
end if	

else
    if (its both children's temporary
        marks are ZTR) then
        in_C.temp_mark = ZTR
    else
        in_C.mark = IZ
end if
}
	$(\frac{N}{2}-\frac{N}{2^L})$ absolute value calculations.
	
	best case: $N-\frac{N}{2^{L-1}}$ comparisons.
	
	$\frac{N}{2}-\frac{N}{2^L}$ absolute value calculations.

Function mark_toppest_layer_temp()
{
    if (abs(T1)>threshold) then
        if (T1)<0 then
            T1.mark = NEG
        else
            T1.mark = POS
        end if
    else
        if (its child's temp mark is ZTR)
            T1.mark = ZTR
        else
            T1.mark = IZ
    end if
}
	Total number of coefficients $\frac{N}{2^L}$.
	
	worst case: $\frac{N}{2^{L-1}}$ comparisons
	
	$\frac{N}{2^L}$ absolute value calculations.
	
	best case: $\frac{N}{2^{L-1}}$ comparisons
	
	$\frac{N}{2^L}$ absolute value calculations.

Function coefficients_index_in_tree( ) { children1. index = 2*(parent.index-1)+length(parent_generation) children2.index = 2* parent.index +length(parent_generation) }	Minimum numbers of adding: $N-\frac{N}{2^{L-1}}+\frac{N}{2^{L}}$ Minimum multiplications: $N-\frac{N}{2^{L-1}}\frac{N}{2^{L}}$
Function Rid_Redundent_ZTR( ) { if parent.mark == ZTR if its children's temp_mark==ZTR not coding children; }	Indexing computation, worst case: Minimum adding: $N-\frac{N}{2^{L-1}}\frac{N}{2^{L}}$ Minimum multiplications: $N-\frac{N}{2^{L-1}}\frac{N}{2^{L}}$

Figure 4.2: Pseudo code for counting the computational complexity for 1D-EZW.

For simplicity, we take the worst case in the column II and sum them up, and so far, the total complexity is:

Comparison	Adding	Multiplication	Absolute value
$\frac{5N}{2}-\frac{N}{2^L}$	$(1-\frac{1}{2^{L-1}})N$	$(1-\frac{1}{2^{L-1}})N$	N

Figure 4.3. Computational complexity in worst case for 1D-EZW one dominant pass.

B: Subordinate pass:

Function subordinate_pass ( )
{ if abs(coef) > top half threshold
      result = 1
   else
      result = 0
   end if
}
	For each *NEG* or *POS*,
	
	Computation for abs: 1
	Compute top half T: 1 add, 1 multi.
	Comparison : 1

Figure 4.4: Computational complexity 1D-EZW's one subdominant case.

C. Best case:

The best case is to sum up the above computational complexity. It contains only one dominant pass and only one subordinate pass.

D. Worst case:

The worst case is when the threshold is set that each time only one coefficient is chosen, as either *NEG* or *POS*, for each pair of dominant pass and each subordinate pass, and at the end, all the coefficients will be chosen. Then at the first round of dominant pass, *N* coefficients are involved, *N-1* for the second round of dominant pass, *N-2* for the third, and 1 for the last. For the subordinate pass, situation is reversed, first time, 1 coefficient needs refined, and at last time, *N* coefficients need to get refined.

E. Conclusion:

Dominant pass / Worst case:

Comparison	Adding	Multiplication	Abs. value
$(\frac{5}{4}-\frac{1}{2^{L+1}})\times N\times(N+1)$	$(\frac{1}{2}-\frac{1}{2^{L}})\times N\times(N+1)$	$(\frac{1}{2}-\frac{1}{2^{L}})\times N\times(N+1)$	$N\times(N+1)$

Figure 4.5: Computational complexity for 1-D EZW dominant passes.

Dominant pass / Best case:

Comparison	Adding	Multiplication	Absolute value
$\frac{5}{2}\times N-\frac{N}{2^{L}}$	$N\times(1-\frac{1}{2^{L-1}})$	$N\times(1-\frac{1}{2^{L-1}})$	N

Figure 4.6: Computational complexity for 1-D EZW dominant passes.

Subordinate pass / Worst case:

Comparison	Adding	Multiplication	Absolute value
$N \times (N+1)$	$N \times (N+1)$	$N \times (N+1)$	$N \times (N+1)$

Figure 4.7: Worst case computational complexity for 1-D EZW subordinate passes.

Subordinate pass / Best Case:

Comparison	Adding	Multiplication	Absolute value
N	N	N	N

Figure 4.8: Best case computational complexity for 1-D EZW subordinate passes.

Summing up the computational complexity from dominant pass and sub-ordinate pass, the computational complexity for 1-D EZW encoding is seen in figure 4.9 and figure 4.10, respectively for the worst case and the best case.

Worst case:

Comparison	Adding	Multiplication	Absolute value
$(\frac{9}{4} - \frac{1}{2^{L+1}}) \times (N+1) \times N$	$(\frac{3}{2} - \frac{1}{2^{L}}) \times (N+1) \times N$	$(\frac{3}{2} - \frac{1}{2^{L}}) \times (N+1) \times N$	$2 \times (N+1) \times N$

Figure 4.9: Worst case computational complexity for 1-D EZW encoding.

Comparison	Adding	Multiplication	Absolute value
$\frac{7}{2}\times N-\frac{N}{2^L}$	$2N\times(1-\frac{1}{2^{L-2}})$	$2N\times(1-\frac{1}{2^{L-2}})$	2**N*

Figure 4.10: Best case computational complexity for 1-D EZW encoding.

F: Complexity computation for the 1-D EZW decoding.

In the worst case, or at least one of the worst cases, is to correspond to the worst encoding process, in that all the wavelet coefficients are encoded eventually, but each dominant pass only encode one coefficient above the threshold as seen in figure 4.11.

IZ				*ZTR*				*ZTR*				*ZTR*			
IZ															
IZ		*ZTR*													
POS	*ZTR*														

Figure 4.11: First round of the dominant pass output from the encoding process.

It has 3 layers. To decode this sequence, it will involve $\frac{N}{2^L}+2\times L-1$ times comparisons, which is to compare if the symbol is *IZ*, or *ZTR*, or *NEG* or *POS*. Then, it will compute the index positions for the descendents of all those *ZTR*'s. There will be $N-\frac{N}{2^L}-2\times L+1$ times multiplications, and $2N-\frac{N}{2^{L-1}}-2\times L+2$ times adding.

In the subordinate pass, it only needs to decode 1 coefficient now. So it involves 1 comparison to compare it lies in the upper half bound with the given threshold. Then it involves 1 adding and 1 multiplication for the refined value.

The output generated by the second round of the dominant pass is shown in the figure 4.12.

IZ				*ZTR*				*ZTR*				*ZTR*			
IZ															
IZ		*ZTR*													
&&	*NEG*														

Figure 4.12: Second round of the dominant pass output from the encoding process.

In this way, the computational complexity for the decoding is shown in figure 4.13.

Comparisons	$\frac{N^2}{2}\left[2^L - \frac{1}{2^{L-1}} - 1\right] + 2\times N\times L\times\left[1-\left(\frac{1}{2}\right)^{L-1}\right] - N$
Adding	$2\times N^2\times\left[1-\left(\frac{1}{4}\right)^{L-1}\right] - \left(\frac{N^2}{2^L} + 2\times L\times N\right)\times\left(1-\frac{1}{2^{L-1}}\right) - N$
Multiplications	$2\times N^2\times\left[1-\left(\frac{1}{4}\right)^{L-1}\right] - \left(\frac{N^2}{2^{L-1}} + 4\times L\times N\right)\times\left(1-\frac{1}{2^{L-1}}\right) - 2\times N$

.

Figure 4.13: Computation complexity for the worst case decoding, dominant pass.

Comparisons	$\frac{N \times (1+N)}{2}$
Adding	$\frac{N \times (1+N)}{2}$
Multiplications	$\frac{N \times (1+N)}{2}$

Figure 4.14: The worst case for the subordinate pass.

Figure 4.15 shows the best scenario. The decoding only needs one dominant pass and one subordinate pass.

POS				POS				NEG				NET			
NEG				POS				POS				POS			
POS		NEG		NEG		POS		NEG		POS		POS		NEG	
P O S	P O S	N E G	N E G	P O S	P O S	P O S	N E G	N E G	P O S	P O S	N E G	P O S	P O S	P O S	N E G

Figure 4.15: Best case the first round of the dominant pass output.

The total complexity is to sum up the number of the comparisons, adding and multiplications from its both dominant and subordinate passes, as seen in figure 4.16.

Comparisons	2*N
Adding	N
Multiplications	N

Figure 4.16. Summary of the best case 1-D EZW.

Because of N is a very big number, even for the best case in which computational complexity is *O(N),* it still takes a long time to compute. It is necessary to reduce the value of N by breaking down it into cycle by cycle. Thus, cycle-to-cycle compression does help reduce the complexity, especially for the worst case, computational complexity is $O(N^2)$.

4.3 1-D Modified EZW

The computational complexity of the 1-D MEZW is the best case of 1-D EZW, thus, its computational complexity is *O(N)* as seen in figure 4.16.

Chapter 5 Summary and Conclusions

The purpose of this study is to use transform compression techniques to compress ECG signals and find their compression effectiveness. All of the algorithms simulated in this thesis employ wavelet transform and the essence of the compression is to find an efficient way to use as few bits as possible to encode, as accurately as possible, the locations of the significant wavelet coefficients in the time-frequency domain and their magnitudes.

This study uses three algorithms to compress ECG. The first is WBLP, a method published in a journal, the second is 1-D EZW, and the third is modified EZW (MEZW). WBLP method does not only approximate the magnitudes of the significant wavelet coefficients from every ECG beats, but also estimates their locations by the generalizations from the sample beats. While MEZW shares the same characteristic as 1-D EZW that both of them accurately encode the locations of the significant coefficients for each of the beats, it is different from 1-D EZW in that it conducts uniform quantization to all of them at once.

The study finds that WBLP is good at compressing sinus-rhythm ECG signals, but poor for compressing abnormal ECG signals, especially poor at high compression ratios. It overlooks the fact that for abnormal ECG signals, the high frequency spectrum and low frequency spectrum do not always appear periodically from beats to beats. Thus, evaluating only a limited number of ECG beats to decide a fixed set of locations of the significant coefficients for all the beats will certainly cause big distortions at recovering those ECG beats who have very different frequency spectrums.

The study uses 12 discrete ECG signal records to test WBLP, and finds when compressing sinus-rhythm ECG, in average, the NRMSE and NMAE are respectively 35.3% and 12% for compression ratio of 25; When compressing abnormal ECG, in average, NRMSE and NMAE are respectively 38.47% and 40.94% for compression ratio of 7.

1-D EZW algorithm is a progressive compression algorithm. Attribute to the energy degradation feature in the wavelet decompositions, and with the help of the spanning tree structure, the locations of the significant coefficients are encoded by four symbols: P, N, IZ and ZR. The more ZR occurs in the spanning tree, the higher compression ratios it can achieve. In this study, 1-D EZW is applied in two different ways: direct implementation and cycle-to-cycle implementation.

This study reports, by averaging the testing results, that for compressing sinus-rhythm ECG, WBLP outperforms 1-D EZW direct implementation at smaller compression ratios, but not at high compression ratios. WBLP beats 1-D EZW cycle-to-cycle implementation for compressing sinus-rhythm ECG at any compression ratio. In compressing abnormal ECG beats, 1-D EZW of both implementations outperform WBLP at any compression ratio.

In average, when compressing sinus-rhythm ECG using 1-D EZW directly implementation, the average NRMSE and NAME of are respectively 26.5% and 18% corresponding compression ratio of 20; when compressing abnormal ECG, the average NRMSE and NMAE are 28.5% and 23.8% with compression ratio of 20. Though 1-D EZW direct implementation has the better compression result, its formidable computational complexity will prohibit it from being used if the computational time is a concern. Thus, 1-D EZW cycle-to-cycle compression with a constant threshold to all the beats is the substitute for compressing abnormal ECG and the testing results show that the average NRMSE and NMAE respectively are: 40% and 25.5% for compression ratios of 15.

MEZW algorithm is used to further increase the compression ratio and to reduce the computational complexity. It one time chooses all the significant coefficients over the threshold, encode their locations in the spanning tree structure, and uniformly quantize their magnitudes. This approach, implemented in both direct and cycle-to-cycle ways, outperforms WBLP and 1-D EZW in compressing both sinus-rhythm and abnormal ECG signals. The study reports that the average NRMSE and NMAE for its direct implementation for compressing sinus-rhythm ECG are respectively 24.5% and 12.2% at compression ratio of 28; for compressing abnormal ECG are respectively 20.5% and 22.5% respectively at compression ratio of 25.

Though the computational complexity of MEZW direct application has been greatly reduced compared with its counterpart of 1-D EZW, its complexity can still be so high that it needs to be implemented cycle by cycle. MEZW cycle-to-cycle with constant desired threshold for all the beats outperforms both its counterpart in 1-D EZW and WBLP in compressing abnormal ECG, its average NRMSE and NMAE respectively are: 25.05% and 38.5% at compression ratio of 23; And its average NRMSE is better than that from WBLP in compressing sinus-rhythm ECG as 30.05% at compression ratio of 20, but worse in average NMAE than WBLP as 21% at compression ratio of 23.

The conclusions from the study are that WBLP is very good at compressing sinus-rhythm ECG. But by using WBLP, this thesis does not get results as good as those from

[2]. Though the testing results from some particular sinus-rhythm ECG objects are as good as shown in [2], the average NRMSE and NMAE from recovering sinus-rhythm ECG in this thesis are about double those from [2]. The discrepancy is caused probably by using different set of sinus-rhythm ECG from [2] for the testing. However, the paper [2] does not discuss applying WBLP to compress abnormal ECG signals. This thesis also shows using EZW and MEZW on compressing sinus-rhythm ECG generates comparable results with those in [2]. The true superior of EZW and MEZW is at compressing abnormal ECG, and WBLP is not effective in this category. On the other hand, MEZW and 1-D EZW direct implementation are good at compressing both sinus-rhythm ECG and abnormal ECG when the computational time is not a concern, and they bypass the *QRS* detection difficulties.

Chapter 6 Recommendations for Future Research

This study did the simulation of the several ECG transform compression techniques. More ECG records can be selected for testing and simulation. In this thesis, all the ECG are original sampled at 250Hz and digitized in 12 bits, future research can test ECG records with different original sampling rate and different resolution levels.

For the WBLP to compress abnormal ECG signals, when deciding a fixed set of index locations of the significant wavelet coefficients, sample beats should be taken from different part of the ECG signal sequence to guarantee abnormal beats' unique features are considered.

For 1-D EZW and MEZW, the future study can try different decomposition levels and different wavelet functions to see the compression results.

LIST OF REFERENCES

[1] Jerome M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Trans. On Signal Processing,* vol.41, No12, 1993

[2] A.G.Ramakrishnan and Supratim Saha, "ECG Coding by Wavelet-Based Linear Prediction, " *IEEE Trans. Biomed. Eng*., vol. 44, No.12, pp.1253-1261, 1997.

[3] J.Pan and W.J.Tompkins, " A Real-Time *QRS* Detection Algorithm," *IEEE Trans. Biomed. Eng*., vol. BME-32, No.3, pp. 230-236, 1985

[4] http://www.physionet.org/cgi-bin/chart

[5] Kahlid Sayood, "Introduction to Data Compression," 2nd Edition, pp.312-319

[6] J. Makhoul, "Linear prediction: A tutorial review," *Proc. IEEE*, vol. 63, pp. 561--580, 1975.

[7] http://medlib.med.utah.edu/kw/ecg/ecg_outline/Lesson1/index.html

[8] S.Jalaleddine and C.Hutchens, "ECG Data Compression Techniques – A Unified Approach," *IEEE Trans. On Biomed. Eng.,* vol. 37, No.4, pp329-343, 1990

[9] P.Hamilton and E.Limited, "Open Source ECG Analysis Software Documentation," http://www.eplimited.com

APPENDICES

Appendix 1 ECG Signal Records

The ECG waves shown in Appendix 1 are those used for the algorithm simulation and ECG compressing testing in this thesis. Their wave forms are not shown in the previous parts of the thesis.

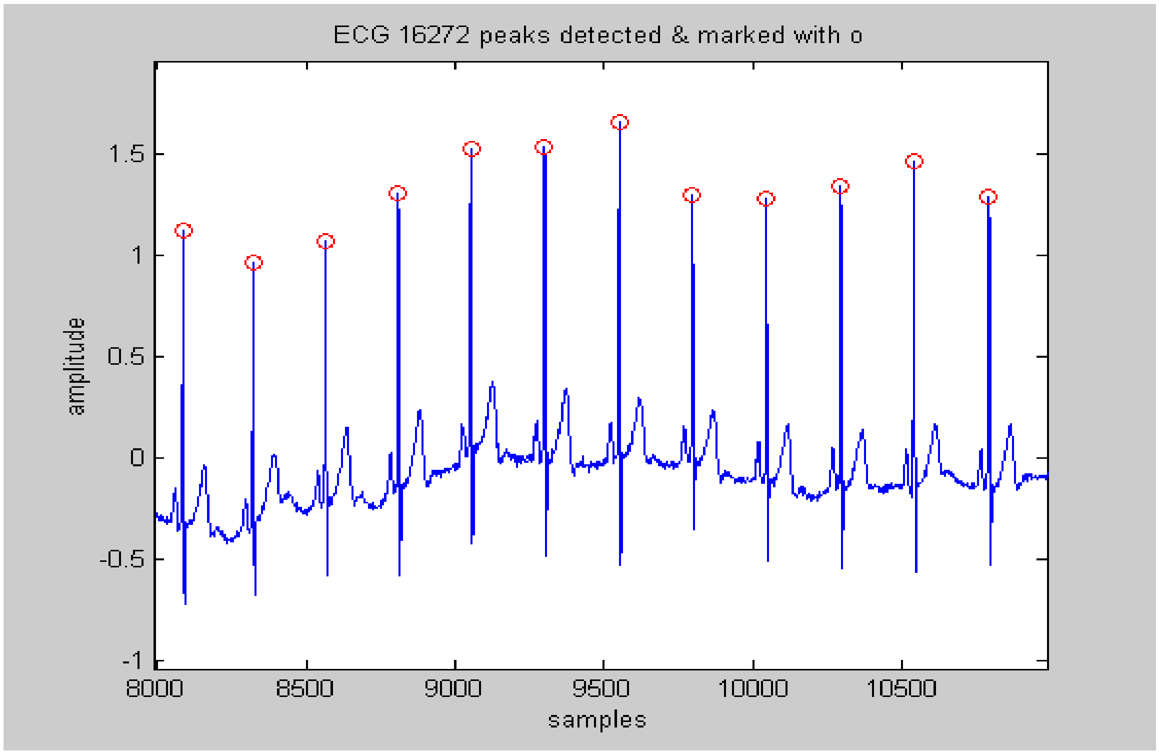


Figure A1.1 Sinus-rhythm ECG 16272 waveform

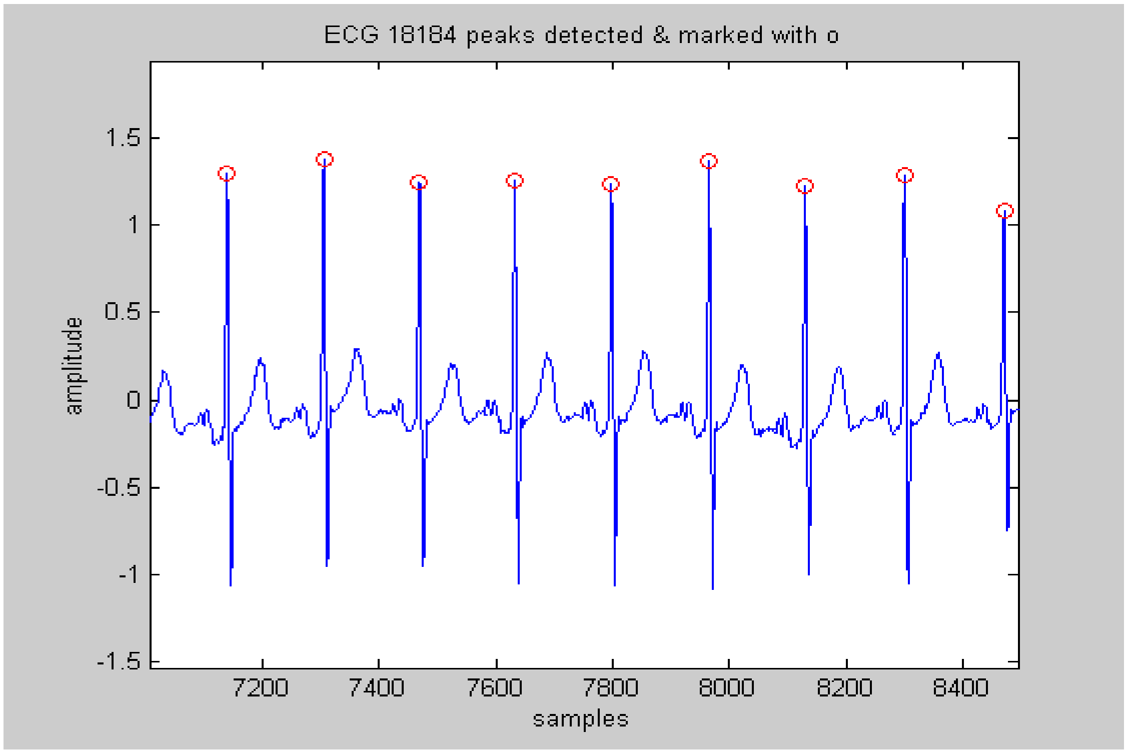


Figure A1.2 Sinus-rhythm ECG 18184 waveform

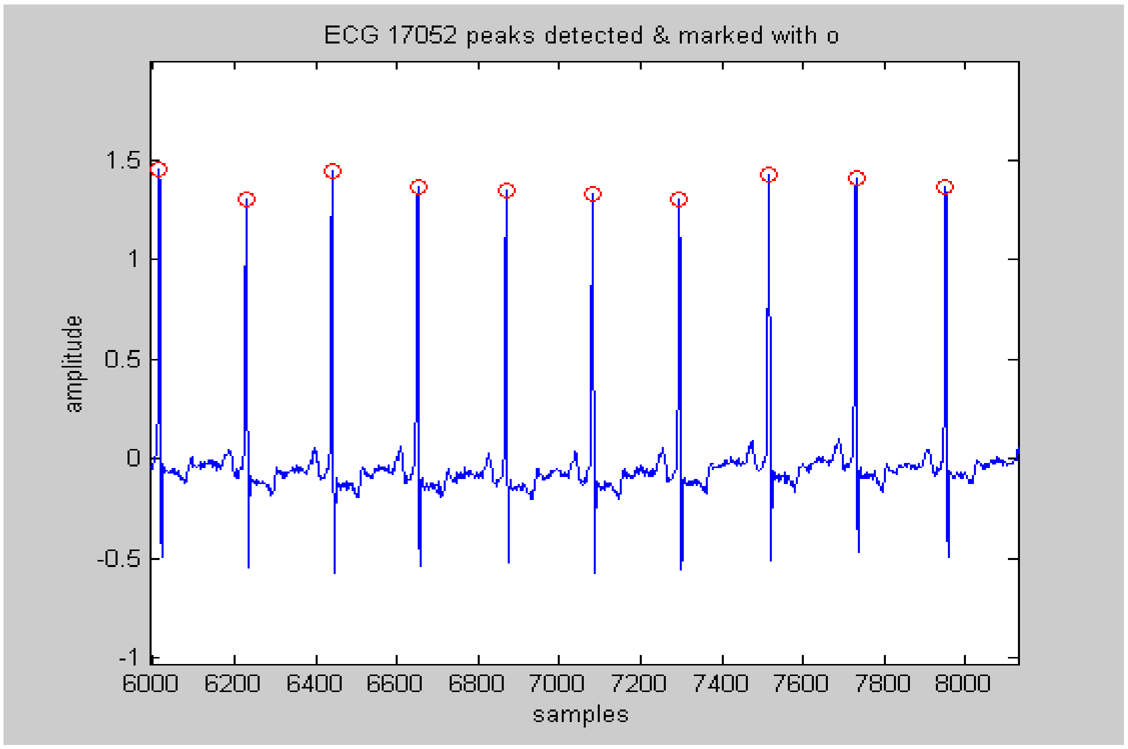


Figure A1.3 Sinus-rhythm ECG 17052 waveform

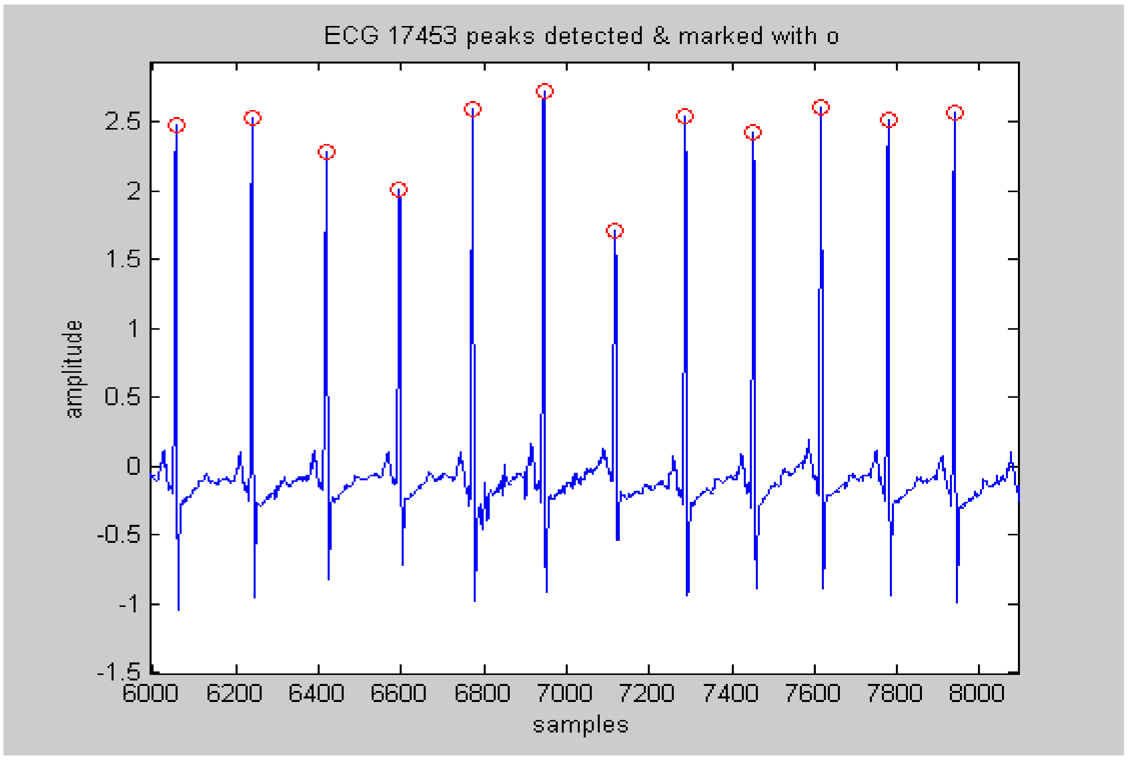


Figure A1.4 Sinus-rhythm ECG 17453 waveform

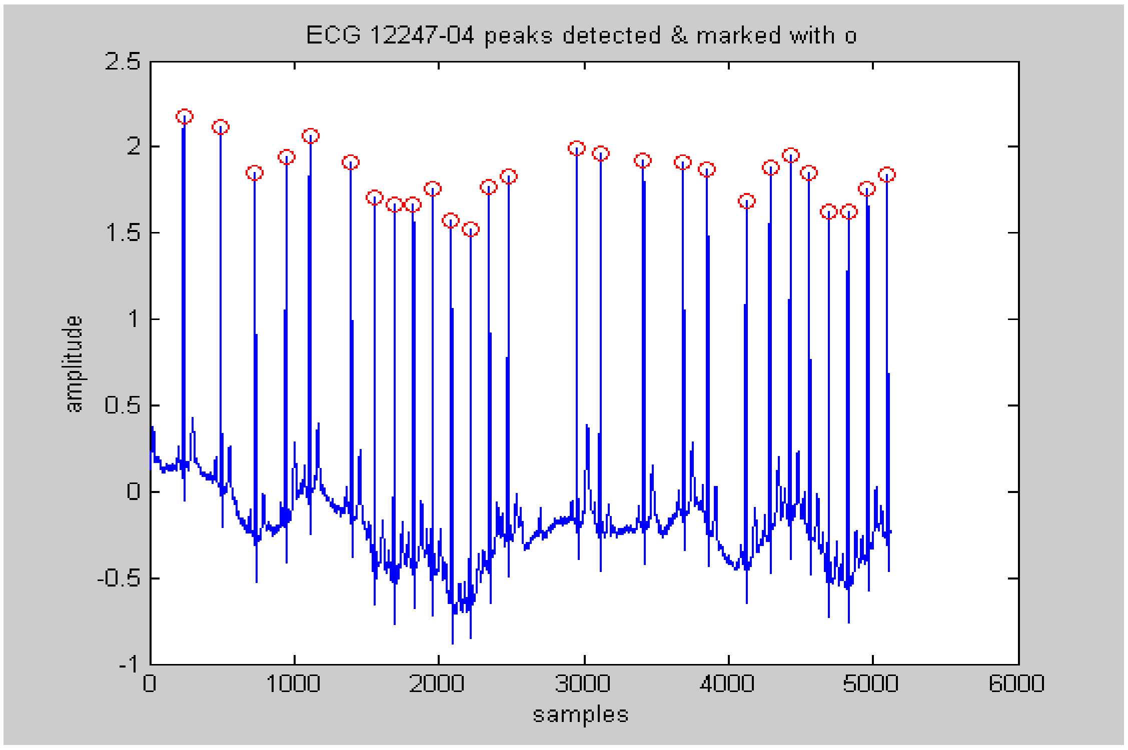


Figure A1.5 Abnormal ECG 12247_04 waveform

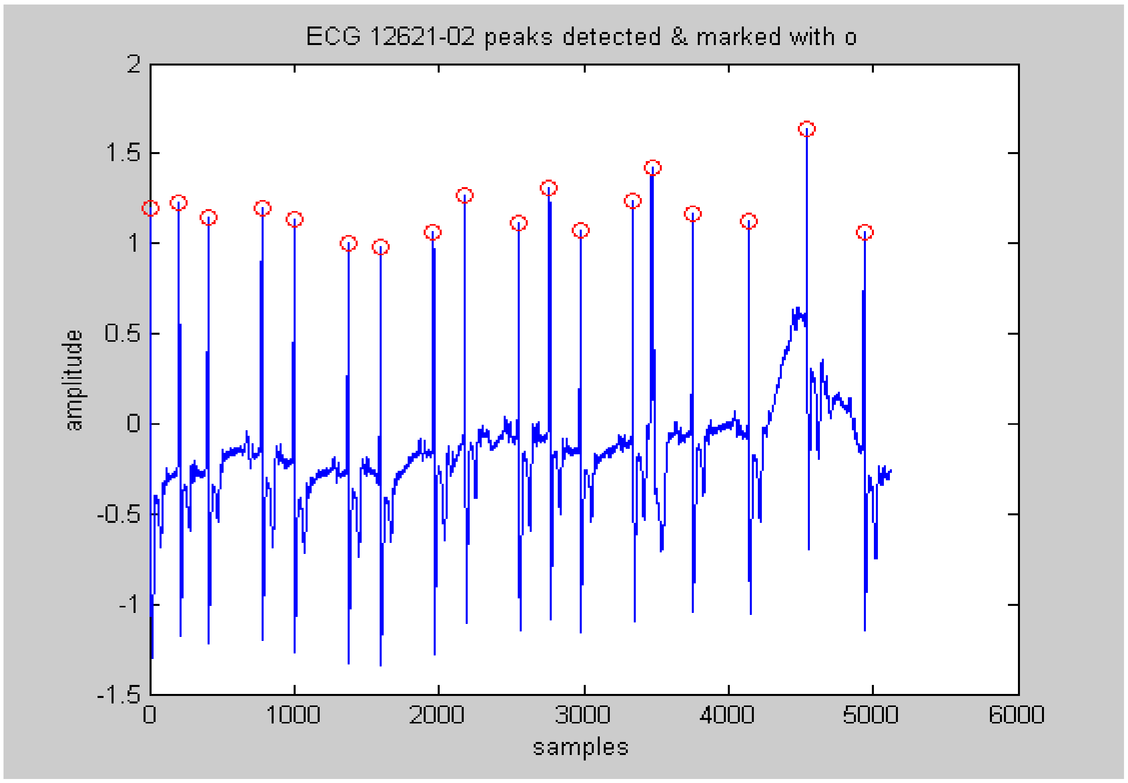


Figure A1.6   Abnormal ECG 12621_02 waveform

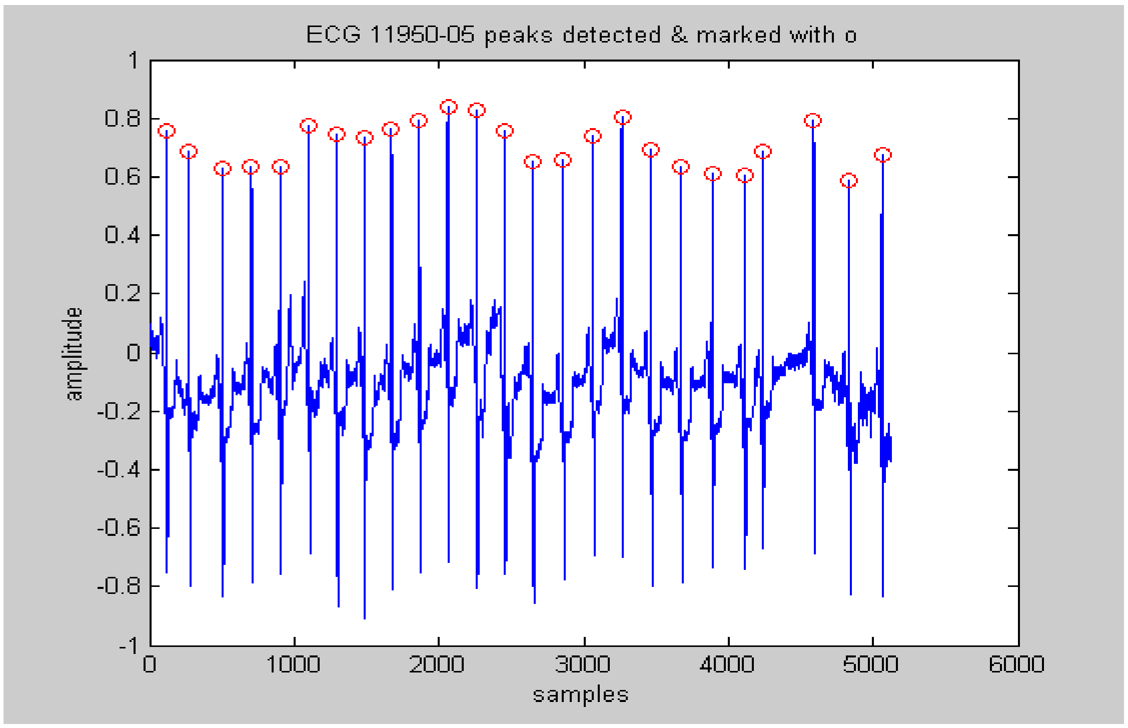


Figure A1.7   Abnormal ECG 11950_05 waveform

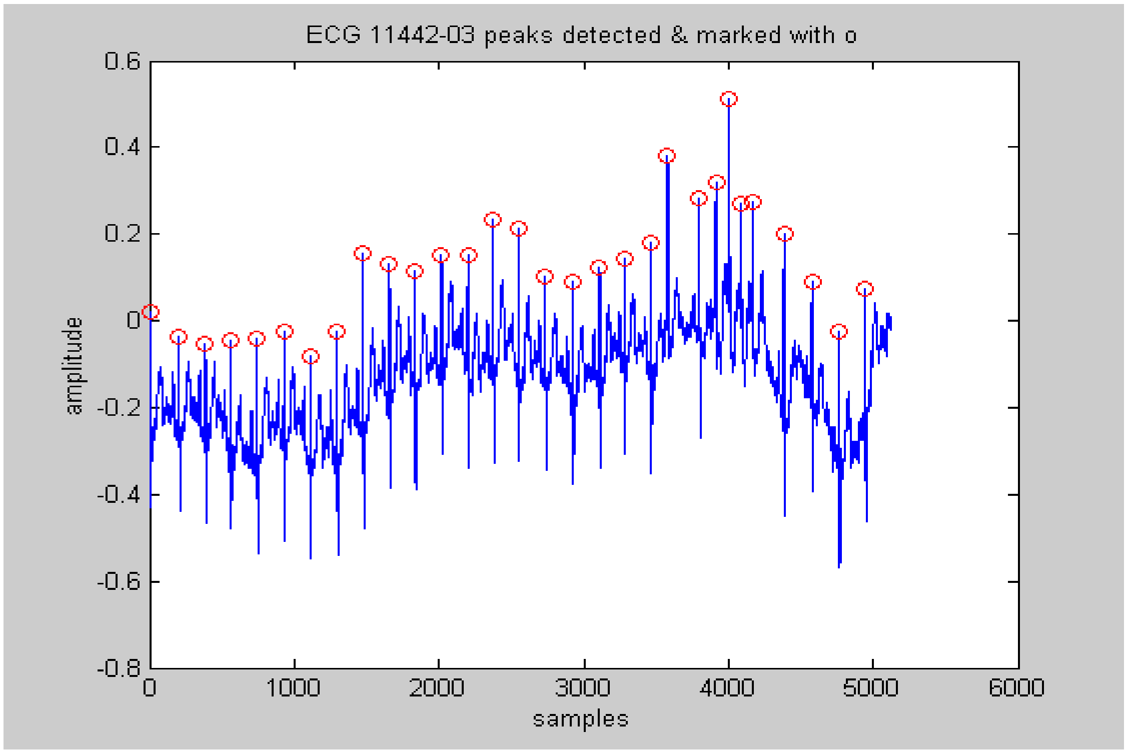


Figure A1.8   Abnormal ECG 11950_05 waveform

Appendix 2 Selected Software

Contents of Selected Software

A2.1 Simulation of Wavelet Based Linear Prediction Algorithm

```
% Program file: WBLP.m
% Author: Xiaohua Zhou
% Time: March, 2003
% Program purpose: To simulate the wavelet based linear prediction method.

clear all;
close all;

% Sinus-rhythm ECG signals
%ecg_normal=load('C:\matlabR12\work\ecg_data\16265.txt');
%ecg_normal=load('C:\matlabR12\work\ecg_data\16773.txt');
%ecg_normal=load('C:\matlabR12\work\ecg_data\16272.txt');
%ecg_normal=load('C:\matlabR12\work\ecg_data\18184.txt');
%ecg_normal=load('C:\matlabR12\work\ecg_data\17052.txt');
% ecg_normal=load('C:\matlabR12\work\ecg_data\17453.txt');
% end of Sinus-rhythm ECG

% ECG containing abnormal beats.
ecg_normal=load('C:\matlabR12\work\ecg_data\11442_03.txt'); % good example to show LP
%ecg_normal=load('C:\matlabR12\work\ecg_data\11950_05.txt');  % very good for LP
%ecg_normal=load('C:\matlabR12\work\ecg_data\12247_04.txt'); % good for lP
%ecg_normal=load('C:\matlabR12\work\ecg_data\12431_03.txt'); %excellent for LP
%ecg_normal=load('C:\matlabR12\work\ecg_data\12621_02.txt'); % very good one for lP
%ecg_normal=load('C:\matlabR12\work\ecg_data\12936_01.txt'); % excellent one, should use it
% end of ECG containing abnormal beats

% to resample the original signals to 250Hz if it is not.
t1=ecg_normal(:,1);
dt=t1(6)-t1(5);

if (dt~=0.004)
   t_start = t1(1);
   t_end = t1(end);
   duration = length(t1);
   t=t_start:0.0040:(2*duration-1)*0.004;
   ecg1=ecg_normal(:,2);
   ecg=resample(ecg1,2*length(ecg1),length(ecg1));

else
   ecg=ecg_normal(:,2);
   t=ecg_normal(:,1);

end
dt = t(6)-t(5);

delay = ceil(0.2/dt);

%low pass filtering starts
```

```
A=[1 0 0 0 0 0 -2 0 0 0 0 0 1];
B=[1 -2 1];
ylow=filter(A,B,ecg);


%high pass filtering starts
A=[1 zeros(1,15),32,zeros(1,15),-1];
B=[1,1];
yhigh=filter(A,B,ylow);

% Getting the derivatives
A=[1 2 0 -2 -1];
A=A./8;
B=[1];
ydev=filter(A,B,yhigh);

% Getting the absolute value
ydev_abs = abs(ydev);

%moving average filtering
N=30;
A=ones(1,N)./N;
B=1;
mov_avrg_sqr=filter(A,B,ydev_abs);

%find RR waves
[QRS,peaks, RR] = find_peaks1(mov_avrg_sqr,dt);

% post processing to get RR waves more accurately
L_QRS=length(QRS);

if peaks(1)>50
   pks(1)= peaks(1)-45;
   [v,p]=max(ecg(1:(peaks(1)+10)));
   pks(1)=p;
else
   [v,p]=max(ecg(1:50));
   pks(1)=p;
end

for i=2:L_QRS
   pks(i)=peaks(i)-45;
   range = pks(i)-15:pks(i)+15;
   [v,p]=max(ecg(range));
   pks(i)=range(p);
end

pks = unique(pks);
L_pks = length(pks);

figure,plot(ecg); title('ECG peaks detected & marked with o');
```

```
ylabel('amplitude');
xlabel('samples');
hold on;

for i=1:L_pks
   plot(pks(i),ecg(pks(i)),'ro');
end

L_ecg = length(ecg);

intv(1)= pks(1);
for i=2:L_pks
   intv(i) = pks(i)-pks(i-1);
end
intv(L_pks+1)=L_ecg - pks(L_pks);
L_intv = length(intv);

%find the largest interval,
% then upgrade signals to its next 2's power
MBP = 2^ nextpow2(max (intv));

% period normalization of the first beat
ecg_beats(1).samples = ecg(1:pks(1));
ecg_rsmpld(1).samples=resample(ecg_beats(1).samples,MBP,intv(1),0);

% period normalization to the beats
for i=2:L_pks
    ecg_beats(i).samples=ecg(pks(i-1)+1:pks(i));
   ecg_rsmpld(i).samples=resample(ecg_beats(i).samples,MBP,intv(i),0);
end
ecg_beats(L_pks+1).samples = ecg(pks(L_pks)+1:L_ecg);

% period normalization of the last beat
ecg_rsmpld(L_intv).samples=resample(ecg_beats(L_intv).samples,MBP,intv(L_intv),0);

% amplitute normalization
[PAN,ecg_scale] = ecg_AmN (ecg_rsmpld);

% to see if PAN can be reconstructed perfectly
% [a,b]= PAN_beats_rec(PAN,intv,ecg_scale, MBP);

clear ecg_r  ;
clear  y3 x2  x2 y  ecg_normal  t x;

% try to select fixed position set from middle
% buddle of the coefficients.
middle= floor(L_pks /2);
lower_middle = floor(middle / 2);
higher_middle = L_pks - middle;

% choose any of pan beat, just want to know
```

```
%the length of the filtered coefficients

[app, deta]=mallat(PAN(2).samples);

L_wvltCoef = length(app);

% get fixed coefficient position set.
coefKept=110; % how many coefficients to keep
[s_Pos_a, s_Pos_d]=Get_fixed_coefs(PAN,lower_middle,higher_middle,coefKept, L_wvltCoef);

% LP flag for if conducting linear prediction or not
LP_flag=1;

%quantization bits
bits=7;
if LP_flag ~= 1
   LP_order = 0;
else
   LP_order = 2;
end

% encoder
[a_residue_hat,a_coef,a,d_residue_hat,d_coef,d,lc]= encoder_quantize(PAN, s_Pos_a, s_Pos_d,
LP_flag,LP_order,bits);

% decoder
PAN_rec = PAN_decoder(a_residue_hat,a_coef, LP_flag, LP_order,s_Pos_a,
s_Pos_d,lc,d_residue_hat,d_coef,a,d, MBP);

% recover from the pan beats
[ecg_rec,ecg_r] = PAN_beats_rec(PAN_rec,intv,ecg_scale, MBP);

L_ecg = length(ecg);
fprintf('LP_flag = %d  ',LP_flag);

 Nt=L_pks-1; % total number of beats
 Nr=coefKept; % number of wavelet coefficients
 Aa=8; %bits for the scale factor
 Ap=8; % for intervals
 p=LP_order;
 bp=bits; %quantiation bits
 bnz=8; % use 8 bits to transmit the s_Pos
 bap=2; % use 2 bits for MBP
 baa=0; % AASF

 % the last 8 bits for encoding
 % how many coefficients kept.
CR = (L_ecg*12)/((Nt*(Nr*bp+Aa+Ap)+Nr*(p*bp+bnz)+bap+baa) + 8) ;

 fprintf('CR = %f  ', CR);
 bit_rate =((Nt*(Nr*bp+Aa+Ap)+Nr*(p*bp+bnz)+bap+baa) + 8)/(Nt);
 fprintf('bit_rate = %f\n', bit_rate);

nrmse = sqrt( sum((ecg-ecg_rec).^2)/ sum(ecg.^2));
fprintf ('NRMSE = %f  ', nrmse);

peaks_errors = (ecg(pks(2:end-1))-ecg_rec(pks(2:end-1)));

s=0;
t=0;
t1=0;
for i=2:L_pks-1
    t=max(abs(ecg_r(i).samples - ecg_beats(i).samples));
    t1=max(ecg_beats(i).samples)-min(ecg_beats(i).samples);
    s=s+t/t1;
 end

 NMAE = s/(L_pks-2);

 fprintf ('NMAE= %f ', NMAE);

 fprintf('NMPE= %f  ', NRMSE_Peak );
 fprintf ( 'C /%d , B %d \n', coefKept, bits);

figure(3);
subplot(3,1,2),plot(ecg_rec); title('reconstructed ecg');
ylabel('Amplitute');
% axis([6000,6900,-3.5,3.5]);
subplot(3,1,1),plot(ecg);
% axis([6000,6900,-3.5,3.5]);
title('original ECG signals');
ylabel('Amplitude')
subplot(3,1,3),plot(ecg-ecg_rec);
% axis([6000,6900,-1.2,1.2]);
title('Reconstruction errors');
xlabel('samples');
ylabel('Amplitude');

i=10;
ab=ecg(pks(i)+1:pks(i+1)); figure(4),plot(ab);
plot(pks(i)+1:pks(i+1), ab);
title('one cycle interpreted according to the original paper');
 xlabel('samples');
 ylabel('amplitute');

figure(5);
plot(pks(2:end-1),peaks_errors,'r*');
hold on;
plot(ecg-ecg_rec);
title('ecgs samples recovery errors with peaks errors marked by *');
xlabel('samples');
```

```
ylabel('amplitute');

fprintf('\n\n');
```

A2.1.1 QRS detections

```
% Program: find_peaks.m
% Author:  Xiaohua Zhou
% Time:     March, 2003
% Program purpose:   A real-time QRS detection algorithm [3]
% Parameters:   ecg_intg     ECG signals after the preprocessing
%               delta_t      Sampling interval
%               QRS          Output every QRS wave
%               peaks        Output every location of R wave
%               RR           output every RR wave

intv1000 = 250; % one second interval, for 250Hz sampling rate;

RR = [250,250,250,250,250,250,250,250]; % intial RR interval contains 8 1 second interval;

nBuffer =zeros(1,8); % initial 8 noise buffers as 0

% initial 8 signal buffers
for i=1:8
   distance = ((i-1)*250+1):i*250;
   sBuffer(i)=max(ecg_intg (distance));
end

PEAKI = max(ecg_intg);
intv200 = 50; % for 250 Hz sampling rate, the 200 ms interval regards to 50 samples.
intv360 = 90; % for 250 Hz sampling rate, the 360 ms interval regards to 90 samples.

SPKI = 0.125*PEAKI + 0.875 * mean(sBuffer);
NPKI = 0.125* PEAKI + 0.875 * mean(nBuffer);

T1 = NPKI + 0.25*(SPKI - NPKI);
T2= 0.5*T1;

delay= 50;
k=1;

ACPT_HIGH = max(RR);
RR_MISS= 1.16*mean(RR);
ACPT_LOW = 90;

delay= ceil(0.3/delta_t);
k=1;
ACPT_HIGH = ceil(0.8/delta_t);
RR_MISS= ACPT_HIGH;
ACPT_LOW = 0.2/delta_t;
flag = 0;
 L=length(ecg_intg);
 T1min = T1;
 T2min = T2;
pos=find (ecg_intg(1:RR_MISS) > T1);
```

```
if ((length(pos)==0) )
    flag = 1;
   pos=find (ecg_intg(1:RR_MISS) > T2);
   if (length(pos)==0)
      [val,pos]=max(ecg_intg(1:RR_MISS));
   end
end

% find the whole range of one peak's family
[p1,lf,rg]=shift_left_right_QRS(pos,ecg_intg,1);

pos = pos(p1)-lf:pos(p1)+rg;

group = filter([1,-1],1,pos);

s= find( group(2:end) > delay -5);

if (length(s)==0)
    s=pos(end);
else
    s=group(s);
end

j=1;

QRS(j).samples = group(1):s(1);

ecg_range = (QRS(j).samples(end)+delay):(QRS(j).samples(end)+delay+RR_MISS);

if QRS(j).samples(1)==1
    NPKI = 0;
else
    NPKI = mean (ecg_intg(1:QRS(j).samples(1)-1));
end

if flag ==1
    SPKI = 0.25*PEAKI+0.85*SPKI;
    flag = 0;
else
    SPKI = 0.125*PEAKI+0.875*SPKI;
end

jj=1;
sBuffer(jj)=SPKI;
SPKI = mean(sBuffer);

while (ecg_range(end)< (L) & length(pos) ~=0 )

    NPKI = 0.125*PEAKI + 0.875*NPKI;
    T1= NPKI + 0.25*(SPKI-NPKI);
```

```
T1=0.8*T1;

T2=1/2*T1;

if T2min >T2
   T2min =T2;
end

pos = find(ecg_intg(ecg_range) > T1);
pos = ecg_range(pos);
if (length(pos)==0 )
    flag = 1;
    pos = find(ecg_intg(ecg_range)>T2);
    if (length(pos)~=0)
       pos=ecg_range(pos);
    else
       pos = find(ecg_intg(ecg_range)>T2*0.8);
       if (length(pos)~=0)
           pos = ecg_range(pos);
       else
          [val,pos]=max(ecg_intg(ecg_range));
           pos=ecg_range(pos);
          while (val<0.8*T2) &  (ecg_range+75 < L)
               ecg_range = ecg_range+75; % extend the searching range
               [val,pos]=max(ecg_intg(ecg_range));
               pos = ecg_range(pos);
          end
       end
    end
 end

if (length(pos)~=0)
   [p1,lf,rg]=shift_left_right_QRS(pos,ecg_intg,ecg_range(1));

   pos=pos(p1)-lf:pos(p1)+rg;
   group = filter([1,-1],1,pos);
   s= find(group(2:end) > delay -5);
   if (length(s)~=0)
      s=group(s);
   else
      s=pos(end);
   end
   j=j+1;
   QRS(j).samples = group(1):s(1);
   [a,p1]=max(ecg_intg(QRS(j).samples));
   ecg_range = QRS(j).samples(end)+delay:QRS(j).samples(end)+delay+RR_MISS;
   RR(j-1)=QRS(j).samples(1)-QRS(j-1).samples(end);
   if (RR(j-1)<ACPT_HIGH & RR(j-1)>ACPT_LOW)
      RR2(k)=RR(j-1);
      k=k+1;
```

```
        end

        if k>8
            RR_AVG2 = floor(mean(RR2(k-8:k-1)));
            RR_LOW = floor(0.92*RR_AVG2);
            RR_HIGH= floor(1.16*RR_AVG2);
            RR_MISS = floor(1.16*RR_AVG2);
            for i=1:7
                RR2(i)=RR2(i+1);
            end
            k=k-1;
        end

         if (j>8)
             RR_AVG(j-8)= floor(mean(RR(j-8:j-1)));
         end

        SPKI = mean (ecg_intg(QRS(j).samples));
        NPKI = max (ecg_intg(QRS(j-1).samples(end)+1: QRS(j).samples(1)-1));

        if flag == 1
            SPKI = 0.25*PEAKI + 0.75*SPKI;
            flag = 0;
        else
          SPKI = 0.125*PEAKI+0.875*SPKI;
        end

        jj=jj+1;
        if jj<=8
            sBuffer(jj)=SPKI;
        else
            for q = 1:7
                sBuffer(q)=sBuffer(q+1);
            end
            sbuffer(8)=SPKI;
        end

       SPKI = mean(sBuffer);


    end
end

if (L-ecg_range(1)>0.5*RR_MISS)

    ecg_range = ecg_range(1):L;
    [val, pos]=max(ecg_intg(ecg_range));
    pos= ecg_range(pos);
    [p1,lf,rg]=shift_left_right_QRS(pos,ecg_intg,L);
    pos=pos(p1)-lf:pos(p1)+rg;
    j=j+1;
```

```
   QRS(j).samples =pos;
end

for i=1:j
   AVG_QRS(i)= mean(ecg_intg(QRS(i).samples));
end

AVG=mean(AVG_QRS)*1/10;

AVG_QRS(1);

AVG;

t=1;
for i=1:j
  if  AVG_QRS(i) > AVG
      QRS1(t).samples = QRS(i).samples;
      t=t+1;
  end
end

QRS=QRS1;

figure
L_QRS = length(QRS);
plot(ecg_intg);
hold on;
v=0;
hold on;
for i=1:L_QRS
[v(i),p]=max(ecg_intg(QRS(i).samples));
peaks(i)=QRS(i).samples(p);

plot(peaks(i),ecg_intg(peaks(i)),'ro');
end

meanV = mean(v);

if ecg_intg(peaks(1))<0.4*meanV
   QRS2=QRS(2:end);
   QRS = QRS2;
end

if ecg_intg(peaks(L_QRS))<0.4*meanV
   QRS2=QRS(1:end-1);
   QRS=QRS2;
end

L_QRS = length(QRS);

peaks = 0;
```

```
% first one treated specially
i=1;
[v(i),p]=max(ecg_intg(QRS(i).samples));
   peaks(i)=QRS(i).samples(p);

   if peaks(i)<0.6*meanV
      N=70;
   else
      N=20;
   end
   range = 1:peaks(1)+N;
   [val, pos ] = max(ecg_intg(range));
   if val > ecg_intg(peaks(i))
      peaks(i)=range(pos);
      QRS(i).samples=peaks(i)-10:peaks(i)+10;
   end

% doing the left and right shift adjustment to find the real peaks
for i=2:L_QRS-1
   [v(i),p]=max(ecg_intg(QRS(i).samples));
   peaks(i)=QRS(i).samples(p);
   RR(i-1)=peaks(i)-peaks(i-1);
   RRM = mean(RR);
    if ecg_intg(peaks(i))<0.6*meanV
      N=ceil(1/2*RRM);
   else
      N=20;
   end
   range = peaks(i)-N:peaks(i)+N;
   [val, pos ] = max(ecg_intg(range));
   if val > ecg_intg(peaks(i)) & ((range(pos)-peaks(i-1)) > RRM)
      peaks(i)=range(pos);
      QRS(i).samples=peaks(i)-10:peaks(i)+10;
   end
end

 if val > ecg_intg(peaks(i))
      peaks(i)=range(pos);
      QRS(i).samples=peaks(i)-10:peaks(i)+10;
   end

   peaks = unique(peaks);
   lp = length(peaks);
% last one treated specially
i=i+1;
[v(i),p]=max(ecg_intg(QRS(i).samples));
   peaks(i)=QRS(i).samples(p);
    if peaks(i)<0.4*meanV
      N=70;
   else
      N=20;
   end
   range = peaks(i)-N:length(ecg_intg);
   [val, pos ] = max(ecg_intg(range));
   if val >1.5* ecg_intg(peaks(i))
      peaks(i)=range(pos);
      QRS(i).samples=peaks(i)-10:peaks(i)+10;
   end

   peaks = unique(peaks);
   lp = length(peaks);


   if peaks(1)<10
      QRS_temp(1).samples = 1:peaks(i)+10;
      start = 2;
   else
      start = 1;
   end


   if peaks(end)>L-10
      QRS_temp(lp).samples = peaks(end)-10:L;
      fini = lp-1;
   else
      fini=lp;
   end

   for i=start:fini
      QRS_temp(i).samples = peaks(i)-10:peaks(i)+10;
   end
   QRS=QRS_temp;
   L_QRS = length(QRS);

for i=1:L_QRS-1
   RR(i)=peaks(i+1)-peaks(i);
end

if (L-peaks(L_QRS))>mean(RR)
   range = QRS(L_QRS).samples(end)+75:L;
   [val,p]=max(ecg_intg(range));
   if val > 0.6*meanV
      QRS(L_QRS+1).samples=range(p);
   end
end

L_QRS=length(QRS);

hold on;

for i=1:L_QRS
[v(i),p]=max(ecg_intg(QRS(i).samples));
```

```
peaks(i)=QRS(i).samples(p);
plot(peaks(i),ecg_intg(peaks(i)),'bo');
end

lp=length(unique(peaks));

hold off;
```

A2.1.2 Get a fixed set of locations for significant wavelet coefficients

```
function [s_Pos_a, s_Pos_d]=Get_fixed_coefs(PAN,p1,p2, Nc, L_wv);
% Program: Get_fixed_coefs.m
% Author: Xiaohua Zhou
% Date: March 2003
% Program purpose: To simulate the algorithm used for deciding the set of
%              locations of significant wavelet coefficients to be retained
%              in each cycle [2].
% Parameters:   PAN     PAN beats
%               p1      starting point
%               p2      ending point
%               Nc      number of coefficients to keep
%               L_wv    Total number of wavelet coefficients
%               s_Pos_a Locations for significant approximate coefs.
%               s_Pos_d Locations for significant detail coefs.

if p2>p1
   size_PAN = p2-p1+1;
else
   size_PAN = p1-p2+1;
end

%the dwt coefficients after the db4 mallat algorithm will be
%L_wv approximate coefs. and L_wv detail coefs.
ca=zeros(size_PAN, L_wv);
cd = zeros(size_PAN, L_wv);
c=zeros(size_PAN,L_wv*2);
caa = zeros(size_PAN, L_wv*2);
c_K =c;
dummy=c;

signals=PAN(p1).samples;
for i=p1+1:p2
   signals = [signals;PAN(p1).samples];
end
```

```
j=1;
for i = p1:p2
  [ca1,cd1]=mallat(PAN(i).samples);
   ca(j,:)=ca1';
   cd(j,:)=cd1';
   caa(j,:)=[ca(j,:),cd(j,:)];
   %if lots of negative compoenents, then compare absolute values
   c(j,:)=abs(caa(j,:));

   [dummy(j,:) ,c_K(j,:)]=sort(c(j,:));    %sort gives a descending order
   c_K(j,:)=c_K(j,end:-1:1);    % reverse the descending order to ascending order
   j=j+1;
end

N=1;
s_Pos=[c_K(1,1:N)];
s_P=zeros(1,Nc);
l_set = length(s_Pos);

while (l_set < Nc)

   for i=2:size_PAN
      l_set = length(s_Pos);
      if l_set < Nc
         s_Pos = union(s_Pos,c_K(i,N));  % get the first set of N biggest
      else
         i=size_PAN + 1;
      end
   end

   if (l_set<(Nc))
     if (N<L_wv*2)
        N=N+1;
        s_Pos = union(s_Pos,c_K(1,N));
        l_set = length(s_Pos);
    else
       l_set = Nc+1; % to end the while loop
    end
   end
end
l_set = length(s_Pos);

j=1;
t=1;
for i=1:l_set
   if s_Pos(i) > L_wv
      s_Pos_d(j)=s_Pos(i);
      j=j+1;
   else
      s_Pos_a(t)=s_Pos(i);
```

```
        t=t+1;
    end
end

s_Pos_d= s_Pos_d - L_wv;
function [QRS, peaks,RR]=find_peaks (ecg_intg, delta_t)
```

A2.1.3 WBLP Encoder

```
function [a_residue_hat,coef_a,a,d_residue_hat,coef_d,d,lc]= encoder_quantize(PAN, s_Pos_a,
s_Pos_d, LP_flag,LP_order,quan_bits)
% Program: Encoder_quantize.m
% Author: Xiaohua Zhou
% Time:  March, 2003
% Program purpose: To encode the selected significant coefficients after stacking them
%                accross the beats
% Input Parameters: PAN         Period and Amplitute normalized beats
%                 s_Pos_a     Locations of approximate coefs.
%                 s_Pos_d     Locations of detail coefs.
%                 Lp_flag     Linear prediction flag
%                 LP_Order    Linear Prediction order
%                 quan_bits   Quantization bits
%
% Output Parameters: a_residue_hat  Quantized LP residues for wvlt. approx. coefs.
%                  coef_a         LP coefs. for approximate wavelet coefs.
%                  a            2D array, storing stacked wvlt. approx. coefs.
%                  d_residue_hat  Quantized LP residues for wvlt. detail. coefs.
%                  coef_d         LP coefs. for detail wavelet coefs.
%                  d            2D array, storing stacked wvlt. detail. coefs.
%                  lc           length of wavelet coefficients for each beat


  v=1;
  L_s_Pos_a = length(s_Pos_a);  %the length of the approximate coeffs.
  L_s_Pos_d = length(s_Pos_d);  %the length of the detail coeffs.
  lg_PAN = length(PAN);
  d = zeros(lg_PAN, L_s_Pos_d);
  d_residue_hat=zeros(lg_PAN,L_s_Pos_d);

  for r=1:lg_PAN
    s=PAN(r).samples;
    ls=length(s);
    [ca1, cd1]=mallat(s);
    lc=length(ca1);
    c1=zeros(1,lc);
    d1= zeros(1,lc);
    L_cd1=length(cd1);

    % stacking up the detail coefficients
    for i=1:L_s_Pos_d
      d(r,i)=cd1(s_Pos_d(i));
      d1(s_Pos_d(i)) = cd1(s_Pos_d(i));
    end
    % end of stacking up the detail coefficients

    % stacking up the approximate coefficients
    for i=1:L_s_Pos_a
      a(r,i)=ca1(s_Pos_a(i));
```

```
        c1(s_Pos_a(i))=ca1(s_Pos_a(i));
      end
      % end of stacking up the approximate coefficients.
    end

B=quan_bits;

if (LP_flag ==1)  % linear prediction quantization of the detail coefficients.
    for i=1:L_s_Pos_d
      [d_residue_hat(:,i),coef_d(:,i)]=Get_LP_error(d(:,i),LP_order,B);
    end
else
    for i=1:L_s_Pos_d  % direct quantization of the detail coefficients.
      coef_d(:,i)=0;
      d_residue_hat(:,i)=quant_ee523(d(:,i),B,4*sqrt(var(d(:,i),1)));
    end
end
%     hold on;
%     figure (1), plot(d_residue_hat(:,3),'ro');

if (LP_flag == 1)    % Linear prediction quantization for approximate coefficients.
    for i=1:L_s_Pos_a
      [a_residue_hat(:,i),coef_a(:,i)]=Get_LP_error(a(:,i),LP_order,B);
    end
else
     for i=1:L_s_Pos_a % direct quantization for the approximate coefficients.
      coef_a(:,i)=0;
      a_residue_hat(:,i)=quant_ee523(a(:,i),B,4*sqrt(var(a(:,i),1)));
    end
end
```

A2.1.4 Recover the PAN beats from residues

```
function PAN_rec= PAN_decoder(a_hat, coef, LP_flag,
LP_order,s_Pos_a,s_Pos_d,lc,d_hat,coef_d,a,d, MBP)

% Program:   PAN_decoder
% Author:    Xiaohua Zhou
% Time:      March, 2003
% Program Purpose: To decode the PAN beats from residue sequences
% Input Parameters:  a_hat      Residues of approx. wvlt. coefs.
%                 coef       LP coefs. of approx. wvlt. coefs.
%                 LP_flag     LP flag
%                 LP_order     LP order
%                 s_Pos_a      Locations of approx. coefs.
%                 s_Pos_d      Locations of detail. coefs.
%                 lc          length of the wvlt. coefs.
%                 d_hat        Residues of detail. wvlt. coefs.
%                 coef_d       LP coefs. of detail. wvlt. coefs.
%                 a           storing the recovered stacked approx. coefs.
%                 d           storing the recovered stacked detail coefs.
%                 MBP          Mean beat period [2].

% Output Parameters:  PAN_rec     Recovered PAN beats.

[row,col]=size(a_hat);
[r,c]=size(d_hat);

if LP_flag==1

   for i=1:col
      u_r(:,i)=LP_decoder(LP_order, coef(:,i),a_hat(:,i));
   end

   for i=1:c
      d_r(:,i)=LP_decoder(LP_order,coef_d(:,i),d_hat(:,i));
   end
else
   u_r=a_hat;
   d_r=d_hat;
end

   L_s_Pos_d = length(s_Pos_d);
   L_s_Pos_a = length(s_Pos_a);
for r=1:row
   d1=zeros(1,lc);
   c1=zeros(1,lc);
   ls=MBP;
   [Lo_R, Hi_R]=wfilters('db4','r');


   for i=1:L_s_Pos_d
```

```
        d1(s_Pos_d(i)) = d_r(r,i);
    end

    tempo1=dyadup(d1);
    tempo1=conv(tempo1,Hi_R);
    d11=wkeep(tempo1,ls);


    for i=1:L_s_Pos_a
        c1(s_Pos_a(i))=u_r(r,i);
    end

    tempo1=dyadup(c1);
    tempo1=conv(tempo1,Lo_R);
    a11=wkeep(tempo1,ls);

    PAN_rec(r).samples=(a11+d11)';

end
```

A2.2 EZW encoding and decoding

```
function [ecg_rec, output]=ecg_cmpr_ezw (s, T)

%   Program: ecg_cmpr_ezw.h
%   Author:  Xiaohua Zhou
%   Time:    August, 2003
%   Program purpose:   To conduct EZW encoding to the wavelet coefficients of the
%                  input discrete ECG signal.
%   Input Parameters:  s   Input discrete ECG signal, either a beat or the whole set
%                  T   Desired threshold for EZW encoder and decoder
%   Output Parameters: ecg_rec  Reconstructed ECG signals magnitudes.
%                  output   Output symbols
%
len_s = length(s);
[L(1).samples, H(1).samples]=dwt(s,'db4');
N=3;  % three level decomposition

for i = 2 : N
   [L(i).samples, H(i).samples]=dwt(L(i-1).samples,'db4');
end
input = L(N).samples;

for i=N:-1:1
input =[input; H(i).samples];
end

input = 10000.*input; % amplify the coefs.
input = ceil(input); % take the ceiling integer value

[output,symout] = EZWencoder2(input',T);  % EZW encode

ReWC = EZWdecoder1(output,T);  % EZW decode

for i=1:N
   len_L(i)=length(L(i).samples);
end

% recovering coefs. at different levels from the decoder output

ReWC = ReWC./10000;
L_r(N).samples = ReWC(1:len_L(N));
H_r(N).samples= ReWC(len_L(N)+1:len_L(N)*2);
L_r(N-1).samples = idwt(L_r(N).samples', H_r(N).samples', 'db4', len_L(N-1));
sP = len_L(N)+len_L(N);

for i=N-1:-1:2
   H_r(i).samples = ReWC(sP+1:sP+len_L(i));
   sP=sP+len_L(i);
   L_r(i-1).samples= idwt(L_r(i).samples, H_r(i).samples', 'db4', len_L(i-1));
end
```

```
H_r(1).samples=ReWC(sP+1:end);
ecg_rec = idwt(L_r(1).samples, H_r(1).samples', 'db4', len_s);
e=sum((s-ecg_rec).^2)/length(input);
```

A2.3 Modified EZW Algorithm (MEZW)

A2.3.1 MEZW Encoder

Apart from the subordinate pass is modified to uniform quantize all the selected significant coefficients, the rest code is the same with the code from the generic EZW algorithm.

```
function [output,symout] = EZWencoder2(in,T, quan_bits)
% Program file: EZWencoder2.m
%
% Program Purpose:  Simulate Embedded Zerotree wavelet (EZW) encoder, modified the
subdominant
                    pass of the generic EZW algorithm.

% Input Parameters:  in           3 level decomposed wavelet coefficients
%                    T            desired threshold
%                    quan_bits    quantization bits for the uniform quantizer
% Output Parameters:  Output      quantized magnitudes of the wvlt. coefs.
%                     symout      output EZW symbols.


global EnFifo;
EnFifo=[];
global EnList;
EnList=[];
global output;
output=[];


% These describe the coding parameters
CodingParms.min_element_type  =-inf;
CodingParms.max_element_type  = inf;
CodingParms.input_length      = length(in);


% Code Alphabet

CodeAlphabet.ZERO = 0; % binary 0
CodeAlphabet.ONE  = 1; % binary 0
CodeAlphabet.ZTR  = 2; % binary 00
CodeAlphabet.POS  = 3; % binary 01
CodeAlphabet.NEG  = 4; % binary 11
CodeAlphabet.IZ   = 5; % binary 10

InWC = in;

% Initiate the threshold
%Threshold = 2^floor(log2(max(abs(InWC))));  % Change

Threshold = T; % add change
% Write the file header to output
output = [output CodingParms.input_length Threshold];
symout = [];

% Define some statistics
zeros = 0;
ones  = 0;
output_byte = 0;

% Do the EZW coding

while (Threshold>=T)
   % Dominant Pass
   [InWC,EnList,EnFifo,output,symout]=...
      dominant_pass(InWC, Threshold,EnList,EnFifo,output,symout);

   % Subdominant Pass
  [output,symout]=subordinate_pass...
                  (Threshold,EnList,output,T,symout, quan_bits);

   Threshold = Threshold/2;
end


% Some functions used for the encoder

% Performs one complete dominant pass. Dominant-pass-codes are sent to the
% output stream and the subordinate list is updated
%
function [InWC,EnList,EnFifo,output,symout]...
    =dominant_pass(InWC,Threshold,EnList,EnFifo,output,symout)
  global CodeAlphabet;
  CodeAlphabet.ZERO = 0; % binary 0
  CodeAlphabet.ONE  = 1; % binary 0
  CodeAlphabet.ZTR  = 2; % binary 00
  CodeAlphabet.POS  = 3; % binary 01
  CodeAlphabet.NEG  = 4; % binary 11
  CodeAlphabet.IZ   = 5; % binary 10
  s=zeros(2,1);  % Define a vector, [x; code]
  min_x=0;
  max_x=0;
  len=length(InWC);
```

```
 for i=0:3
     s(1)=i;
     [InWC,EnList,s]=process_element(InWC, Threshold,EnList,s);
     EnFifo=put_in_fifo(s,EnFifo);
 end


 while (~isempty(EnFifo))
   [s,EnFifo]=get_from_fifo(EnFifo);
   %if (~isempty(EnFifo))
   [output,symout]=output_code(s(2),output,symout);
   %end
    if ((s(2)~=CodeAlphabet.ZTR))
      if(s(1)>=4)
        min_x = s(1)*2;
        max_x = min_x+1;
        if ((max_x<=len))
                for x=min_x:max_x
                  Ps=zeros(2,1);
                  Ps(1)=x;
                 % fprintf('x= %d, min_x = %d, max_x = %d ',x, min_x, max_x);
                %  fprintf('InWC_len = %d\n', length(InWC));
                  [InWC,EnList,s]=process_element(InWC, Threshold,EnList,Ps);

                  EnFifo=put_in_fifo(s,EnFifo);
                end
        end
    else
        Ps=zeros(2,1);
        Ps(1)=s(1)+4;
        [InWC,EnList,s]=process_element(InWC, Threshold,EnList,Ps);
        EnFifo=put_in_fifo(s,EnFifo);
    end
  end

  end


 % Perform one subordinate pass
function [output,symout]=subordinate_pass(Threshold,EnList,output,T,symout, quan_bits)

  y = quant_ee523(EnList, quan_bits, 4*sqrt(var(EnList,1)));
  y=ceil(y);
  output= [output,y];

% Builds a dominant pass EZW element from a matrix element and a threshold

function [oInWC,oEnList,os]=process_element(InWC, Threshold,EnList,s)

 global CodeAlphabet;
```

```
  CodeAlphabet.ZERO = 0; % binary 0
  CodeAlphabet.ONE  = 1; % binary 0
  CodeAlphabet.ZTR  = 2; % binary 00
  CodeAlphabet.POS  = 3; % binary 01
  CodeAlphabet.NEG  = 4; % binary 11
  CodeAlphabet.IZ   = 5; % binary 10

  if (s(1)+1)<=length(InWC)
    temp=InWC(s(1)+1);
 end

  if (abs(temp)>=Threshold)
     if (temp>=0)
        s(2)=CodeAlphabet.POS;
     else
        s(2)=CodeAlphabet.NEG;
     end
  else
     if (zerotree(InWC,s(1),Threshold)==1)
        s(2)=CodeAlphabet.ZTR;
     else
        s(2)=CodeAlphabet.IZ;
     end
  end


  if ((s(2)==CodeAlphabet.POS)|(s(2)==CodeAlphabet.NEG))
     EnList=[EnList,abs(temp)]; % put only coefficient magnitude in list,
                         % sign is already coded
     InWC(s(1)+1)=0;
  end

  oInWC=InWC;
  os=s;
  oEnList=EnList;


 % Return 1 if descendance tree is a zerotree

function id = zerotree (InWC, x, Threshold)
  global CodeAlphabet;
  CodeAlphabet.ZERO = 0; % binary 0
  CodeAlphabet.ONE  = 1; % binary 0
  CodeAlphabet.ZTR  = 2; % binary 00
  CodeAlphabet.POS  = 3; % binary 01
  CodeAlphabet.NEG  = 4; % binary 11
  CodeAlphabet.IZ   = 5; % binary 10
  min_x=0;
  max_x=0;

  temp=0;
  maxvalue=0;
  stop=0;
  len=length(InWC);

  if (x<=3)    %  The coarsest subband
     temp=InWC(x+1);
     x=x+4;
     if (abs(InWC(x+1))>=Threshold)
        stop=1;
     else
       ret1=zerotree(InWC, x,Threshold);
       stop=~ret1;
     end
  else
    min_x=x*2;
    max_x=(x+1)*2;
    if (min_x==len)
       id=1;
       return;
    end

    maxvalue=0;
    while (max_x<=len);
        CCC = abs(InWC((min_x+1):max_x));
        III = find(CCC>=Threshold);
        if (~isempty(III))
           stop=1;
           break;
        end
        min_x=min_x*2;
        max_x=max_x*2;
    end
  end
    if (stop==1)
       id=0;
    else
       id=1;
    end

% Puts dominant-pass and subordinate-pass codes in the output stream
 function [output,symout]=output_code(code,output,symout)
   global CodeAlphabet;
   CodeAlphabet.ZERO = 0; % binary 0
   CodeAlphabet.ONE  = 1; % binary 0
   CodeAlphabet.ZTR  = 2; % binary 00
   CodeAlphabet.POS  = 3; % binary 01
   CodeAlphabet.NEG  = 4; % binary 11
   CodeAlphabet.IZ   = 5; % binary 10

   switch (code)
    case CodeAlphabet.ZERO
```

```
      output=[output, 0];
      symout=[symout,' ZERO'];
   case CodeAlphabet.ONE
      output=[output, 1];
      symout=[symout,' ONE'];
   case CodeAlphabet.POS
      output=[output,0,1];
      symout=[symout,' POS'];
   case CodeAlphabet.NEG
      output=[output,1,1];
      symout=[symout,' NEG'];
   case CodeAlphabet.ZTR
      output=[output,0,0];
      symout=[symout,' ZTR'];
   case CodeAlphabet.IZ
      output=[output,1,0];
      symout=[symout,' IZ'];
   otherwise
      disp('Coding Wrong');
      return;
   end

 %  Add a new element FIFO buffer, "first in first out" buffer
 function EnFifo=put_in_fifo(s,EnFifo);
    EnFifo=[EnFifo,s];


  %  Get a element from FIFO buffer
 function [s,EnFifo]=get_from_fifo(EnFifo);
    Len=size(EnFifo,2);
    s=EnFifo(:,1);
    EnFifo=EnFifo(:,2:end);
```

A2.3.2 MEZW Decoder

Apart from the while loop at the main function is modified to decode the uniform quantizer to generate all the selected significant wavelet coefficients, the rest code is the same with the code from the generic EZW algorithm.

```
 function ReWC = EZWdecoder(inputstream,T)
 % Program file:  EZWdecoder.m
%  Program Purpose:  Simulate Modifed Embedded Zerotree wavelet M(EZW) decoder
 % Input Parameter:    Inputstream             The data sequence generated by the encoder.
%                                      T             The desired threshold
%  Output Parameter:  ReWC                   The recovered wavelet coefficients.

global EnFifo;
EnFifo=[];
global EnList;
```

```
EnList=[];
global output;
output=[];

% Code Alphabet

CodeAlphabet.ZERO = 0; % binary 0
CodeAlphabet.ONE  = 1; % binary 0
CodeAlphabet.ZTR  = 2; % binary 00
CodeAlphabet.POS  = 3; % binary 01
CodeAlphabet.NEG  = 4; % binary 11
CodeAlphabet.IZ   = 5; % binary 10
output = inputstream;

% Read the file header to get the height, width and image of the image

Threshold = output(2);
Length    = output(1);
output    = output(3:end);

% Create a empty reconstruction matrix
ReWC = zeros(1,Length);

% Define some statistics
zeros = 0;
ones  = 0;
input_byte = 0;
pixels = 0;

% Do the EZW decoding
while (Threshold>=T)
   % Dominant Pass
   [ReWC,EnList,EnFifo,output,pixels]=...
      dominant_pass(ReWC,Threshold,EnList,EnFifo,output, pixels);
   d=0;
   if (Threshold>=T)
      for i=1:pixels
          d=EnList(:,i);
          temp=ReWC(d+1);
         if (temp < 0)
            ReWC(d+1)=-1*output(i);
         else
            ReWC(d+1)=output(i);
         end

      end
   end
   Threshold = Threshold/2;
end
```

```
% Some functions used for the decoder

% Performs one complete dominant pass. Dominant-pass-codes are sent to the
% output stream and the subordinate list is updated
%
function [ReWC,EnList,EnFifo,output,pixels]=...
   dominant_pass(ReWC,Threshold,EnList,EnFifo,output,pixels)
 global CodeAlphabet;
 CodeAlphabet.ZERO = 0; % binary 0
 CodeAlphabet.ONE  = 1; % binary 0
 CodeAlphabet.ZTR  = 2; % binary 00
 CodeAlphabet.POS  = 3; % binary 01
 CodeAlphabet.NEG  = 4; % binary 11
 CodeAlphabet.IZ   = 5; % binary 10

 s=zeros(2,1);  % Define a vector, [x; y; code]
 min_x=0;
 max_x=0;
 len=length(ReWC);

 for i=0:3
    s(1)=i;
    [ReWC,s,output,EnList]=input_element(ReWC,Threshold,s,output,EnList);
    EnFifo=put_in_fifo(s,EnFifo);
 end


 while (~isempty(EnFifo))
   [s,EnFifo]=get_from_fifo(EnFifo);

   if ((s(2)==CodeAlphabet.POS)|(s(2)==CodeAlphabet.NEG))
         pixels = pixels+1;
   end

    if (s(2)~=CodeAlphabet.ZTR)
       if (s(1)>=4)
       min_x = s(1)*2;
       max_x = min_x+1;
       if (max_x<=len)
            for x=min_x:max_x
                  Ps=zeros(2,1);
                  Ps(1)=x;
                  [ReWC,s,output,EnList]=...
                      input_element(ReWC,Threshold,Ps,output,EnList);
                  EnFifo=put_in_fifo(s,EnFifo);
            end
       end
    else
           Ps=zeros(2,1);
           Ps(1)=s(1)+4;
           [ReWC,s,output,EnList]=...
```

```
            input_element(ReWC,Threshold,Ps,output,EnList);
          EnFifo=put_in_fifo(s,EnFifo);
      end
     end
    end


   % Perform one subordinate pass
  function [ReWC,output,pixels]=...
     subordinate_pass(ReWC,Threshold,EnList,output,pixels,T)
      global CodeAlphabet;
      CodeAlphabet.ZERO = 0; % binary 0
      CodeAlphabet.ONE  = 1; % binary 0
      CodeAlphabet.ZTR  = 2; % binary 00
      CodeAlphabet.POS  = 3; % binary 01
      CodeAlphabet.NEG  = 4; % binary 11
      CodeAlphabet.IZ   = 5; % binary 10

     d=0;
     if (Threshold>=T)
        for i=1:pixels
           d=EnList(:,i);
           temp=ReWC(d+1);
           [output, code]=input_code(output,1);
           if (code==CodeAlphabet.ONE)
              if (temp<0)
                 ReWC(d+1)=temp-Threshold/4*3;
              else
                 ReWC(d+1)=temp+Threshold/4*3;
              end
           end
        end
     end

  %  Add a new element FIFO buffer, "first in first out" buffer
  function EnFifo=put_in_fifo(s,EnFifo)
     EnFifo=[EnFifo,s];

   %  Get a element from FIFO buffer
  function [s,EnFifo]=get_from_fifo(EnFifo)
     s=EnFifo(:,1);
     EnFifo=EnFifo(:,2:end);
  %
  %  Builds a matrix element from dominant pass and a threshold
  %
  function [ReWC,s,output,EnList]=input_element(ReWC,Threshold,s,output,EnList)
    global CodeAlphabet;
    CodeAlphabet.ZERO = 0; % binary 0
    CodeAlphabet.ONE  = 1; % binary 0
    CodeAlphabet.ZTR  = 2; % binary 00
    CodeAlphabet.POS  = 3; % binary 01
    CodeAlphabet.NEG  = 4; % binary 11
    CodeAlphabet.IZ   = 5; % binary 10

    d=s(1);
    [output,code]=input_code(output,2);

    if (code==CodeAlphabet.POS)
       ReWC(s(1)+1) = 1; %add change
       EnList=[EnList, d];
    elseif (code==CodeAlphabet.NEG)
    %    ReWC(s(1)+1)=-1*Threshold; % change
       ReWC(s(1)+1) = -1; % add change
       EnList=[EnList, d];
    end
    s(2)=code;

   % Read a code from the input stream

  function [output, code]=input_code(output,count)

    global CodeAlphabet;
    CodeAlphabet.ZERO = 0; % binary 0
    CodeAlphabet.ONE  = 1; % binary 0
    CodeAlphabet.ZTR  = 2; % binary 00
    CodeAlphabet.POS  = 3; % binary 01
    CodeAlphabet.NEG  = 4; % binary 11
    CodeAlphabet.IZ   = 5; % binary 10

    temp=0;
    temp1=0;
    temp=output(1);
    output=output(2:end);

    if (temp==0)
         if (count==1)
            code = CodeAlphabet.ZERO;
         else
            temp1=output(1);
            output=output(2:end);
            if (temp1==0)
               code = CodeAlphabet.ZTR;
            else
               code = CodeAlphabet.POS;
            end
         end
    elseif (temp==1)
         if (count==1)
            code = CodeAlphabet.ONE;
         else
            temp1=output(1);
            output=output(2:end);
            if (temp1==0)
               code = CodeAlphabet.IZ;
            else
               code = CodeAlphabet.NEG;
            end
         end
  else
         disp('Something wrong in the input stream');
         return;
  end

  function [output, code]=input_code_change(output,count)

  global CodeAlphabet;
  CodeAlphabet.ZERO = 0; % binary 0
  CodeAlphabet.ONE  = 1; % binary 0
  CodeAlphabet.ZTR  = 2; % binary 00
  CodeAlphabet.POS  = 3; % binary 01
  CodeAlphabet.NEG  = 4; % binary 11
  CodeAlphabet.IZ   = 5; % binary 10

  temp=0;
  temp1=0;
  temp=output(1);
  output=output(2:end);

  if (temp==0)
         if (count==1)
            temp = output(1);
         else
            temp1=output(1);
            output=output(2:end);
            if (temp1==0)
               code = CodeAlphabet.ZTR;
            else
               code = CodeAlphabet.POS;
            end
         end
  elseif (temp==1)
         if (count==1)
            code = CodeAlphabet.ONE;
         else
            temp1=output(1);
            output=output(2:end);
            if (temp1==0)
               code = CodeAlphabet.IZ;
            else
               code = CodeAlphabet.NEG;
            end
         end
  else
```

```
    disp('Something wrong in the input stream');
    return;
end
```