

Real-time Intrapartum Fetal Electrocardiogram Analysis

By Mark Crittenden, M.Eng.

Department of Electrical and Electronic Engineering.

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy.

September 1997.

Contents

1 Introduction

1.1 Fetal monitoring during labour 1

1.2 FECG monitoring 3

1.2.1 The heart 3

1.2.2 ECG changes under stress 5

1.2.3 FECG system 7

1.2.3.1 Nottingham system 7

1.2.3.2 Cinventa system 10

1.2.4 Clinical trials 11

1.3 Summary of current FECG monitoring 13

1.4 Outline of research presented in this thesis 14

2 Development of the FECG analysis system

2.1 Requirements and solution 15

2.2 Instrumentation 16

2.2.1 Stage 1: Adaptation of present board 20

2.2.2 Stage 2: New prototype board 21

2.2.3 Stage 3: Final ECG board 26

2.3 Software 28

2.3.1 Microsoft Windows environment 28

2.3.2 Object Oriented Programming 29

2.3.3 Borland ObjectWindows 30

2.4 Software algorithms 32

2.4.1 R - peak detection 33

2.4.2 Time coherent averaging 39

2.4.3 Linear modelling 47

2.4.4 Parameter extraction 49

2.5 Software implementation 52

2.6 Discussion 55

3 FECG Simulation system

3.1 Introduction 56

3.2 ECG amplitude adjustment 59

3.3 ECG interval adjustment 62

3.4 HR adjustment 64

3.5 Addition of noise 64

3.5.1 Single frequency noise 64

3.5.2 White noise 67

3.5.3 Movement artefacts 69

3.6 Code 70

3.7 Hardware 72

3.8 Discussion 73

4 Validation of the FECG analysis algorithms

4.1 Introduction 75

4.2 R - peak detection 76

4.3 Time-coherent averaging 82

4.4 Linear modelling and parameter extraction 87

4.4.1 PR interval measurements 87

4.4.2 T/QRS measurements 88

4.5 Parameter extraction under noisy conditions 90

4.6 Discussion 98

5 Frequency content of the ECG and its relation to the design of the ECG front-end filter

5.1 Introduction 101

5.2 Theory 102

5.2.1 Modelling of HRV 104

5.2.2 Amplitude modulation 108

5.2.3 ECG spectrum 111

5.3 Results - Effects on FECG parameters 115

5.3.1 R-R interval 116

5.3.2 P-R interval 117

5.3.3 T/QRS ratio 119

5.3.4 Removal of mains frequency 121

5.3.5 Effect of current front-end filtering electronics 122

5.4 Morphological changes and discussion of results 124

6 Validation of the complete system

6.1 Introduction 130

6.2 Testing via use of the simulator 131

6.3 Use of database 132

6.4 Discussion 147

7 Conclusions and further work 148

References 152

Appendix I Analysis user guide 162

Appendix II Analysis software 167

Appendix III Simulator user guide 188

Appendix IV Simulator software 194

Appendix V Simulator algorithms 215

Abstract

The research within this thesis concerns the monitoring of the fetus during labour, using the fetal electrocardiogram (FECG). A versatile FECG analysis system was developed for the Microsoft Windows environment, to allow various FECG parameters to be extracted. Algorithms, currently used in other FECG analysis systems, were implemented using Object Oriented Programming, thus allowing new algorithms to be easily added at a later stage. Although these current algorithms have been demonstrated by several authors, it was felt that they had been used with only partial investigation of their limitations, and with failure to fully determine their accuracy in controlled conditions. These factors are fully addressed within this thesis.

By developing a FECG simulator, in which heart-rate, morphology, and noise levels could be varied, the ability of the analysis algorithms to extract the parameters, and the accuracy of these parameters under different noise conditions, were thoroughly checked. Both ability and accuracy were shown to be very good in ideal noiseless conditions; but, with the addition of noise, there exists a compromise between parameter accuracy when the morphology is static, and parameter accuracy when the morphology is changing.

The accuracies of the most common indices in this field (the Conduction Index, and the T/QRS ratio) were determined for different levels of simulated noise, and their values demonstrated for data previously recorded from the fetal scalp. Errors

as large as 0.3 in the CI and 0.05 in the T/QRS suggested that in the clinical environment, an indication of the accuracy of each index ought to be displayed, and this may be estimated from the measured level of noise. Furthermore, this analysis system allows the direct comparison of both indices.

Finally, in order to design a more effective front-end filter, it is important to be aware of the frequency content of the underlying FECG. The Integral Pulse Frequency Modulation (IPFM) model, combined with Pulse Amplitude Modulation (PAM), was used to estimate realistic frequency components within the FECG signal. The effects of filtering could then easily be modelled to show the distortion of both the FECG and any parameters taken from it. For a FECG front-end filter, distortion was found to be insignificant provided that, above 1 Hz, both the gain remained constant and there was no phase-distortion.

List of abbreviations

CI	Conduction Index
CTG	Cardiotocogram
ECG	Electrocardiogram
EFM	Electronic Fetal Monitoring
FBS	Fetal Blood Sample
FECG	Fetal Electrocardiogram
FHR	Fetal Heart Rate
FSE	Fetal Scalp Electrode
IPFM	Integral Pulse Frequency Modulation
IUP	Intra Uterine Pressure
PAM	Pulse Amplitude Modulation
RI	Ratio Index
SNR	Signal to Noise Ratio
STAN	ST-ANalyser

Acknowledgements

First, I would like thank both my supervisors, Dr. John Crowe and Dr. Barrie Hayes-Gill for their ideas and guidance throughout this work. I am most grateful to the Perinatal Research Group within the Department of Obstetrics and Gynaecology, particularly Dr. Briony Strachan, for supplying magnetic tapes of pre-recorded FECG data, which proved useful in demonstrating some of this work.

I would like to express my appreciation to my colleagues in the Medical Signal Processing and VLSI Laboratories within the Department, and other university friends, all of whom made this period at Nottingham very enjoyable, and motivated me during this project. Thank you all, especially Adrian, Andy, Ashley, Beatrice, Clare, Dale, Jeanette, Jon, Jules, Simon C, Simon S, and Tiranan.

Thanks must also go to my parents, and brother for their words of encouragement during the tough "writing up" period.

The project was funded by the EPSRC, as a CASE award, with Oxford Instruments as the sponsor.

1. Introduction

1.1 Fetal monitoring during labour

Of the 680,000 births that took place in England and Wales in 1993, the perinatal mortality rate was 0.76 %; approximately 4.3 % of these deaths being intrapartum related, of which about two-thirds may be attributed to intrapartum asphyxia [National Advisory Body, 1995]. Not only is asphyxia a cause of mortality and morbidity in the new-born, it also may lead to neurological disability, in particular cerebral palsy. Of the 22 cases of cerebral palsy found in a study of 13000 live-born children, 6 showed signs of intrapartum asphyxia [Grant et al., 1989]. Through intervention, some of these deaths and disabilities, may be prevented. To help the clinician decide if intervention is necessary, it is important to assess the condition of the fetus. Electronic Fetal Monitoring (EFM), is presently the main tool: continuous fetal heart rate (FHR) is interpreted together with uterine contractions, in the form of a cardiotocograph (CTG). Further information may be obtained through analysis of a fetal blood sample (FBS).

The electrical activity of the heart, known as the electrocardiogram (ECG) may be measured during labour, using a fetal scalp electrode (FSE), which is attached to the presenting part of the fetus.. From the ECG, the FHR is calculated, and, when shown together with the intrauterine pressure (IUP), the system is known as a cardiotocogram. Interpretation of a cardiotocogram involves monitoring aspects of the FHR and its relationship with an increase in the IUP. Furthermore, the FHR can also be found using ultrasound.

There is no doubt that intrauterine fetal death is often preceded by changes in the FHR, which could be detected by a continuous monitoring device [Sureau, 1994]. However, the CTG was introduced before large clinical trials to determine its use in assessing fetal well being were carried out. Concerns have arisen with the ability of the CTG to predict fetal condition, whether the signal is obtained via a FSE or by ultrasound [Sykes et al., 1983]. 70% of all claims in relation to fetal brain damage during labour, are based upon abnormalities of the CTG and electronic fetal monitoring [Symonds, 1994a]. Fear of such litigation has contributed to a rise in the number of caesarean sections, and if litigation continues to rise, obstetricians are more likely to deliver in this way [Broadhead and James, 1995]. The rise in caesarean section rates is a matter for concern, as the maternal risk of mortality is two to four times the figure when compared to a vaginal delivery [Consensus in Medicine, 1981]. Ironically, one argument to reduce the number of caesarean sections, is to keep low risk cases away from obstetricians! [Treffers and Pel, 1993].

Furthermore, much misinterpretation of the CTG occurs [Barrett et al., 1990, Murphy et al., 1990], and so to aid the obstetrician in the decision to intervene, analysis has been automated via computers [Sturbois et al., 1973, Chung et al., 1995, Keith et al., 1995]. There has also been an interest in the ECG: at first, ECG analysis was mainly considered as a means of reducing death, but now current research is also concerned with reducing the number of interventions, and the number of FBSs.

1.2 FECG monitoring

1.2.1 The heart

The heart is composed of two separate pumps: a *right heart* that pumps blood through the lungs, and a *left heart* that pumps blood through the peripheral organs [Guyton, 1991]. Each of these two separate halves contains two chambers, the *atrium*, and the *ventricles*. The atrium acts as an entryway into the ventricle, acting as a blood reservoir, but also pumps the blood into the ventricle. The ventricle is the main pump that then propels the blood through either the pulmonary or peripheral circulation.

Rhythmical impulses are generated from a small, specialised muscle, known as the sinus node, within the heart. These conduct rapidly through both atria, and into the ventricles (via the A-V bundle), causing contraction of the heart muscle. As this impulse passes through the heart, electrical currents spread into the tissues surrounding the heart, and a small proportion, to the surface of the body. By placing electrodes on the skin on opposite sides of the heart, one can measure a proportion of the electrical potential across the heart. The recording of this is called the *electrocardiogram* (ECG). A typical ECG can be seen in Figure 1.1.

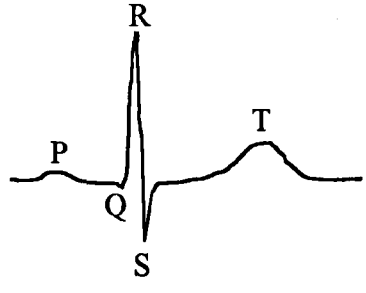


Figure 1.1: Component waves of the electrocardiogram

The ECG is composed of the P, Q, R, S and T waves, with the Q, R and S waves known collectively as the QRS complex. The P wave is caused by depolarisation of the atria, prior to contraction. As the depolarisation wave spreads through the ventricles, the QRS complex is generated. The T wave is due to ventricular repolarisation.

In 1906, Cremer attached an intravaginal electrode and an abdominal electrode to a pregnant patient, and connected them to a galvanometer [Cremer, 1906]. This gave a paper trace showing both the fetal and maternal QRS complexes. However the resolution of the fetal ECG (FECG) was very poor.

The problem of clear definition of the FECG was not just a need for amplification of the signal, but also that the signal is immersed in electrical noise, arising from many sources [Jenkins, 1986], such as mains interference; noise at the interface between the fetal electrode and the skin; the maternal electrocardiogram; the fetal electromyogram; the maternal electromyogram; and the fetal electroencephalogram;

By the 1960's, advances in electronics meant that amplification of the signal was no longer a problem. Amplification of the underlying FECG, inevitably amplified the noise. To reduce this noise, and remove the maternal complex, an electrode could be attached directly to the presenting part of the fetus, instead of the maternal abdomen [Hunter et al., 1960, Hon and Lee, 1963a, Hon and Lee, 1963b]. Work was also carried out to design a scalp electrode, to further reduce

the noise, and improve the FECG resolution [Hunter et al., 1960]. An averaging technique, known as time-coherent averaging was also introduced, whereby several complexes contributed towards an average waveform, using a computer [Hon and Lee, 1963b, Hon and Lee, 1964]. This improved the signal-to-noise ratio (SNR), so that even the P and T waves could be seen.

In a study of the FECG in 234 labours, using Hon's electrodes to obtain the signal, the FECG was passed to a computer to enhance the FECG waveform using the same averaging technique [Pardi et al., 1974]. The averaging process gave a good improvement in signal quality, so that wave components were clearly defined. Few enough waveforms were averaged that transient changes in the waveform could be detected. This now allowed detailed analysis of the FECG morphology.

1.2.2 ECG changes under stress

Changes in the T wave have been known for some time during hypoxia in adults [Kahn and Simonson, 1957]. This led to an interest in the fetal T wave during labour. Much research was carried out in the evaluation of changes in the ST waveform as a method for fetal surveillance.

To compensate for the changes in FECG signal strength, it was proposed that ST waveform changes could be identified by measuring the T/QRS ratio [Greene et al., 1982]. This was calculated as the ratio of the T wave amplitude to the QRS complex amplitude (*see* Figure 1.2). Experiments with fetal lambs [Greene et al.,

1982, Rosen, 1986, Greene and Rosen, 1989] and human fetuses [Lilja et al., 1985, Arulkumaran et al., 1990] showed there was an increase in the T/QRS ratio, associated with fetal distress.

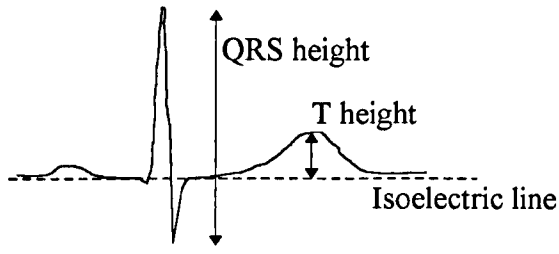


Figure 1.2: Derivation of the T/QRS ratio

Much work has been carried out to determine the range of T/QRS ratios in a healthy fetus [Newbold et al., 1989, Lilja et al., 1989, Newbold et al., 1991] and the effect of contractions on the ratio [Thaler et al., 1988, Newbold et al., 1995]. The T/QRS ratio is raised during contractions, but the significance of this increase remains uncertain.

Other research concerned the P wave and the PR interval (the time between the P wave peak and the R-peak). In a study of the FECG of 110 patients during labour, it was found that towards the end of labour, the P-R interval was up to 10% shorter during contractions, and the P wave amplitude fell by 30% over the last hour [Marvell et al., 1980]. The ST segment displayed no significant changes.

An evaluation was carried out into the patterns of changes of FECG variables, in response to changes in FHR [Family, 1982]. Using statistical techniques, a strong inverse relationship was observed between the PR interval, and the FHR, in the

clinically normal fetus. However, the opposite was true for clinically abnormal fetuses.

1.2.3 FECG systems

The interest in the FECG can be broken into 2 main areas: Firstly, research based around Nottingham, showed most interest in the PR interval [Symonds, 1994b]. Secondly, research based on a system developed in Sweden [Cinventa], which was interested in the amplitude of the T wave. Development work using this machine continued at the Perinatal Research Group, Plymouth General Hospital [Rosen and Luzietti, 1994].

1.2.3.1 Nottingham system

The first Nottingham system was the Surveillance for Asphyxia with the Fetal Electrocardiogram (SAFE) system, a computer-based system, to reduce interfering noise [Shield, 1977]. This original system was in two parts: one for the data acquisition and storage of the data; the second to retrospectively process the recorded FECG data, using a Honeywell DDP 516 16-bit minicomputer. Using this system, changes were made to the data processing software to further enhance the signal [Marvell, 1979, Marvell and Kirk, 1980].

The first generation of monitor at Nottingham was not capable of on-line analysis, as it was too large to be moved into a labour suite. The operation of the system and presentation of results were complex, and not suitable for use by the clinician. If such a monitoring system were to be of use, it would need to be used on-line. The second-generation system was, therefore, based on a smaller DEC LSI-11/23

minicomputer, with dedicated graphical display and an adapted Sonicaid FM3R fetal heart rate monitor as the front-end [Smith, 1983].

The Conduction Index

Using the DEC minicomputer developed by Smith, the fetal electrocardiograms from 309 fetuses during labour were studied [Murray, 1986, 1992]. The P-R and R-R intervals correlation was found to change with increasing fetal acidosis. Murray termed the correlation between the PR interval and the FHR as the *Conduction Index* [Lui, 1989].

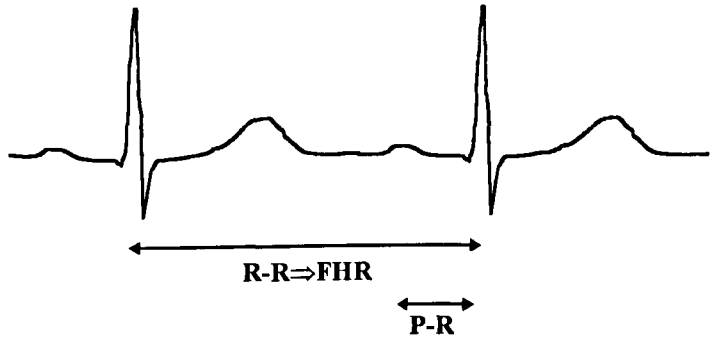


Figure 1.3: The parameters used to calculate the Conduction Index

The PR and FHR were found every second for the last 5 minutes. The conduction index (CI) is then the Pearson's correlation coefficient of this data set:

$$r = \frac{N\sum_{i=1}^{N} X_i Y_i - \left(\sum_{i=1}^{N} X_i\right)\left(\sum_{i=1}^{N} Y_i\right)}{\sqrt{\left\{N\sum_{i=1}^{N} X_i^2 - \left(\sum_{i=1}^{N} X_i\right)^2\right\}\left\{N\sum_{i=1}^{N} Y_i^2 - \left(\sum_{i=1}^{N} Y_i\right)^2\right\}}}$$

where N=300, $X_1 \ldots X_N$ are the previous PR intervals and $Y_1 \ldots Y_N$ are the previous FHR. However, the Conduction Index system presently being used, takes only the P-R intervals and FHRs, measured from averages obtained every 2 seconds, over

the last 2½ minutes [Sahota, 1997]. A CI value, remaining positive for more than 20 minutes, was considered clinically significant [Murray, 1992].

The third generation of monitor built at Nottingham, was based around a signal processing board containing a National Semiconductor NS32016 32-bit processor, and included four other boards for isolation, ADC conversion, control, and graphical output [Lui, 1989]. The previous monitor only calculated FECG parameters every 15 seconds, but the increase in processing power now allowed a beat-to-beat processing of the FECG. The NS32016 was later incorporated into a 386 IBM compatible personal computer (PC), which handles the graphical output and data storage. The Sonicaid FM3R was again used as the front-end. This system measures 22 parameters, associated with the morphology and time constants of the waveform [Symonds, 1994b].

The Ratio Index

The CI was subject to short-term fluctuations, and the importance of these short-term swings from positive to negative was unknown. The CI was criticised for having no temporal basis, which would allow quantification of the changing relationship [Mohajer et al., 1994]. The latest system was used to obtain the FECG, and then develop a new index, called the Ratio Index (RI). The RI is calculated as follows:

Firstly the FHR and PR interval were calculated every 10 seconds over the whole labour. The mean (Fhr_{mean}) and standard deviation (Fhr_{sd}) of the FHR were

calculated as well as the mean (Pr_{mean}) and standard deviation (Pr_{sd}) of the PR interval. The individual values, for each 10-second period, were then transformed, using the standard deviate transform:

$$Z_{Fhr} = \frac{(Fhr_t - Fhr_{mean})}{Fhr_{sd}}$$

where Z_{Fhr} is the transformed value for fetal heart rate and Fhr_t is the current value of fetal heart rate at time t.

$$Z_{Pr} = \frac{(Pr_t - Pr_{mean})}{Pr_{sd}}$$

where Z_{Pr} is the transformed value for P-R interval and Pr_t is the current value of P-R interval at time t.

The transformed product (ZTP) was then calculated as:

$$ZTP = Z_{Fhr} \times Z_{Pr}$$

The mean and standard deviation of all ZTP values are calculated. The total number of ZTPs, which were in excess of 2 positive standard deviations above the mean was found. The Ratio Index was the percentage of times this occurred, as a percentage of total recording (labour) time. A 4% level of the RI was chosen for identification of hypoxia.

1.2.3.2 Cinventa system

In 1989, the ST ANalysis (STAN) system was produced [Cinventa], and it was used to collect 201 FECG recordings during labour [Rosen and Lindecrantz, 1989]. An input for uterine activity was also included. Output was via a printer,

which gave the usual CTG trace and a trace of the T/QRS ratio, and a graphical display.

As the Nottingham system also measured T/QRS ratio, a comparison between the Nottingham and STAN systems, was carried out and showed some disagreements in the T/QRS ratios [Skillern et al., 1994]. These were attributed to way the two systems measured T height. The Nottingham system measured T height from the T-P region, whilst the STAN system measured it from the P-R segment (*see* Figure 1.4).

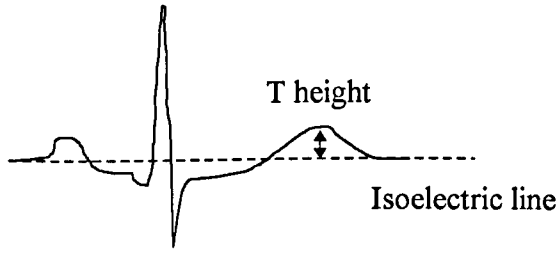


Figure 1.4(a): T height calculated by Nottingham system

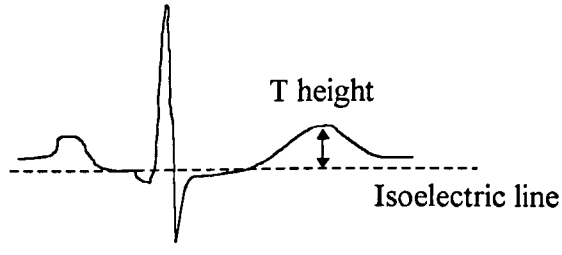


Figure 1.4(b): T height calculated by STAN system

1.2.4 Clinical trials

In Plymouth, a large randomised trial of 2400 high risk labours was carried out using the STAN monitor [Westgate et al., 1992, Westgate et al., 1993, Rosen and Luzietti, 1994]: the STAN clinical guidelines are summarised in Table 1.1. Not only was the T/QRS ratio used in this trial, but also the occurrence of ST

depression, with biphasic negative T-waves. The trial tested the hypothesis that the combination of ST waveform and CTG analysis, compared with CTG analysis only, would reduce operative interventions for fetal distress, without placing the fetus at risk. During ST and CTG analysis, they found a highly significant reduction of 46% in operative deliveries for fetal distress, with no difference in deliveries for other reasons. ST waveform analysis could also reduce the number of FBSs taken [Johanson et al., 1992, Cockburn, 1992].

	T/QRS		
CTG	Normal	High and stable	Negative or rising
Normal	No action	No action	FBS/deliver
Intermediate	No action	FBS/deliver	Deliver
Abnormal	FBS/deliver	Deliver	Deliver

Table 1.1: Basic summary of clinical guidelines for the STAN

In a retrospective study, clinical guidelines using the RI, the CI and EFM were established [Reed et al., 1996]. An RI greater than 4% and a CI positive for more than 20 minutes were again classed as abnormal. Intervention was deemed necessary if two of the three variables were abnormal. The proportion of cases of missed acidosis at delivery would have been reduced from 8.5% to 4.5%, and the number of FBS reduced from 85.5% to 26.8%.

In a study of 214 cases, the addition of the CI and RI to EFM, was compared to monitoring EFM alone [vanWijngaarden et al., 1996]. The criteria for this are

shown in Table 1.2. Should the EFM become acutely abnormal, management was allowed on the EFM only (*see* * in table). It was shown that the addition of this FECG analysis to conventional monitoring, could reduce the number of patients undergoing FBS significantly, without an increase in adverse outcome.

	EFM normal		EFM abnormal	
FECG criteria	RI < 4% or CI + < 20 min	RI >= 4% and CI + >=20 min	RI < 4% and CI + < 20 min	RI >= 4% or CI + >=20 min
ACTION	NONE	FBS or DELIVERY	NONE*	FBS or DELIVERY

Table 1.2: Management criteria for EFM, RI and CI

1.3 Summary of current FECG monitoring

The CTG has become the main tool for fetal monitoring, and, although the technique has been available for many years, analysis of the FECG has not yet become popular. To become widespread, FECG analysis needs a clear clinical protocol for when intervention should take place, which should be incorporated into the system; thus the system would clearly show whether intervention was necessary.

FECG analysis can be broken into 2 stages: the first involves obtaining *measured* parameters from the ECG, such as PR interval and T/QRS height. Secondly *derived* parameters, such as the Conduction Index and Ratio Index, can be calculated from these measurements.

However, there are many unanswered questions, regarding the technique of FECG analysis. Firstly, there appears to be different philosophies as to the purpose of FECG, from saving lives, to assisting the clinician in the decision whether to intervene or obtain a FBS. There are many views about the correct bandwidth of signal, needed in the front-end filtering. Fetal orientation and electrode position must affect the FECG signal, as well as choice of scalp electrode type [Fisher, 1993, Westgate et al., 1990]. Furthermore, even when the parameters have been calculated, there appears no indication as to the accuracy of the values. It is unknown how the derived parameters will be affected by this filtering process. Therefore, this thesis aims to investigate some of these questions.

1.4 Outline of research presented in this thesis

This investigation was carried out by firstly designing a versatile FECG analysis system, which is described in Chapter 2. The analysis system then needed to be tested to check the ability of the currently used algorithms to accurately calculate the measured and derived parameters, and asses the accuracy under different noise conditions. A novel FECG simulator was developed for this task, and is presented in Chapter 3. Testing the software of the analysis system, to determine the limitations of these algorithms, is then shown in Chapter 4. To ascertain the frequency response required within the system hardware, a theoretical analysis of filtering is presented in Chapter 5. Both software and hardware validation of the analysis system is shown in Chapter 6. Finally, in Chapter 7, conclusions to the work are presented about the current state of analysis, along with suggested future improvements.

2. Development of the FECG analysis system

2.1 Requirements and solution

The work presented in this Chapter concerns the development of a versatile system to provide the Conduction Index and T/QRS ratio. This would be made within a platform that would allow other analyses of the electrocardiogram, such as the RI, to be readily produced as required. The system is required to extract and display both parameters at the same time, with the versatility of extracting further parameters within a short development time.

Both parameters could be obtained, by connecting both existing STAN and Nottingham systems to the fetus, but this would not be a practical solution. Furthermore, the systems would have to be synchronised so that data sets could be compared, and the data would have to be transferred to a third machine to compare the parameters. No extra parameters could be studied. Initially, the algorithms, presently used in these systems, would be implemented.

Rather than design a complete FECG analysis system, it was decided to connect a commercially available CTG monitor to a PC (*see* Figure 2.1). The CTG monitor would isolate, amplify, filter and digitise the raw FECG signal from the scalp electrode, before sending the data to the PC via a serial RS232 link, chosen for its simplicity and widespread availability. This would reduce the time required to develop the front-end signal conditioning. Also, as medical staff are familiar with CTG systems (and some with the chosen system), they will be more familiar with

this system. The Sonicaid Meridian 800 [Oxford Instruments] was chosen due to the facts that it was both a widely-used machine, and that it was made by the sponsors of the author, and so could be readily adapted.

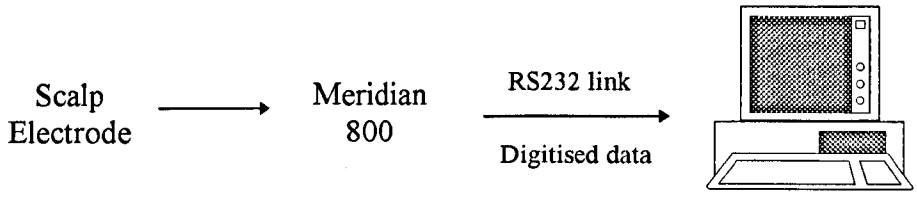


Figure 2.1: Overview of FECG analysis system

Due to the widespread use of PCs, and the fact that a customer may already have an unused PC, it was decided to carry out all the signal processing and display on a PC. Performing the analysis on a PC confers several benefits on the analysis system, since their widespread availability ensures ease of standardisation, regarding data formats, storage media, hardware, etc. Furthermore, if a customer already has an unused PC and a Meridian 800, the cost of installing the system would be significantly less than installing a whole new system. After modifications within the Meridian 800, the task becomes one of software development, and upgrades become both simplified and quick.

2.2 Instrumentation

The Sonicaid Meridian 800 (*see* Figure 2.2), produced by Oxford Instruments, is a monitoring system for use during either the antepartum period or management of labour [Oxford Instruments, 1992]. FHR monitoring can operate with 2 inputs, these being:

1. Ultrasound/FECG
2. Ultrasound/MECG
3. Ultrasound/Ultrasound [option]
4. FECG/MECG [option]

The dual heart rate monitoring on the Meridian 800 allows simultaneous monitoring of two heart rates - twins or maternal and fetal. This option requires a third board to be added inside the Meridian. In a similar way, in this project a third board has been added for the hardware required to isolate, filter, and digitise the signal before sending the data to the PC. As an additional option an intra-uterine pressure (IUP) transducer can also be provided.

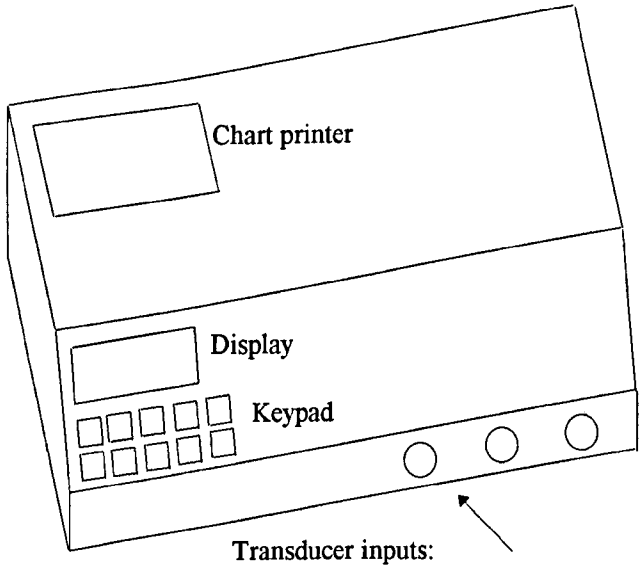


Figure 2.2: Meridian 800

The 3 main outputs from the system are:

1. A 2-line, 20-character per line, vacuum fluorescent display used for all the mode and heart-rate display functions.
2. A thermal dot-matrix chart printer, printing its own graticule on blank paper, on top of which are printed the heart-rates and IUP.
3. An audio output, which can be switched to either ultrasound or ECG channel on request.

However, for this application two main changes need to be made to a normal ECG board (*see* Figure 2.3). The first is the bandwidth of the filter used in the Meridian. As a CTG is only looking for an R peak, the band-pass filter has been designed to only pass those frequencies associated with an R-peak. The -6 dB points of the Meridian are 10 Hz and 100 Hz. However in this application, the whole *undistorted* ECG is required. Therefore the bandwidth must be increased.

In the case of adult ECGs, the American Heart Association (AHA) [Pipberger et al., 1975] recommend a maximum attenuation of 1 dB at 0.14 Hz and 30 Hz and 3 dB at 0.05 Hz and 60 Hz. The phase response should be linear over this latter region. It has been suggested that, because of the higher heart rate of the fetus, a higher 3 dB cut-off frequency of 0.16 Hz is tolerable [Marvell et al., 1980].

The STAN system has a bandwidth from 0.07 Hz to 100 Hz [Arulkumuran et al., 1990]. The Nottingham system has a reported bandwidth from 0.16 Hz to 500 Hz [Mohajer et al., 1994], although, as the signal is sampled at 500 Hz, it must be

assumed that an anti-aliasing filter is included so that the higher cut-off frequency is below 250 Hz. There appears to be much confusion into the necessary front-end filtering, and this will be discussed in detail in Chapter 5.

The second hardware change required was to add RS232 communication, so as to send the digitised data to the PC.

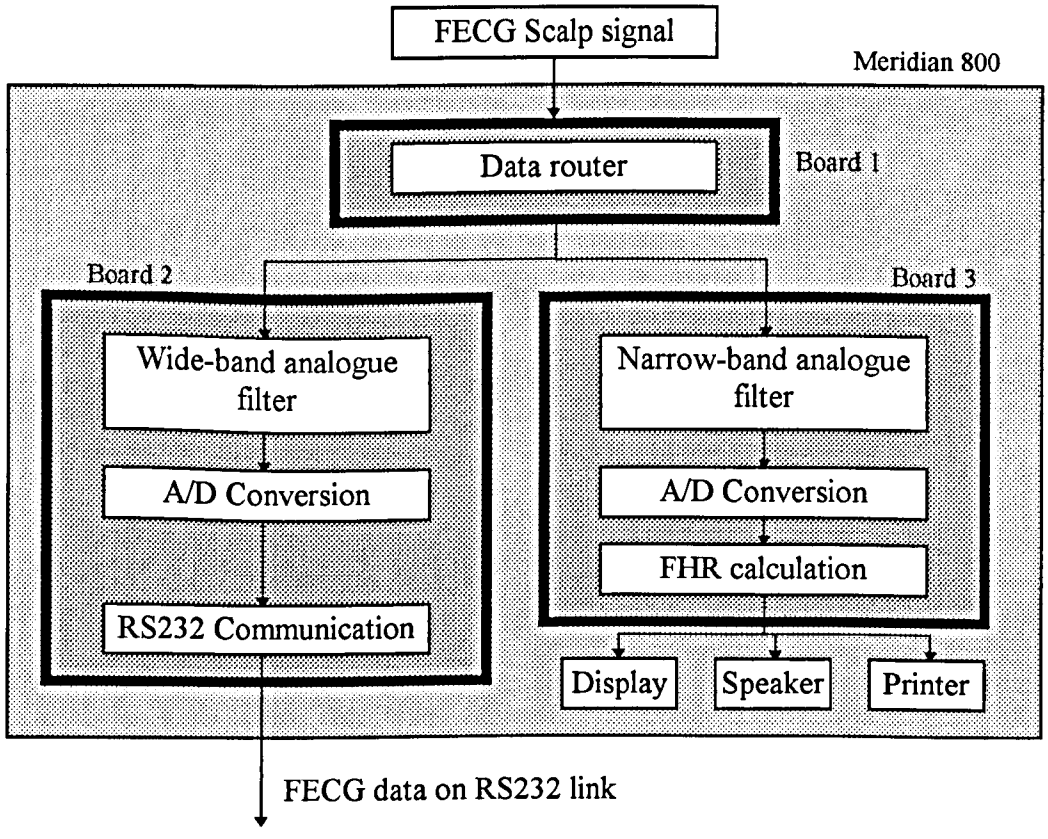


Figure 2.3: Board operations within Meridian

For additional patient safety, an optical link on the RS232 port, would be used to perform the task of electrically isolating the Meridian from the PC. This would

ensure that the equipment meets the safety criteria specified by British Standard BS5724.

The modifications to the Meridian were carried out by the manufacturer, Oxford Instruments. The author contributed ideas to the design, carried out the final testing, and aided in the fault detection. The modifications were done in 3 stages, which are detailed below.

2.2.1 Stage 1: Adaptation of present board

Components on a standard FECG board were altered by Oxford Instruments to increase the bandwidth. In addition, communication hardware was added to transmit the FECG data. 12 bit data was sent in 2-byte words at a sample rate of 500 Hz. Using both start and stop bits, but no parity bit, the rate of transfer of data was 10 kbits per second (10×2×500). The lowest standard baud rate, capable of this data rate was 19200 baud. At this stage, the port was not isolated, but this could be achieved with an RS232 optical link, connected from the Meridian to the PC. The Amplicon *Model 490* self-powered fibre optic RS232 link [Amplicon] was used, due to its relatively low cost, and the fact that the model derived its power from the RS232 port itself.

The FECG board was tested, and the following problems were found:

a) A notch filter could be clearly seen at 50 Hz, effecting both gain and phase. As the frequency spectrum of a FECG complex contains 50 Hz, it was proposed that if the FECG signal was passed through a notch filter to reduce mains

noise, the waveform would also be distorted. Chapter 5 details the importance of all the ECG components, including the mains frequency. Figure 2.4, however, summarises the importance of this 50 Hz frequency component. The component has been removed in 2.4(b), and has been altered in phase by 180^0 in 2.4(c). Such distortion is unacceptable, hence the notch filter must be removed.

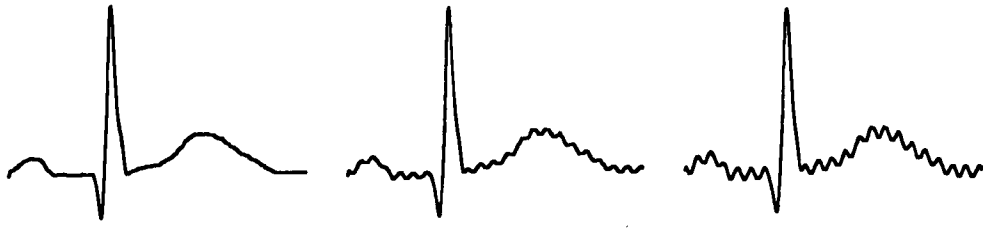


Figure 2.4: (a) Unfiltered (b) 50 Hz component removed (c) 180^0 phase shift

b) The pass-band has cut-off frequencies (-3 dB) at 0.6 and 17 Hz, which is of insufficient bandwidth.

c) The phase response over the whole bandwidth is not linear.

2.2.2 Stage 2: New prototype board

A new prototype board was built by Oxford Instruments, with a wider pass-band and isolated communications hardware. Several designs were considered to allow a wider pass-band. Figure 2.5 shows one such design. The signal from the FSE is high-pass filtered at 0.05 Hz to remove baseline drift. This is amplified, and passed to an A/D converter. The PIC microcontroller takes this signal, and sends it to a PC with the use of a Max232 communication chip. The

high-pass filter can be replaced by circuitry containing a low-pass filter, as in Figure 2.6: this is the same as using a high-pass filter. A more elaborate scheme is shown in Figure 2.7. Initially, a DC level is removed from the signal. Once the signal has been digitised, the microcontroller checks to see if the signal is close to the supply rails. If it is, the DC level is altered to bring the signal back between the rails. The microcontroller keeps track of any changes, and passes the adjusted digitised data to the communication chip.

The final design was a combination of Figures 2.6 and 2.7. Figure 2.8 shows a block diagram for the board. The signal from the FSE is passed through an instrumentation amplifier; this has an output relative to ref. In turn this is passed to an anti-aliasing filter, before the A/D conversion. The signal is also passed through a low-pass filter, with a cut-off frequency below 0.05 Hz. This changes the reference to the instrumentation amplifier, to remove low-frequency drift. The microcontroller checks to see if the signal has saturated, and reached one of the supply rails. If it has, a signal is sent to the low pass filter, via the up and down connection, causing the reference to change more quickly. The digital ECG signal is sent from the PIC, to the RS232 communication circuitry, which sends the data to the PC.

The FECG data was transmitted from the Meridian via an optical cable and connected to the PC RS232 port. A test ramp signal (from 8000_H to $83FF_H$) could be transmitted by the board, by setting an internal switch.

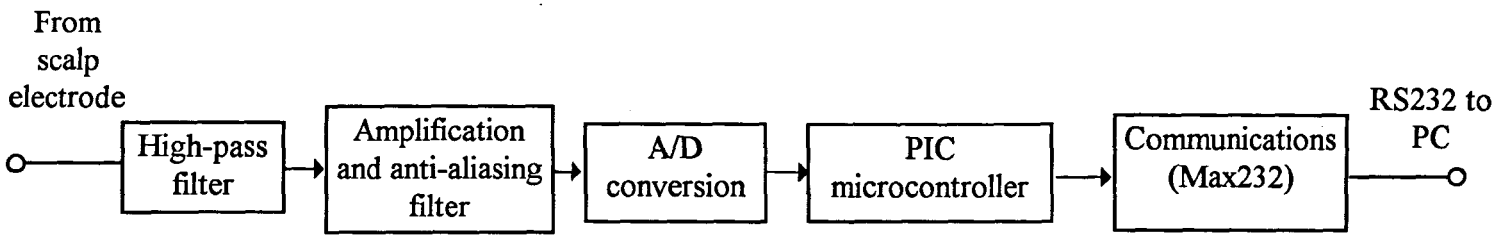


Figure 2.5: Block diagram of front-end, using high-pass filter to remove drift

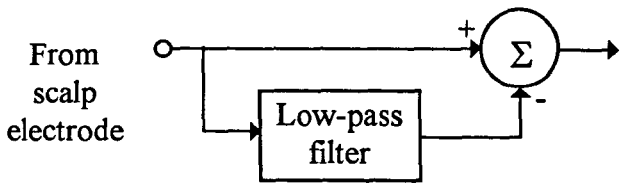


Figure 2.6: Use of low-pass filter

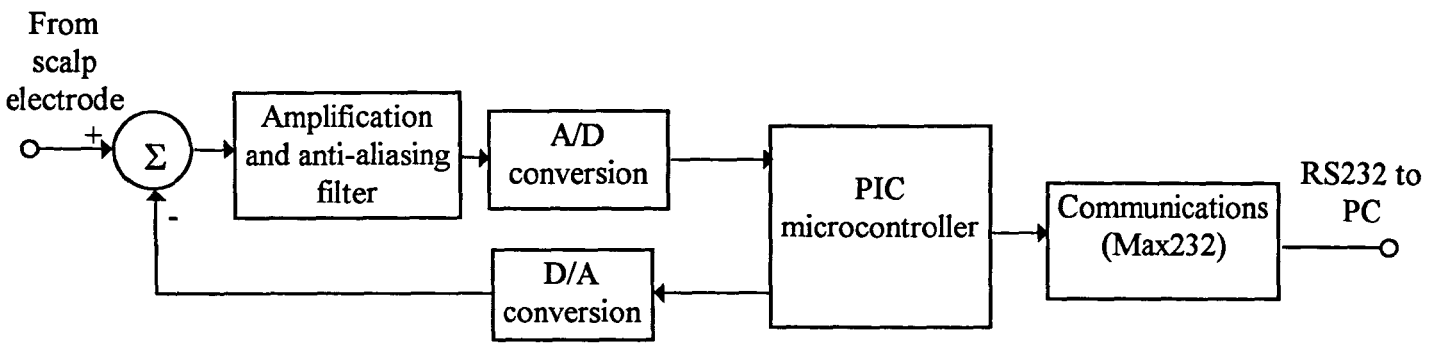


Figure 2.7: Removal of DC level through use of a D/A converter

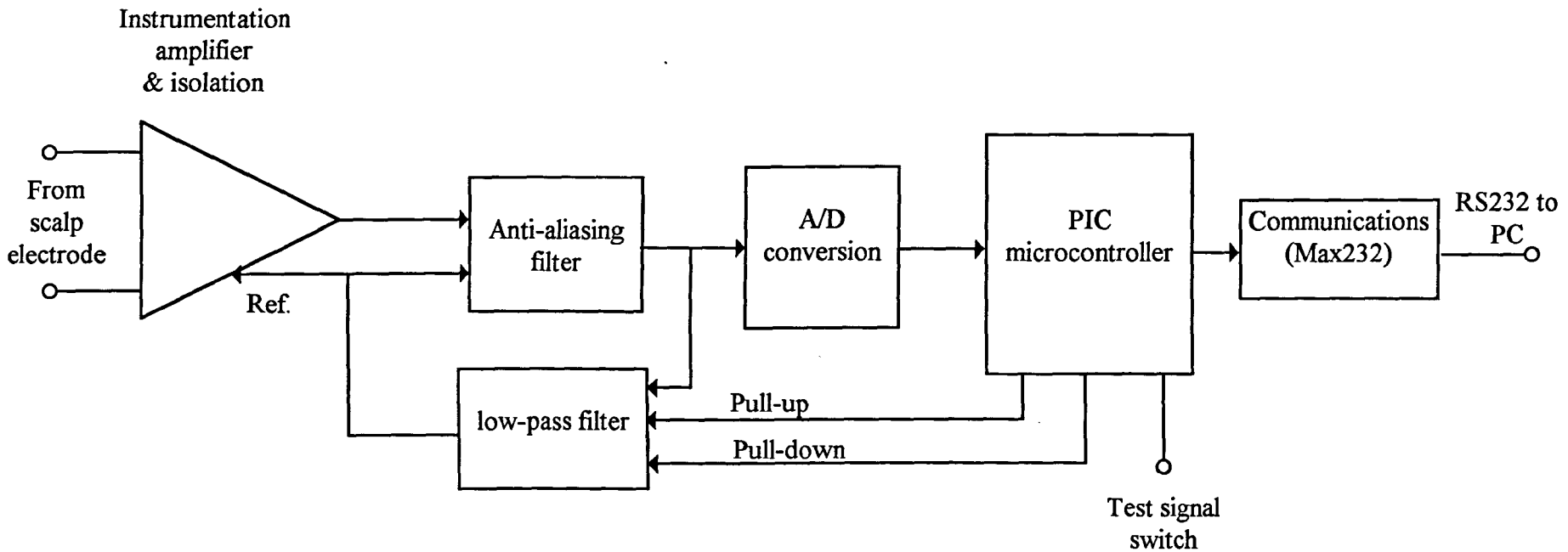


Figure 2.8: Block diagram of stage 2 and stage 3 front-end boards

Testing of the bandwidth gave:

- 1 dB cut-off frequencies at 0.02 Hz and 110 Hz
- 3 dB cut-off frequencies at 0.015 Hz and 200 Hz
- A near linear phase response between 0.05 Hz and 60 Hz

This conformed with the AHA recommendations.

The test signal was then used to test the link, and two problems were found:

1. The Interrupt Service Routine (ISR) for hard-disk access takes longer than the time between successive bytes arriving at the COM port (1 ms). Therefore, whenever disk access occurs, a new byte arrives at the port before the last byte can be read, causing data to be lost. The problem can be solved in 2 ways.

   The first solution, was to stop **all** disk access during processing, but as disk access was required to store both the raw data and FECG parameters, this was unacceptable. The second was to install a high speed UART into the PC. The 16550 UART has two 16 byte FIFOs to buffer both input and output data. The disk ISR is carried out within 16 ms (i.e. 16 bytes of data), so that no data is lost, and the problem solved. This does, however, create an additional problem. Namely, any PC on which this software is to be installed, must have an existing high speed COMs port, or have one added[1]. Greenleaf software was used to communicate with the COMs port [Greenleaf].

[1] The cost of these PC cards is small, at approximately £20.

2. The second problem was that every 70 seconds, data was lost from the Meridian. To test the origin of the problem, a second PC was used to send this data, and so emulate the Meridian. In this case, no error occurred. This indicated a problem within the Meridian occurring every 70 seconds. Further investigation of the problem showed that the prototype data capture board takes its power from the original ECG circuit on the Meridian analogue board. With no input to the original board, the software tests for a faulty electrode connection, occurring every 70 seconds. To test the connection, power to the isolated section of the ECG circuit is switched off to restart the test sequence. This normally does not matter as, by definition, there is no good signal to lose. But the wide-band ECG will lose its power at the same time, and will restart the test ramp when the power supply is resumed. This could be resolved by ensuring that there is a good ECG input signal, so that the analogue board does not initiate an electrode test. The next version, would not be powered from this board, so this would not be a problem.

2.2.3 Stage 3: Final FECG board

The prototype board was finally produced on a printed circuit board (PCB) by Oxford Instruments. The data transmission from the Meridian to the PC was also checked again. No bytes were lost if a 16550 UART was used. The gain and phase responses were rechecked for this board, using a Voltech Frequency Analyser [Voltech] and these are shown in Figure 2.9. The front-end frequency response will be discussed in detail in Chapter 5.

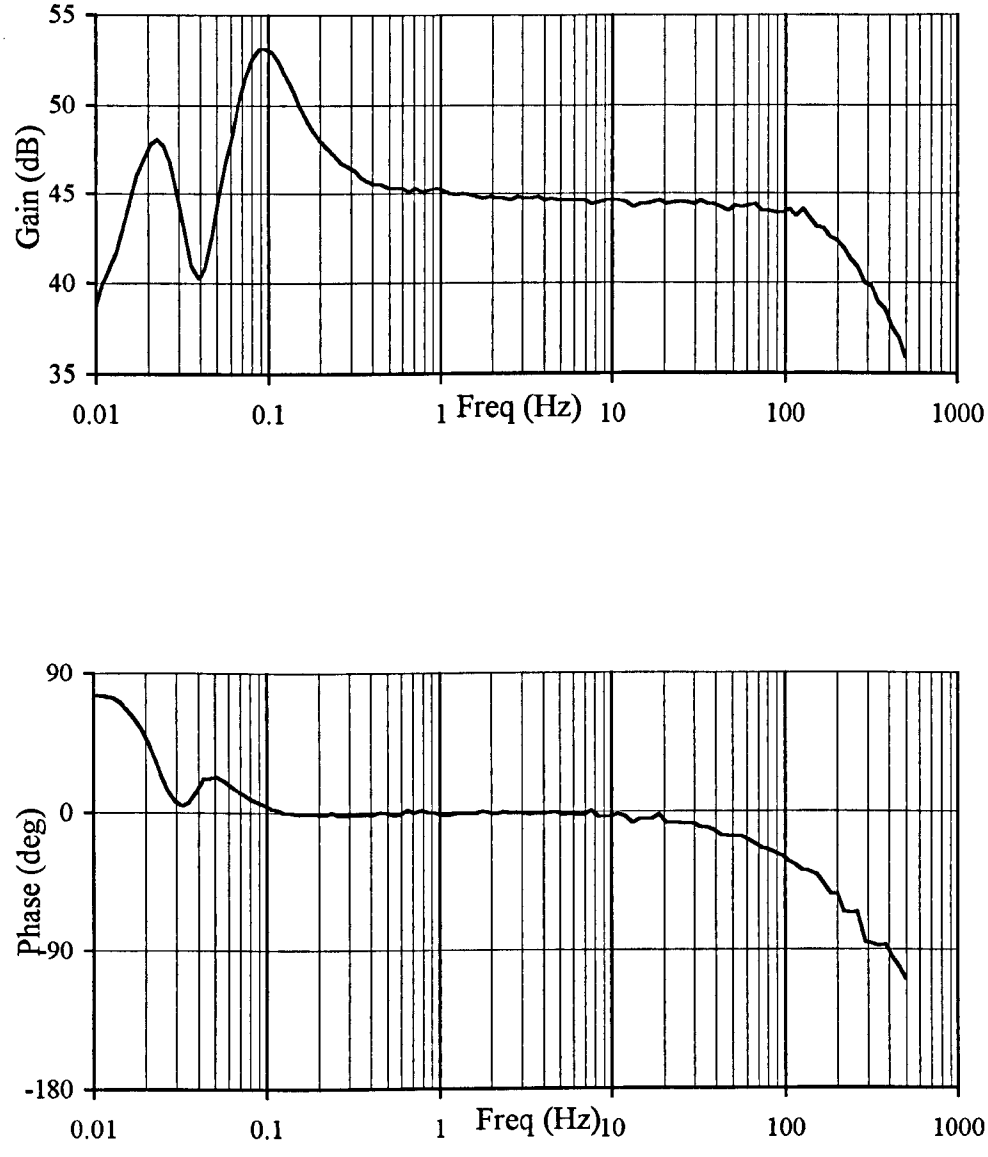


Figure 2.9: Frequency response of the FECG board

2.3 Software

2.3.1 Microsoft Windows environment

It was decided to display the output within the Microsoft Windows environment, to make program use both intuitive and obvious. Windows is a Graphical User Interface (GUI) environment for computers using the Microsoft Disk Operating System (MS-DOS). The use of Windows offers many advantages to both user and programmer [McCord, 1992]. Each window within the Microsoft Windows environment shares the same basic features. This similarity amongst applications, allow the user to adapt more easily to new applications. Furthermore, within the GUI, graphics-based images represent applications, functions and data. The use of a mouse to click on these images, allows easier control within a program. In respect to the FECG system, the popularity of Windows improves the likelihood of the user being able to immediately interact with the system.

The advantages to the user become advantages to the programmer. As the interface within every application is basically the same, the programmer can spend less time on the interface design, decreasing development time. The GUI allows graphical representation of functions and features, such as pop-up menus, so design is easier. Windows allows device-independent graphics to be developed. Well-designed Windows programs do not access the graphics hardware directly, but through the Graphics Device Interface (GDI) and Windows device drivers, which translate general drawing commands into precise actions on the relevant output devices. Thus, the application can operate with any display or printer which has a Windows device driver. The programmer does not have to provide device

drivers for all possible video displays and printers. Windows provides direct mouse and keyboard support, further decreasing development time. Windows also contains memory management, allowing more memory to be accessed than using traditional MS-DOS based applications.

2.3.2 Object Oriented Programming

Writing the software in C++ further decreases development time, through the use of Object Oriented Programming (OOP) [McCord, 1992, Borland International, 1991]. The computer environment is modelled as objects, which interact with each other by sending messages. Objects contain both properties and behaviours. The properties are not directly accessible from outside the object, but can be manipulated through the behaviours of the object, which are carried out when a message is received by the object. As these objects contain both properties and behaviours, the objects make the programming more modular, which improves ease of development and maintainability of code.

Other advantages of OOP involve the following properties-

1. Encapsulation - an object contains both data and code that manipulates that data. Within an object, part of the code and data may be private and inaccessible outside the object. In this way, an object provides protection against accidental modification, or incorrect use.
2. Polymorphism - one name can be used for several related, but slightly different tasks, allowing one name to be used for a general case of action.

3. Inheritance - the process by which one object can acquire the properties of another object. Within the Windows environment, this insulates the user from the details of the Windows programming. Programs can be thus written with less time and effort.

2.3.3 Borland ObjectWindows

This left the choice between Microsoft and Borland ObjectWindows. Research within the Department of Electrical and Electronic Engineering at Nottingham University [Huang et al., 1994] had previously been carried out into the real-time analysis of the FECG from abdominal recordings. Code had been written in Borland ObjectWindows to perform processing and display for this related work. This code could be adapted for this research. Therefore, it was decided to continue using ObjectWindows, so that this code could be adapted.

ObjectWindows [Borland International, 1991] is an object-oriented library, taking the advantages of OOP to make the programming of Windows applications easier. The Windows elements, including the windows themselves, are treated as objects. The objects encapsulate data that every window requires, respond to common Windows messages and events, and perform common windows operations. The important features of ObjectWindows are:

- **encapsulation of windows information** - although ObjectWindows defines the behaviour, attributes and data for windows, dialog boxes and controls, functions within the Windows Application Programming Interface (API) must

implement the physical representation. By storing parameters for these functions, access to the windows become easier.

- **abstraction of many of the Windows functions** - Windows applications get their appearance and behaviour by calling the required functions from the 600 available from the Windows API. Each function may require several different parameters. Although these functions can be called directly, the task is simplified through the use of ObjectWindows object member functions that abstract many of the function calls. Related API functions are also grouped into single member functions of an object to perform higher-level tasks. Many of the parameters for the Windows calls are stored in the object, and passed to the Windows function, when the ObjectWindows function is called.
- **automatic message response** - within ObjectWindows, Windows messages are handled by object member function calls. A member function can be defined for each message that must be handled. When the object receives a message, the appropriate member function is called automatically.

The Borland C++ compiler provides all the tools required to produce working Windows programs. Claims of up to 50% increased productivity have been made when using Borland C++ in place of Microsoft C [Roetzheim, 1992].

2.4 Software algorithms

The signal obtained from the Meridian, contains a significant amount of noise within the frequency band of the ECG signal. If one simply tried to filter out this noise, using bandpass and notch filters, one would distort the waveform. The technique of time-coherent averaging is presently used in both the STAN [Arulkumaran et al., 1990] and the Nottingham system [Lui, 1989], to reduce this noise. This method, however, requires accurate alignment of each waveform, and thus the R-peak is used as a reference point.

In the Nottingham system, a linear model of this signal is obtained, from which the PR parameter is extracted. The analysis stages are outlined in Figure 2.10. Each of these stages will now be discussed in detail.

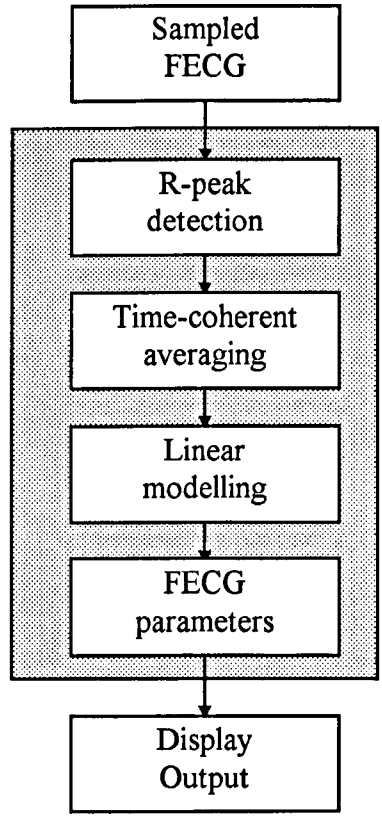


Figure 2.10: Overview of FECG analysis algorithms

2.4.1 R-peak detection

As mentioned above, the R-peak is used to align each waveform, before averaging of the fetal complexes can be performed. The interval between subsequent R-peaks, called the R-R interval, is used to calculate the fetal heart rate on a beat-to-beat basis. Although it has been stated that the waveform will be distorted if the signal is bandpass filtered, one can in fact filter the signal to locate the R-peak, and then align each *unfiltered* waveform using this R-peak position [Hon and Lee, 1963, Peasgood, 1993]. The algorithms used by Peasgood were implemented, and are detailed below [Peasgood, 1993].

The signal can be bandpass filtered around the expected R-peak frequencies, then once this signal exceeds a fixed *threshold*, an R-peak is detected. A FIR filter is used for the bandpass filter. Such filters have symmetrical impulse responses and the shape can be optimised to closely resemble the fetal R wave. This is a form of matched filter, so that the closer the match between the template and fetal R wave, the greater the output. The specification for the filter is shown as the whole shaded area in Figure 2.11. The filter was designed using the Parks-McClellan method within a software package, called Hypersignal [Hyperception]. Minimising the number of taps decreases calculation time, so that 64 coefficients were used. Figure 2.12 shows the actual frequency response for the filter. The impulse response is shown in Figure 2.13: it may be noted how closely the impulse response resembles an R-peak.

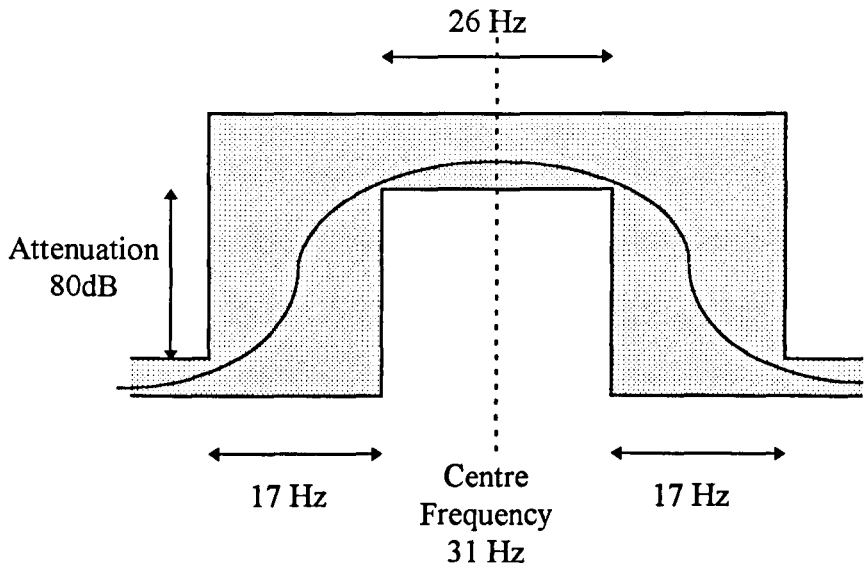


Figure 2.11: Bandpass filter for R-peak detection

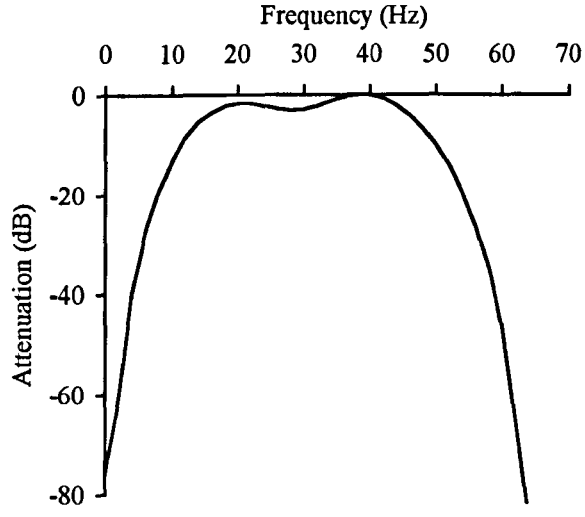


Figure 2.12: Frequency response of FIR filter

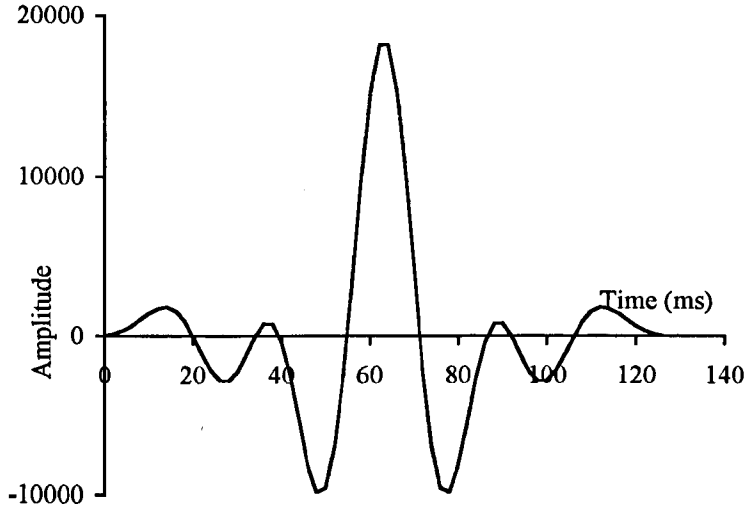


Figure 2.13: Impulse response of FIR filter

An R-peak is deemed to have been detected when the output from the filter exceeds a given threshold value. When the value is exceeded, the following data points are scanned to find when the filtered value drops back below the threshold. The R-peak location, within the filtered data, is determined as the position of the maximum of the filtered data, between these two crossings. The FIR filter has a delay of 32 data points, so that the R-peak location within the unfiltered data array, was determined as the filter delay, subtracted from the occurrence time of the maximum in the filtered data.

Initially the maximum and minimum of the first 2 seconds of filtered data were found. It was necessary to determine the orientation of the data, as electrodes connected the wrong way (or a breech) would cause the ECG signal to be inverted. The orientation was determined by comparing the maximum with the absolute value of the minimum. If the minimum was greater, all subsequent data

was multiplied by -1. This procedure worked on the assumption that the R-wave would have a larger amplitude than any other wave. The threshold was set to 0.55×maximum or 0.55×|minimum|, whichever is the largest. R-peak detection is then carried out on the same 2-second data, and the R-peak maxima stored. A new threshold value is then taken as 0.7×(average maxima), and was stored as the reference threshold. Initially, the threshold was updated on a beat-to-beat basis, taking a new value of 0.7×(peak value of filter).

However, Peasgood found that noise could either make the threshold too low, so that only baseline noise was detected, or too high, in which case all complexes were missed. To prevent corruption of the threshold, and to stabilise it, the threshold was taken as 0.8×(average of the last 3 peak values). Furthermore, if the filter value *exceeded* 1.5×(reference threshold), the threshold value was updated with only 1.5×(reference threshold). This prevented large noise spikes from increasing the reference threshold.

These algorithms were implemented in this system and are shown in Figures 2.14-2.16. However, during periods of poor signal quality, R-peak detection was found to fail and not always recover, once the signal improved. This was due to the threshold having become either too large or too small. With a large threshold setting, no R-peaks were detected. With a low threshold setting, small noise spikes can cause the threshold to be exceeded, and trigger an R-peak detection, resulting in an excessively high heart rate. If the heart rate became too low for 30 seconds, or too high for 4 seconds, the threshold was re-initialised.

```
Sum=0
For last 64 data points
    Multiply data point by FIR filter coefficient and sum
Sum is filtered data point
```

Figure 2.14: Algorithm to filter data

```
For each data point
    Filter data
Find minimum and maximum of filtered data
minimum = - minimum
If maximum > minimum
    threshold = 0.55*maximum
    Set invert to off
else
    threshold = 0.55*minimum
    Set invert to on
For each data point
    Invert data if necessary
    If R peak detection is disabled
        Increase inhibit counter
        If inhibit counter > 40
            Enable R peak detection
    else
        If R peak detected flag is not set
            If data point > threshold (i.e. start of R-peak)
                Set R peak detected flag
                Peak value = data point
        else
            If data point > peak value (i.e. middle of R-peak)
                Peak value = data point
            else
                If data < threshold (i.e. end of R-peak)
                    Disable R peak detection
                    Set inhibit counter to 0
                    Store peak value in peak value array
Calculate average of values in peak value array
Store this average as last 3 peak values in peak value array
Set threshold to 0.7 x average
```

Figure 2.15: R-peak detection set-up algorithm

```
For each data point
    Filter data and invert if necessary
    If R peak detection is disabled
        Increase inhibit counter
        If inhibit counter > 50
            Enable R peak detection
    else
        If R peak detected flag is not set
            If data point > threshold (i.e. start of R-peak)
                Set R peak detected flag
                Peak value = data point
                Store peak position
        else
            If data point > peak value (i.e. middle of R-peak)
                Peak value = data point
                Store peak position
            else
                If data < threshold (i.e. end of R-peak)
                    Disable R peak detection
                    Set inhibit counter to 0
                    If Peak value > 1.5 * threshold
                        Peak value = 1.5 * threshold
                    Store Peak value in peak value array
                    threshold = 0.8* last 3 peak value
                    Store (Peak position-filter delay) in Unfiltered R pos. array
                    heart-rate = 30000/(difference in last 2 R positions)
                    Increment total number of complexes counter
                    Increment good complexes counter
                    calculate valid rate
                    increment R peak counter
                    if R peak counter >= 10
                        end this algorithm
```

Figure 2.16: R-peak detection algorithm

The R-peak detection described by Peasgood has been implemented in the analysis system described in this thesis, although the detection could be improved in a number of ways. Different methods of R-peak detection could be investigated, such as matched filtering, whereby the incoming raw ECG is correlated with a QRS complex template, and when this exceeds a given threshold, the R-peak is detected. The QRS template can be updated with each detected complex.

A matched filtering approach with adaptive normalisation, and error correction, has been previously developed [Gibson, 1995, Gibson et al., 1997]. Firstly, the correlation output can be normalised, using half or full normalisation, to give unity when an exact match is found between the raw ECG and QRS template. Half normalisation involves dividing the correlated output, by the template energy; full normalisation involves dividing the output by the geometric mean of the template energy and the energy of the ECG window. Gibson found the best detection performance by applying half normalisation when the signal was deemed clean, and full normalisation, when the signal was deemed noisy.

Furthermore, a method was employed to correct for false positives and negatives. A falsely detected beat would produce two R-R intervals, where there would normally be one. If the sum of two consecutive intervals lay within acceptable limits from the preceding and following intervals, the two intervals were replaced with a single interval, which had the sum of the two durations. A missed beat would give rise to an interval approximately equal to double the previous and next intervals. In this case, the incorrect interval was replaced with two intervals, with half the duration.

2.4.2 Time-coherent averaging

In 1963, the principle of time-coherent averaging a number of FECGs was used [Hon and Lee, 1963]. This improved the signal to noise ratio [Rompelman and Ros, 1986, Goovaerts and Rompelman, 1991]. Each waveform had the same

weighting, and averages of 10, 20 and 50 waveforms were taken over 3 different sliding windows.

Consider the case when N waveforms are averaged. Once the waveforms have been aligned, the average is calculated for each point at a given position on every waveform. This is repeated for all positions on the waveform. Thus if $x_{i,k}$ is used to represent the i th element on the k th waveform, and $y_{i,j}$ is used to represent the i th element on the j th average waveform, then,

$$y_{i,j} = \frac{1}{N} \sum_{k=j-N+1}^{j} x_{i,k}$$

This is demonstrated in Figure 2.17, where N=2 (i.e. only 2 waveforms are time-coherent averaged).

This improves the SNR by $\sqrt{N}$. However, in order to do this, every data point on N waveforms would have to be stored. In the 1960's, this data overhead was a problem, so a recursive algorithm was used, whereby each new average is calculated from the previous average, and the latest noisy waveform [Rhyne, 1968]. This increases the signal to noise ratio by a factor of $\sqrt{2N-1}$. For each point, this is done with the following formula:

$$y_{i,j} = \frac{N-1}{N} y_{i,j-1} + \frac{1}{N} x_{i,j}$$

It has been shown that this is essentially a low-pass filter [Scott, 1970]. The averaging process is demonstrated in Figure 2.18.

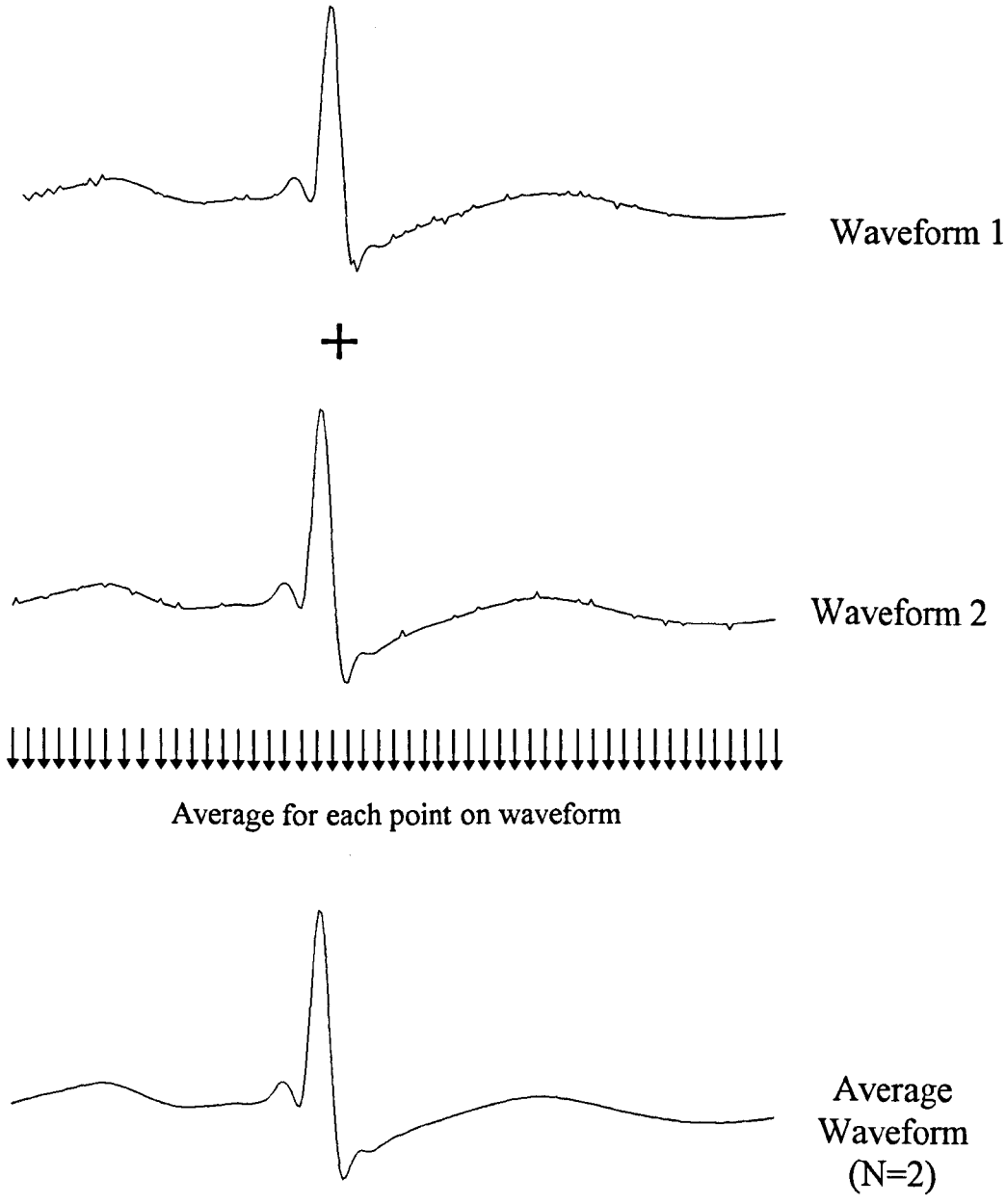


Figure 2.17: Time-coherent averaged waveform, by summing previous waveforms.

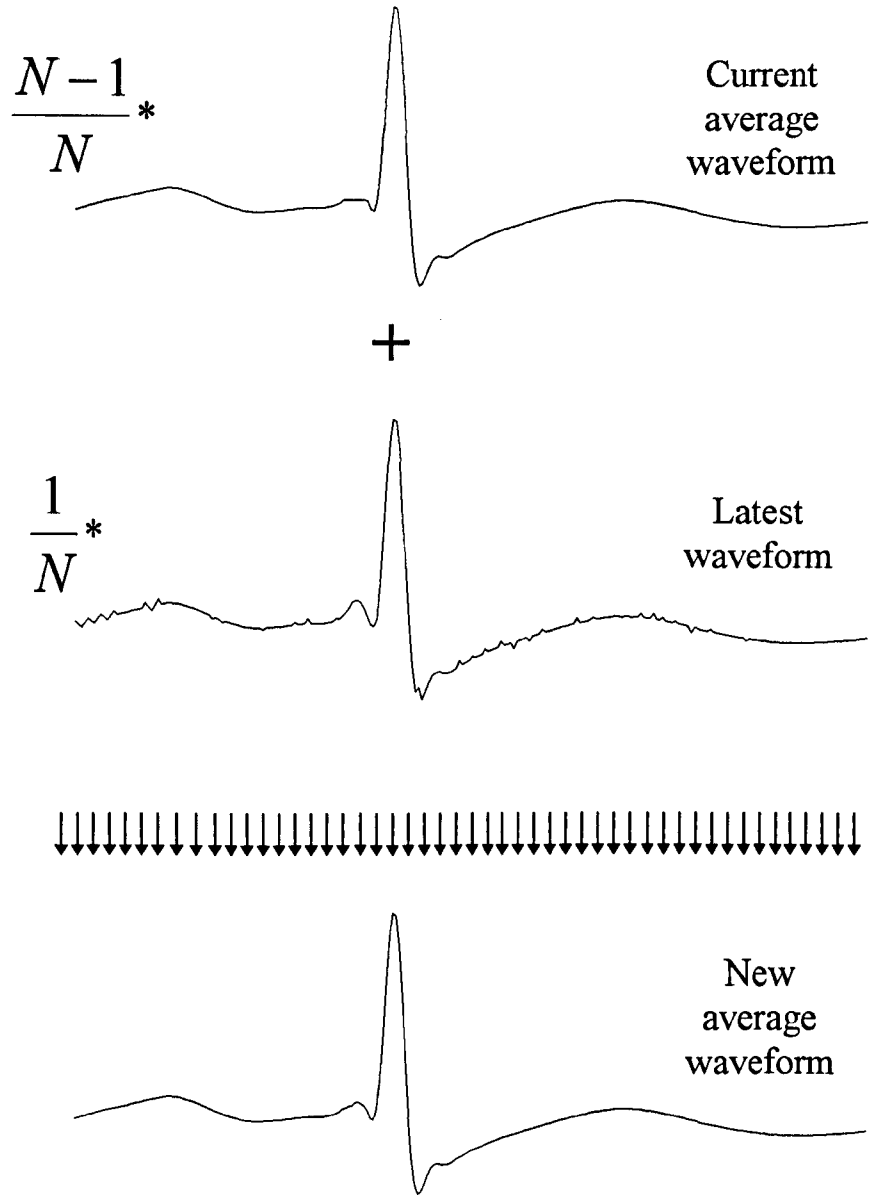


Figure 2.18: Time-coherent averaged waveform, using recursive algorithm

The STAN system uses the moving window, whereas the Rhyne recursive approach had been implemented in the presently used Nottingham system [Sahota, 1997]. This Rhyne approach was also implemented in this thesis. Memory

restrictions are no longer a problem, and once tested, this algorithm could be improved upon.

In order to follow any dynamic changes within the FECG, N needs to be as small as possible. However, reducing N is at the cost of reducing the signal to noise ratio. It was found that a value of N=10 was a good compromise between these two considerations, and this is further demonstrated in Chapter 4. The algorithms to average the waveform are given in Figure 2.19.

Calculate *mean* of latest waveform
if this is first detected complex
    for each data point in new waveform
        $y(i,j) = x(i,j) - mean$
    else
        for each data point in new waveform
            if number of detected complexes, **n** < Number to average over, **N**
                $y(i,j) = \left[x(i,j) - mean + (n-1)y(i,j-1)\right] \div n$
            else
                $y(i,j) = \left[x(i,j) - mean + (N-1)y(i,j-1)\right] \div N$

Figure 2.19: Time coherent averaging algorithm

Noise problems

The detection of the QRS can be triggered by a noise burst, so a procedure was developed to validate the FECG, before it was included in the average [Lui, 1989]. The procedure consisted of 3 tests:

1. Baseline test - movement of the fetal scalp electrode may cause the underlying baseline to wander. The baselines for the P section, QRS section and T sections were calculated. If the difference in baseline between any pair of wave

sections exceeded half the QRS height, the QRS was rejected. This algorithm is given in Figure 2.20.

2. Noise test - large amounts of noise were sometimes picked up by the electrode. To determine the noise strength, the baselines of the P section and T section were subtracted from each point, within the corresponding sections. The differences were then summed over each section, and the average difference calculated for each section. If the average difference was greater than a quarter the QRS height for either section, the waveform was rejected. Figure 2.21 illustrates this algorithm.
3. Saturation test - if the scalp electrode was significantly disturbed, saturation of the A/D could occur. If saturation occurred for more than a third of the duration of the waveform, it was rejected.

These tests prevented noisy waveforms contributing to the average. The first 2 tests were therefore included in the analysis program (*see* Figure 2.22). If saturation occurred, the third test could be added.

```
Find average for each data point in P-wave
Find average, minimum, and maximum for each data point in QRS complex
QRS complex height = maximum - minimum
Find average for each data point in T-wave
If two or more of:
1. the difference in P&QRS average
2. the difference in P&T average
3. the difference in QRS&T average
are greater than half the QRS complex height
    Baseline has drifted significantly
else
    Baseline has not drifted significantly
```

Figure 2.20: Algorithm to test drift of baseline

```
For each point in P wave
    Sum the absolute difference between the data point and the average
Divide this sum by the number of points to give the average P wave difference
For each point in T wave
    Sum the absolute difference between the data point and the average
Divide this sum by the number of points to give the average T wave difference
If either of these average difference were greater than a quarter the QRS
    Waveform is too noisy
else
    Waveform is not too noisy
```

Figure 2.21: Algorithm to test if waveform is too noisy

```
If baseline has not drifted significantly and waveform is not too noisy
    Include waveform in average
else
    Do not include waveform in average
```

Figure 2.22: Algorithm to test drift of baseline

At present, no pre-filtering is carried out on the raw ECG signal to remove baseline wander and mains interference, which should mostly be removed during coherent averaging. If such noise later proved to be a problem, several approaches might be investigated to further filter out these noise types. For example, FIR filters, with a linear phase response have been suggested [vanAlste and Schilder, 1985, vanAlste et al., 1986]. Using a high-pass filter, with a cut-off frequency as high as 0.8 Hz, the baseline wander was removed without distorting the ECG. However, removal of mains at 50 Hz, using a band-stop filter, may remove some of the frequency components in the underlying signal, and so distort the ECG signal. A Chebyshev polynomial could also be used to approximate the baseline [Outram et al., 1995]. The baseline can then simply be subtracted.

It has been shown that no amount of averaging could do away with mains interference [Strackee and Peper, 1992]. In this case, it may be advantageous to include other methods of noise reduction, to reduce the effects of mains noise. A method, whereby the mains signal was assumed to be of constant frequency and amplitude could be used [Chester et al., 1995]. A discrete Fourier transform (DFT) was performed on the ECG data set, to determine the exact mains frequency, its amplitude and phase. This sinusoid could then be removed from the signal, in the time domain.

Improvements to the coherent-averaging technique would be to replace the simple recursive weighting with a more complex filter. Butterworth filters have been previously investigated and have been shown to give good performance [Shield, 1977, Shield and Kirk, 1981]. Time-varying Wiener filters (TVWF) may be applied to the averaged waveform to reduce the noise. It has been shown that similar SNRs are obtained through the use of TVWF, as when approximately three times as many complexes are averaged, with no additional filtering [Woolfson et al., 1995].

2.4.3 Linear modelling

Before the FECG parameters can be calculated, various parts of the waveform must be located. Use of a linear model on the waveform has been detailed previously [Marvell, 1979, Kirk and Smith, 1986]. Various lines of best fit were found (*see* Figure 2.23), by linear regression. Each line in the linear model has a fixed search region, over which several lines of best fit are calculated. The line then chosen depends upon the required gradient for the particular linear model line. If the required gradient is zero, the best fit line with the smallest gradient is chosen; otherwise the line with the largest magnitude of gradient is selected. The search range parameters have been previously documented [Lui, 1989], although this appears to contain errors. Table 2.1 shows the search region, where the R-peak is located at position 100. The algorithm for the software implementation is given in Figure 2.24.

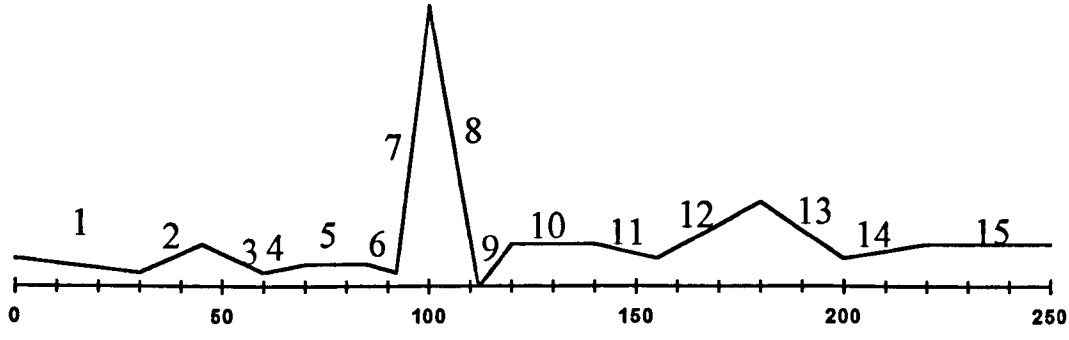


Figure 2.23: FECG linear model

Line No	Search start	Search end	Gradient	Length
1	10	30	-	10
2	25	60	+	10
3	40	70	-	10
4	60	75	+	10
5	75	86	0	10
6	83	94	-	10
7	87	102	+	8
8	98	112	-	8
9	107	120	+	10
10	115	135	0	10
11	125	150	-	10
12	150	190	+	20
13	180	215	-	20
14	210	225	+	10
15	225	249	0	10

Table 2.1: Search region for linear model

```
for each line of best fit
    for each possible time location of line
        calculate line of best fit by linear regression
        if first time occasion
            store line of best fit
        else
            if line should be positive
                if line is more positive than one stored
                    store as best fit so far
            if line should be negative
                if line is more negative than one stored
                    store as best fit so far
            if line should be flat
                if line is flatter than one stored
                    store as best fit so far
Calculate points of intersection of all these lines
```

Figure 2.24: Linear model algorithm

The idea of the linear model may be improved by instead using Chebyshev curves [Outram et al., 1995]. It has been shown that a smooth Chebyshev curve may be fitted to the ST segment of either the raw, or average, fetal waveform, and the T/QRS ratio measured from this.

2.4.4 Parameter extraction

The linear model then allows an easier location of different artefacts on the waveform. The expected positions of the ECG components are listed in Table 2.2 [Kirk and Smith, 1986].

	Intersection	
	of	with
P onset	1	2
P peak	2	3
P term	3	4
Q onset	5	6
Q peak	6	7
R peak	7	8
S peak	8	9
S term	9	10
T onset	11	12
T peak	12	13
T term	13	14

Table 2.2: Position of ECG components

When an inverted T wave occurs, the T wave peak is found as the intersection between lines 11 and 12.

The statistical relationship between any pair of parameters has been previously studied [Family, 1982]. In a linear relationship, the coefficient of correlation, r, gives an indication to the degree of close-fit of the variables X and Y to a straight line. r is defined as:

$$r = \frac{N\sum_{i=1}^{N} X_i Y_i - \left(\sum_{i=1}^{N} X_i\right)\left(\sum_{i=1}^{N} Y_i\right)}{\sqrt{\left\{N\sum_{i=1}^{N} X_i^2 - \left(\sum_{i=1}^{N} X_i\right)^2\right\}\left\{N\sum_{i=1}^{N} Y_i^2 - \left(\sum_{i=1}^{N} Y_i\right)^2\right\}}}$$

This is in fact the covariance of the sample, divided by the standard deviation of the sample X and the standard deviation of the sample Y.

r takes a value between -1 and +1 and falls into three distinct categories:

1. r is significantly negative: all points lie on a straight line with a negative slope. The two variables, X and Y, are inversely related.
2. r is zero or takes a small value. The points are scattered randomly. The two variables are not significantly related or no relationship exists.
3. r is significantly positive. All the points lie on a straight line with positive slope. The two variables, X and Y, are directly related.

Examples of this are shown in Figures 2.25(a)-(e).

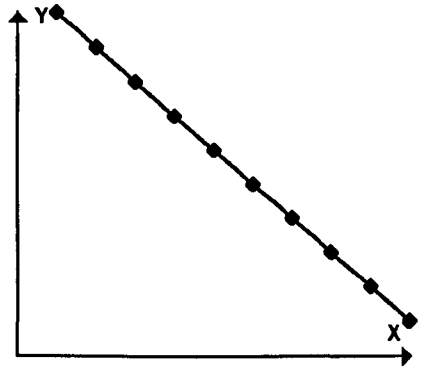


Figure 2.25 (a) r = -1

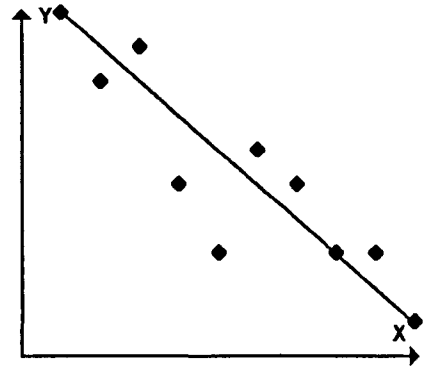


Figure 2.25 (b) r = -low

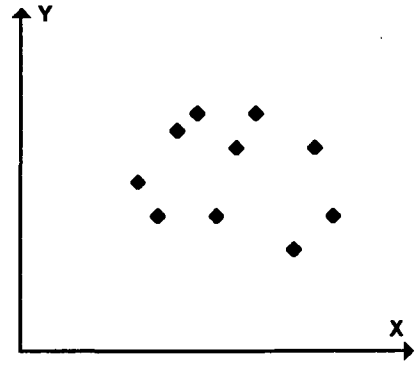


Figure 2.25 (c) r ≈ 0

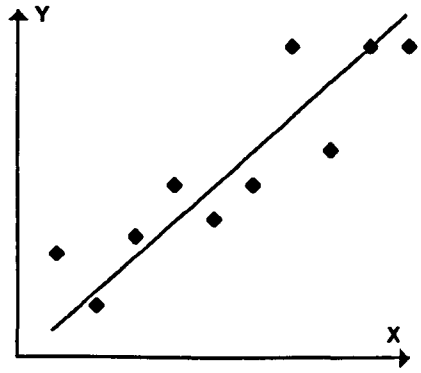


Figure 2.25 (d) r = low

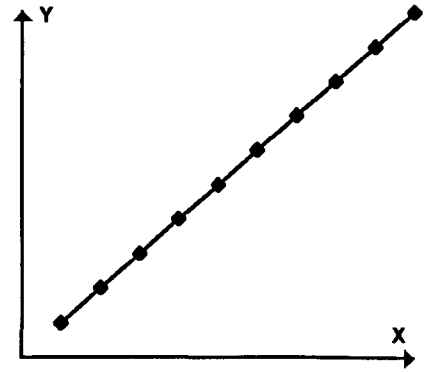


Figure 2.25 (e) r = 1

The FHR and PR have been shown to be related or inversely related depending upon the condition of the fetus [Murray, 1986]. In order to calculate the *Conduction Index*, the coefficient of correlation was originally calculated using the 1-second values of the P-R intervals and FHRs, for the last 300 seconds [Lui, 1989]. However, the Conduction Index system presently being used in the clinical studies, takes only the 2-second values of the P-R intervals and FHRs, over 2½ minutes [Sahota, 1997]. This was implemented in the system, presented in this thesis. The P-R interval is found by subtracting the time locations of the R-peak and the P-peak.

In the STAN system, the T wave elevation is quantified as *the ratio between the T wave height, measured from the level of the P-Q interval, and the QRS peak-to-peak amplitude (T/QRS ratio)* [Rosen and Lindecratz, 1989]. This was implemented in the system, described in this thesis. The code can be easily changed to find the isoelectric line from the T-P interval, instead of the P-Q interval.

2.5 Software Implementation

The program was to split into 2 main sections. The first concerned the configuration of the system, from which the user could choose processing options, display raw ECG, stored within a pre-recorded FECG data file, and enter patient details. The second section performed the processing and analysis of data, from either the Meridian or a data file. A detailed user-guide is presented in Appendix I; details of the software are presented in Appendix II.

Figure 2.26 shows the analysis display during processing. The following are displayed:

1. The raw data, collected from either the Meridian, or read from a pre-recorded data file;
2. The average ECG after R-peak detection and time-coherent averaging of the raw data;
3. The linear model, showing the lines of best fit applied to the average ECG;
4. The fetal heart rate;
5. The Conduction Index;
6. The T/QRS ratio.

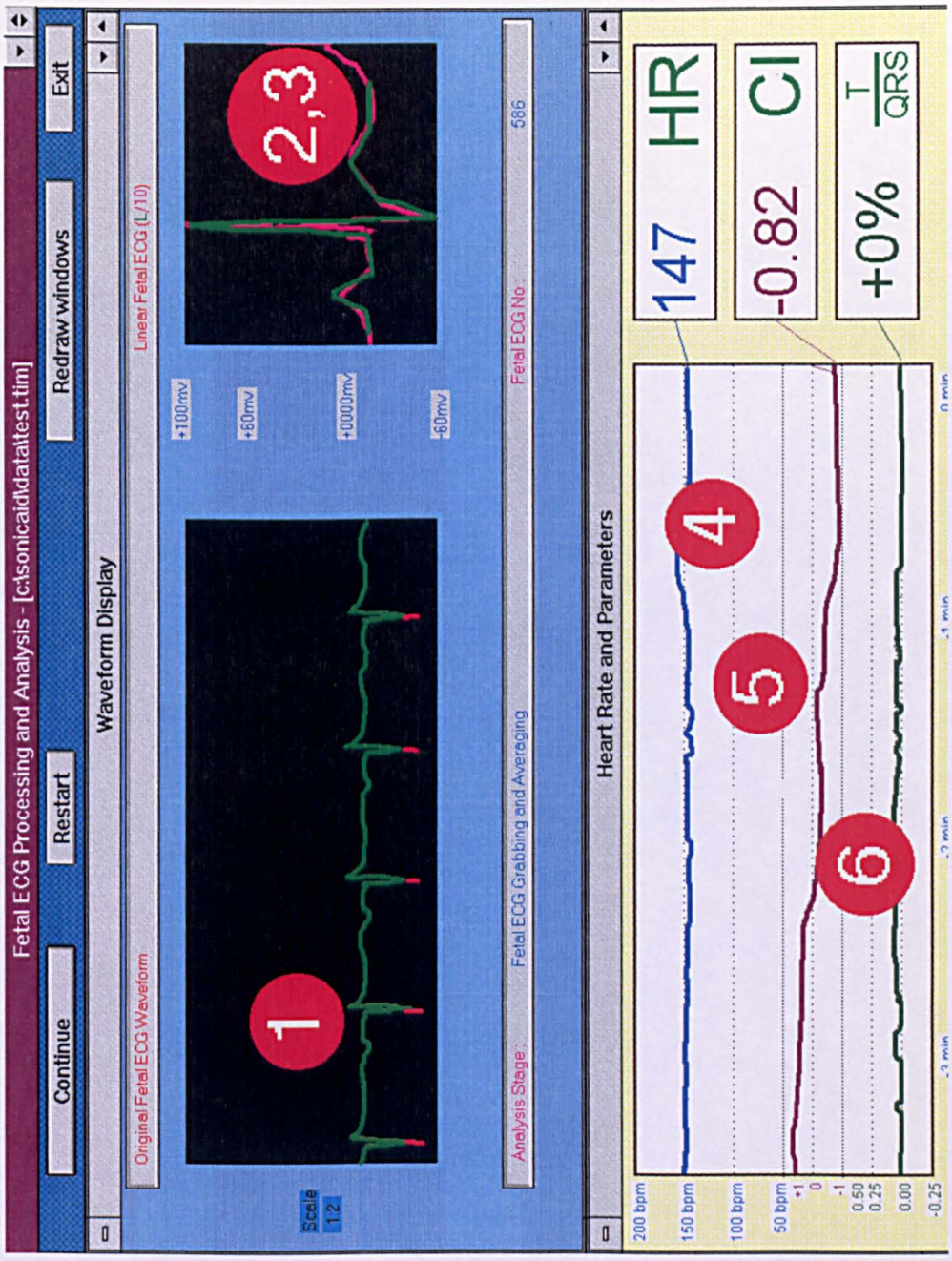


Figure 2.26: The FECG processing window

The photograph of the system is shown in Figure 2.27.

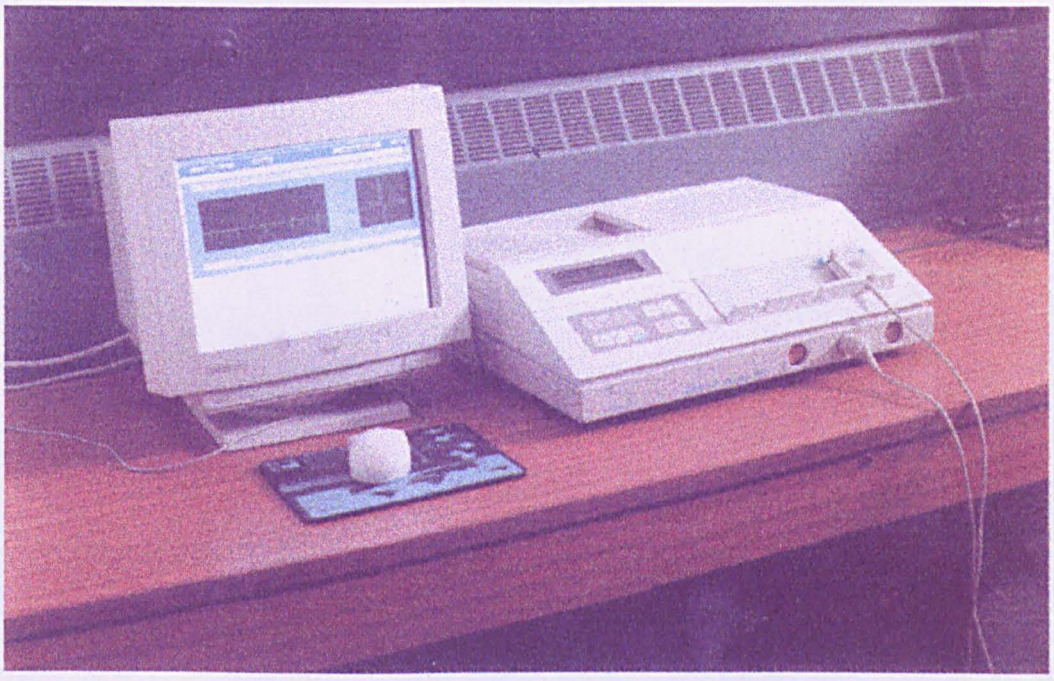


Figure 2.27: Photograph of FECG analysis system.

2.6 Discussion

The algorithms detailed in this chapter allow various FECG parameters to be extracted. Algorithms, which are currently used in other FECG analysis systems, have been implemented. The suggested improvements could be added at a later date, and compared against the present algorithms, to test their effectiveness. The modular method of encoding, together with the high level of software documentation, will allow the software to be adapted quickly, as required, for other analysis methods, and the extraction of other parameters, derived from the ECG. Ideas for improvements to each stage of the processing algorithms, are given.

Initially, it was intended to write the software to allow 3 levels of user (the midwife, the obstetrician, and the researcher), so that the system would be of more commercial use. However, within the scope of this period of research, it was decided to only encode the mid-level (obstetrician) display, and to leave this software development to the later commercial development stage. The system interface could then be designed to the requirements of the user.

Although the algorithms used in this analysis system have been demonstrated by several authors, it was felt that little testing had been done on them to investigate their limitations, and determine their accuracy in controlled conditions. To this end, a FECG simulator has been developed, and this is described in the next Chapter. Then this simulator is used to test the algorithms, with the results from this given in Chapter 4.

3. FECG Simulation system

3.1 Introduction

An analysis system has been developed that processes the ECG signal from a Fetal Scalp Electrode, implementing the algorithms used in the Nottingham *Conduction Index* and Cinventa STAN systems. The system was tested to determine its limitations, in terms of both the software algorithms and the hardware. The plan was for a test signal, with known parameters, to be presented to the system, and the parameters extracted by the analysis software, compared with the known signal parameters. A simulated signal would allow the relevant parameters to be pre-set. This gives an additional benefit over using real data, in that a controlled range of parameters can be used. The development of a simulator for this purpose, is described in this Chapter. The simulator was developed to generate both digital and analogue data. Digital data allowed direct testing of the algorithms, and this is described in Chapter 4. Validation of the full system (including the front-end), using analogue signals, is described in Chapter 6.

The simulator needed to be able to alter the generated FECG signal in the following aspects:

1. Amplitude of different wave components of the ECG (e.g. T height)
2. Duration between fixed points on the ECG (e.g. P-R interval)
3. Heart-Rate
4. Noise, namely respiration noise, mains noise, white noise, and movement artefacts.

Many ECG simulation systems have been developed to test adult ECG equipment. These are capable of generating a single channel ECG [Sandige et al., 1992, Marino, 1993], or several channels for testing multi-lead systems [Evans et al., 1984, Miyahara et al., 1984, Teppner et al., 1988, Schwid, 1988, Franchi et al., 1994]. Commercial systems are available which can produce a fetal ECG, such as the Metron PS-416M Patient Simulator [Metron]. All these systems generated analogue signals, some of which had been converted from a digital signal. However, none of these were suitable for testing this software, as they could neither change the necessary morphology of the fetal ECG, nor simulate all the types of noise required. Therefore, it was decided to write software to produce a simulated FECG. This could also be converted to an analogue signal, using a Digital to Analogue Converter (DAC).

In order to connect the simulator to any analysis equipment, the simulator would have to generate the signal in real-time. The possibility existed to generate and store a signal, with unlimited generation time for each data sample, and then replay the signal in real-time. This would not, however, allow the signal to be adjusted on-line. On-line adjustment limited the complexity of signal generation to the computational power of the PC. It was therefore decided to use an ECG template, whereby the amplitudes of the component wave could be adjusted and the intervals between points on the waveform altered. The heart-rate could be altered by delaying the start of each ECG, and the different types of noise could then be added.

The ECG template was obtained by averaging many intrapartum FECG complexes, so that the noise became negligible. The signal had been recorded using the current Nottingham *Conduction Index* system. It may be argued that this average is not necessarily representative of the morphology of all ECGs. This does not matter, as morphological changes would be made using the simulator software.

The average waveform can be seen in Figure 3.1. Upon the waveform are placed 12 nodes, to provide reference points, between which the time intervals and amplitudes can be changed. Table 3.1 shows the significance of each node.

No.	Node
0	P-Wave onset
1	P-Wave peak
2	P-Wave end
3	Q-Wave onset
4	Q-Wave peak
5	R-Wave peak
6	S-Wave peak
7	S-Wave end
8	T-Wave onset
9	T-Wave peak
10	T-Wave end
11	Wave end

Table 3.1: Reference Nodes

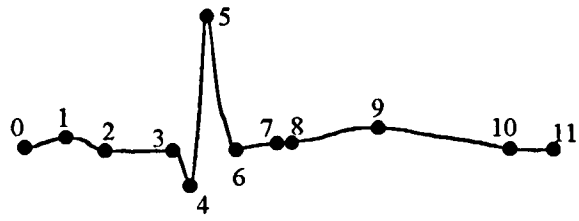


Figure 3.1: Average waveform with reference nodes

3.2 ECG amplitude adjustment

The amplitude of a wave component was changed by setting the amplitude of the relevant node on the waveform. Data points between the previous node and the following node were then changed to fit the node setting. Consider the curve (*see* Figure 3.2) between 3 nodes P_1, P_2 and P_3 on the averaged waveform. The user may wish to stretch the curve such that node P_2 moves to $P_2^{|}$, giving a scale factor to the amplitude adjustment, which will be called α.

$$\alpha = \frac{h_2^{|}}{h_2}$$

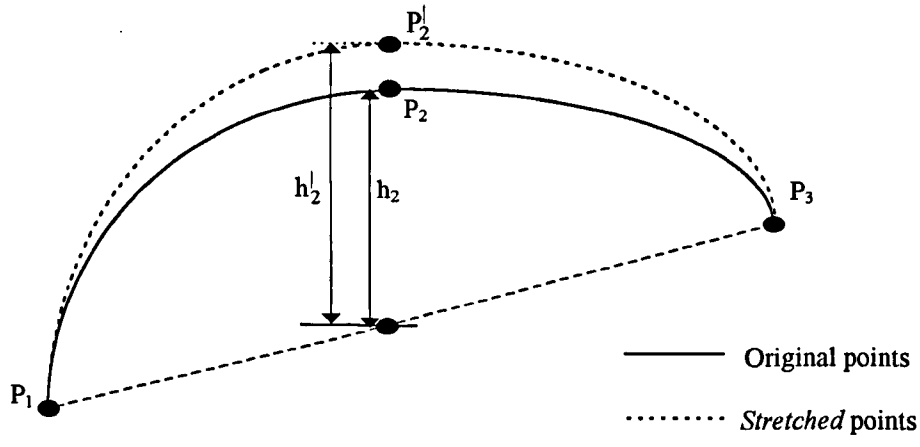


Figure 3.2: Setting node amplitude

All data points on the curve will be *stretched* by the same scale factor, α, from the baseline P_1P_3. The general point on the curve P_a (*see* figure 3.3), will be changed so that

$$h'_a = \alpha h_a$$

The algorithm to do this is derived in Appendix V.

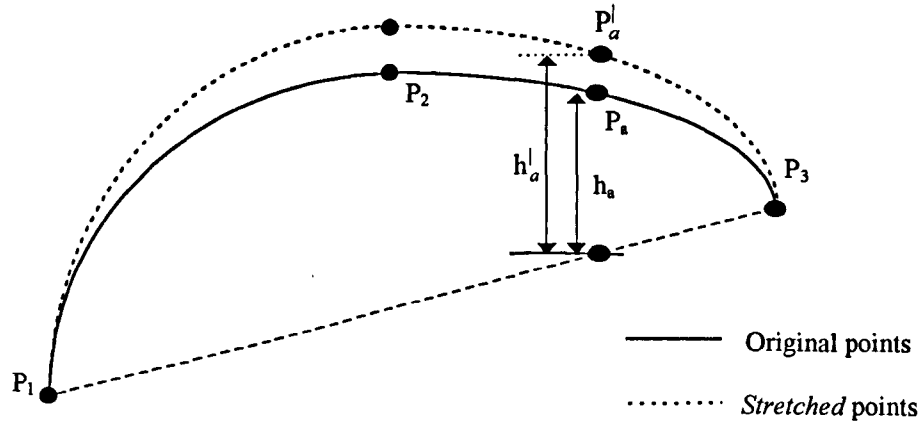


Figure 3.3: Adjusting wave amplitude

If the co-ordinates of P_1, P_2, and P_3 are (x_1,y_1), (x_2,y_2), and (x_3,y_3), the data points between P_1 and P_3, were altered using the algorithm in Figure 3.4 (*see* Appendix V for the derivation).

$$\alpha = \frac{y'_2 - y_c}{y_2 - y_c} \quad where \quad y_c = y_1 + \frac{(x_2 - x_1)(y_3 - y_1)}{(x_3 - x_1)}$$

For each data point (x_a, y_a) between P_1 and P_3

$$y'_a = \alpha(y_a - y_b) + y_b \quad where \quad y_b = y_1 + \frac{(x_a - x_1)(y_3 - y_1)}{(x_3 - x_1)}$$

The data point now has co-ordinates (x_a, y'_a)

Figure 3.4: Algorithm to filter data

Thus to change the T-wave height, the amplitude of the *T-Wave peak* node must be changed. Figure 3.5 shows an ECG, produced by the simulation system. Signal amplitudes between ~20μV to 400μV may be expected [Deans et al., 1996]. This example has amplitude 200μV and 0 V DC level, and the T-wave height has been set to different amplitudes, relative to the QRS amplitude.

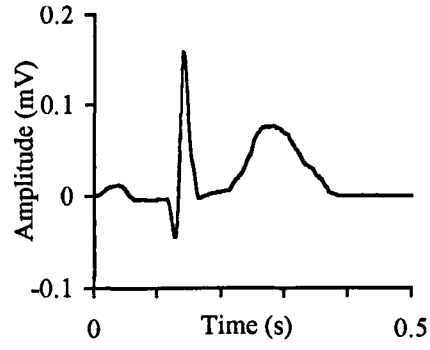


Figure 3.5(a): ECG with T/QRS set to 0.4

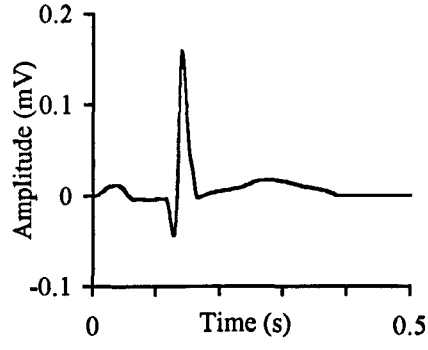


Figure 3.5(b): ECG with T/QRS set to 0.1

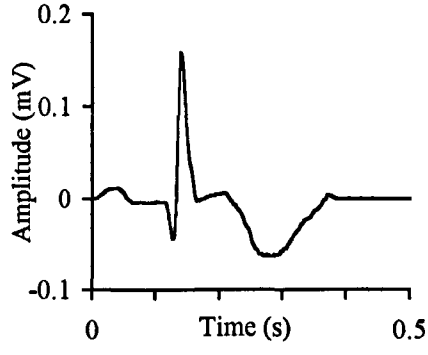


Figure 3.5(c): ECG with T/QRS set to -0.3

3.3 ECG interval adjustment

The ECG time intervals were adjusted by interpolation, which is best explained with an example. Consider 4 data points y_0, y_1, y_2 and y_3 which we wish to time-stretch by one data point (*see* Figure 3.6). If we calculate $y_{0.75}$, $y_{1.5}$ and $y_{2.25}$ (using a method of interpolation), and call them y'_0, y'_1, y'_2, y'_3 and y'_4, we have effectively stretched the curve by one data point.

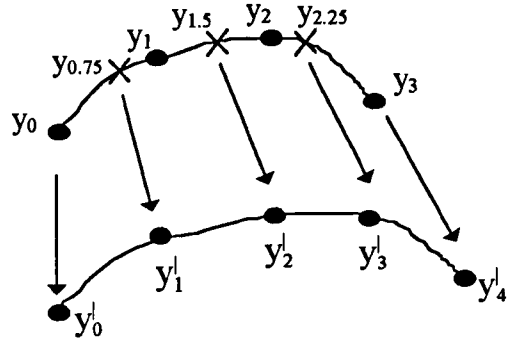


Figure 3.6: ECG interval adjustment

In the general case, stretching from m nodes to n nodes, one uses:

$$y'_i = y_{\left(\frac{m-1}{n-1}\right)i} \qquad i = 0,1,\ldots,n-1$$

which is found by interpolation. The interpolation may be used, when the time-base is increased (n > m) or decreased (n < m).

Initially polynomial interpolation was carried out [Press et al., 1992], with a model order equal to the number of points to be interpolated. However, there may be up to about 100 data points between nodes, and thus the model order could be 200.

This not only proved very slow to calculate, but also caused large errors around the first, and last data points. Thus, it was decided to use a polynomial of order 5. This gave good results, with little processing time. The algorithm for changing the time interval between nodes *N1* and *N2* is shown in Figure 3.7.

```
If interval is being reduced
        diff = decrease in interval
        for each data point, data(i), between N2 and end
                data(i) = data(i + diff)
else
        diff = increase in interval
        for each data point, data(i), between end and N2
                data(i + diff) = data(i)
for each node from N1 to (N2 - 1)
        for each data point, data(i), between node and (node + 1)
                interpolate to find new data(i)
```

Figure 3.7: Algorithm to filter data

Figure 3.8 shows the PR interval being changed, by setting the interval between the *P-Wave end* node and the *Q-Wave onset* node. The PR interval of 200 ms is extraordinarily long, but is shown for demonstration purposes.

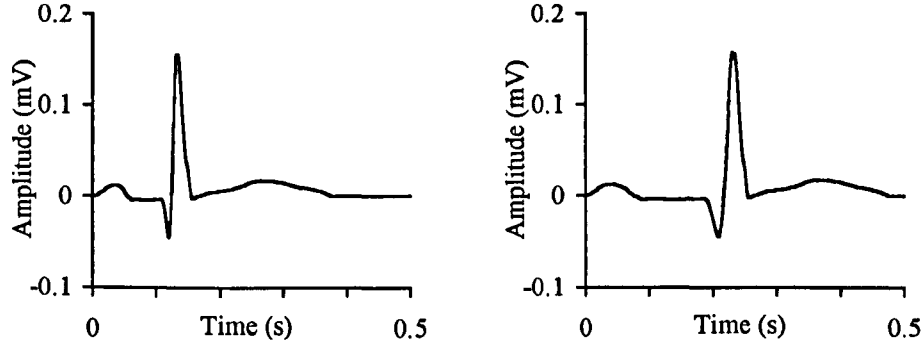


Figure 3.8: ECG with PR interval changed from 100 ms to 200 ms

3.4 HR adjustment

The heart rate could be adjusted by simply increasing the number of data points between each waveform (*see* Figure 3.9).

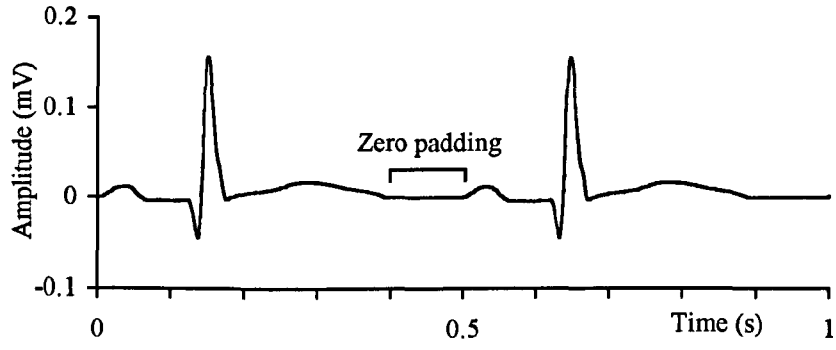


Figure 3.9: Adjustment of heart-rate

3.5 Addition of noise

Three types of noise were simulated:

1. Single frequency noise, namely respiration noise and mains noise, which could both be added to the signal concurrently.
2. White noise
3. Artefacts, where DC shifts occur.

3.5.1 Single frequency noise

The mains noise frequency was assumed to be at 50 Hz, and could be altered to a different mains frequency, such as the U.S. mains frequency of 60 Hz, through simple software adjustment. The frequency of baseline wander was assumed to be at the respiration frequency of 0.3 Hz, and could also be altered to any frequency in the software code.

Three controls were added to the software, so that the user could set the signal/noise ratio (SNR), for respiratory, mains and white noise.

The noise power was simply:

$$Noise_power = \frac{Signal_power}{SNR}$$

In the case of single frequency noise, the noise power is related to the amplitude, A, of the sinusoidal signal as follows,

$$Noise_power = \frac{A^2}{2}$$

Thus a sinusoid for each noise type was added to the signal with amplitude

$$\sqrt{\frac{2.Signal_power}{SNR}}$$

Figure 3.10 shows the ECG with different levels of mains noise added. In this case, the QRS height has been set to 200 μV and the DC level to 0 V.

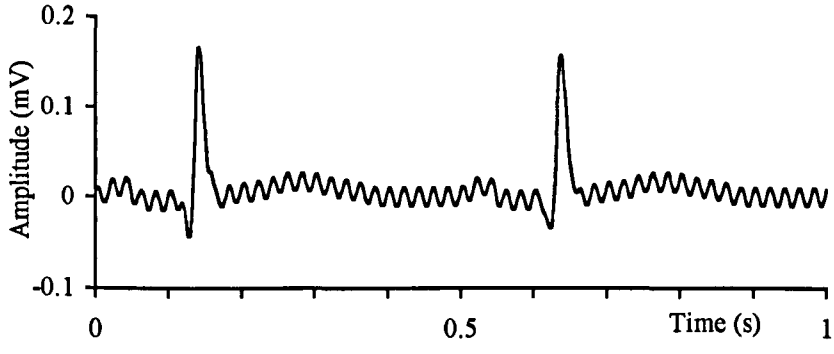


Figure 3.10(a): ECG with mains noise (SNR at 10 dB)

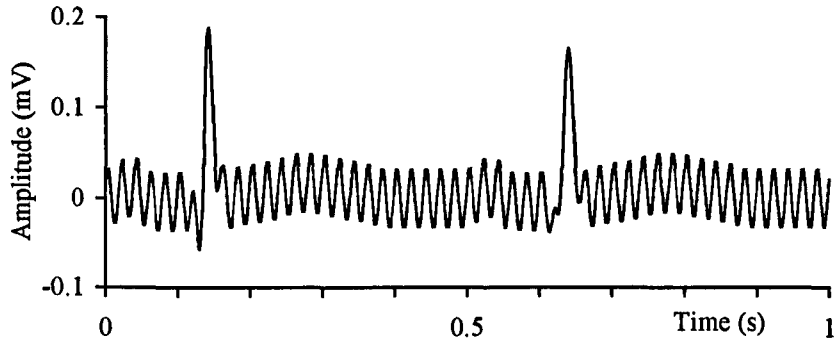


Figure 3.10(b): ECG with mains noise (SNR at 0 dB)

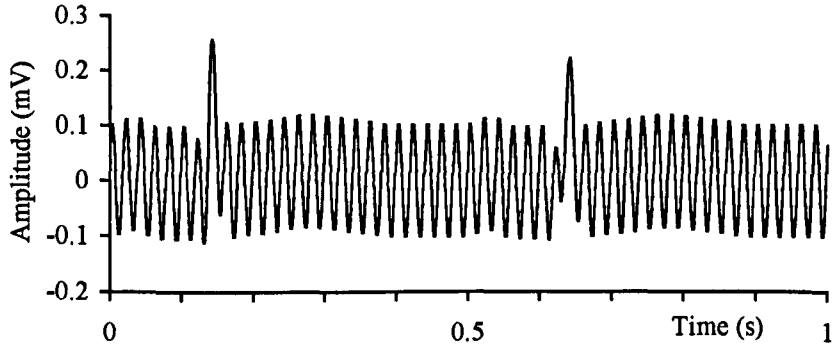


Figure 3.10(c): ECG with mains noise (SNR at -10 dB)

Figure 3.11 shows the ECG with different levels of respiration noise added. Again, the QRS height has been set to 200 μV and the DC level to 0 V.

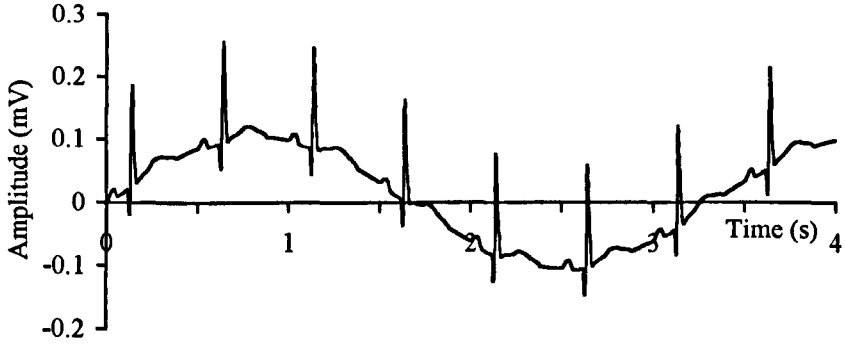


Figure 3.11(a): ECG with respiration noise (SNR at -10 dB)

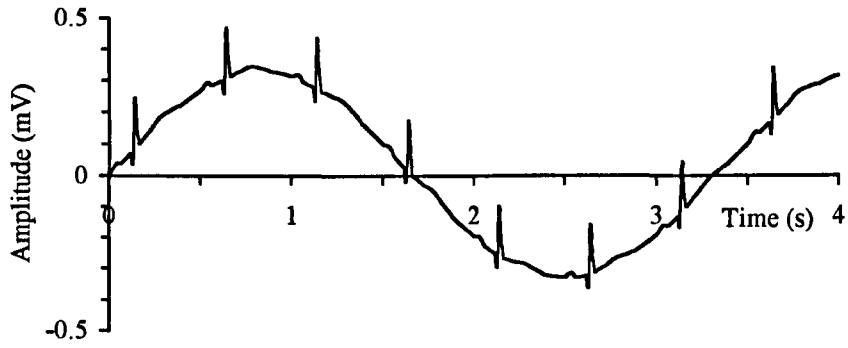


Figure 3.11(b): ECG with respiration noise (SNR at -20 dB)

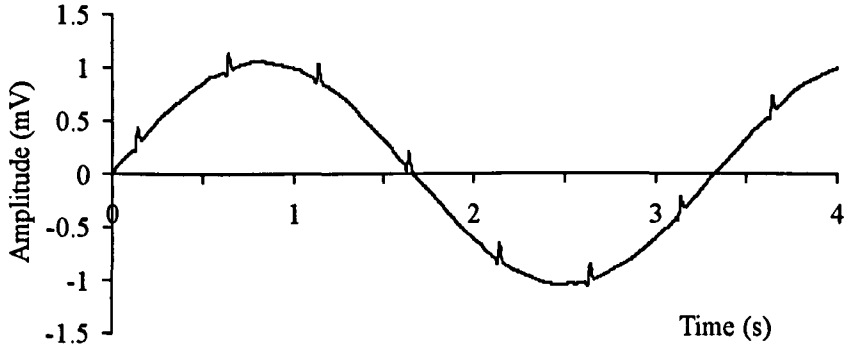


Figure 3.11(c): ECG with respiration noise (SNR at -30 dB)

3.5.2 White noise

For white noise the probability of a particular amplitude occurring, at any instant on time, is given by the Gaussian probability function [Betts, 1976]. This was generated, using previously developed code [Press et al., 1992]. This generated a normally distributed deviate, with zero mean and unit variance. Multiplying, each generated value by A, and adding this to the signal, gave a noise power:

$$Noise_power = A^2$$

Thus, the amplitude A, needed to be set to:

$$\sqrt{\frac{Signal_power}{SNR}}$$

Figure 3.12 shows the ECG with different levels of white noise added.

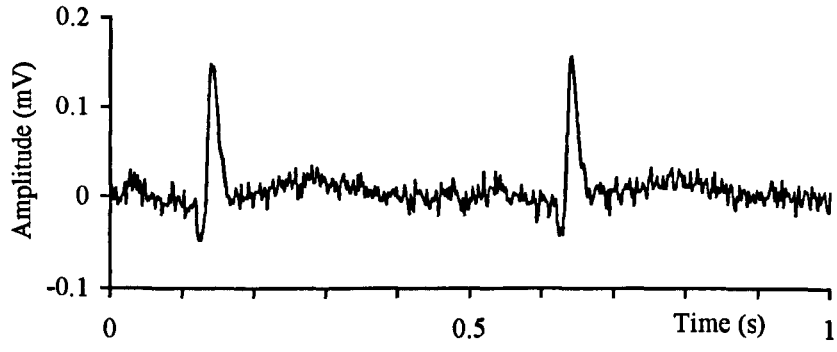


Figure 3.12(a): ECG with white noise (SNR at 10 dB)

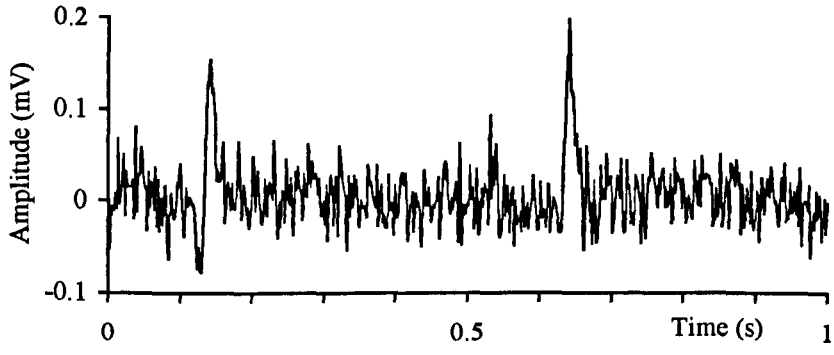


Figure 3.12(b): ECG with white noise (SNR at 0 dB)

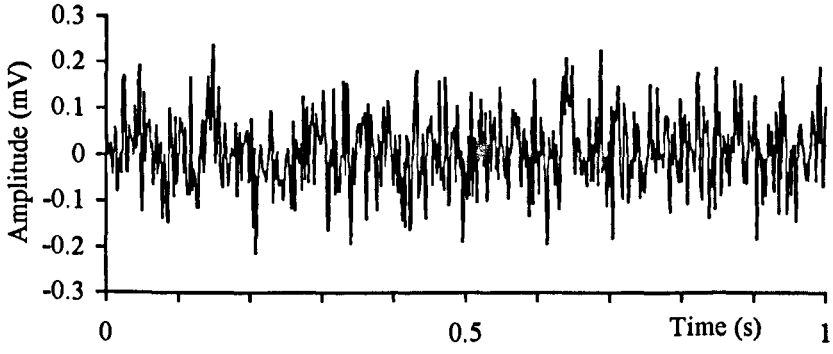


Figure 3.12(c): ECG with white noise (SNR at -10 dB)

The SNR for the generated noise was validated in the following manner: Two signals were generated, one with noise added, and one without. A separate program was written to subtract these two signals, giving a signal containing just the noise. The power of the noise signal was then compared to the power of the noiseless signal to give the SNR. This was repeated for various signals, to check for agreement between the SNR set in the simulation program, and the SNR obtained using this validation method.

3.5.3 Movement artefacts

Adjustment to the artefacts could be made via 3 controls: setting the likelihood of a shift occurring; the maximum, and the minimum DC baseline levels. Figure 3.13 shows an ECG with a baseline of between -0.5 mV and +0.5 mV, changing, on average, once per second, and the 8 DC shifts are highlighted. (The likelihood of a shift occurring has been set so high for demonstration purposes.) The algorithm is shown in Figure 3.14.

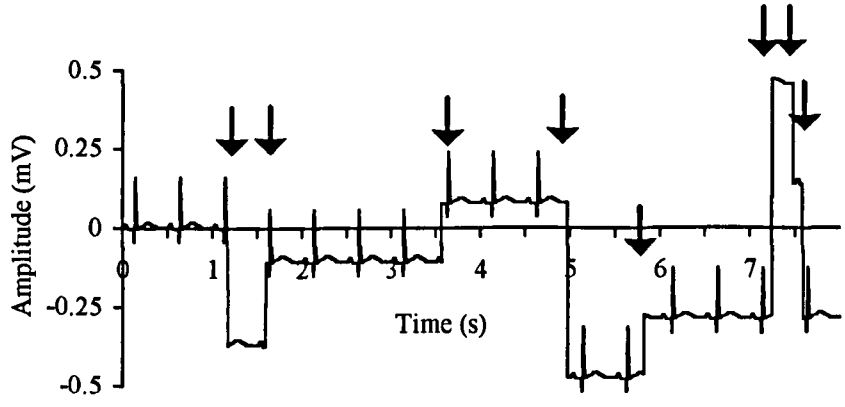


Figure 3.13: ECG with baseline shifts

```
Probability of DC shift per sample (p) = Probability per second × sample time
If p > 0
        Generate random integer between 0 and (p-1)
        If integer = 0
                Generate random integer between min. and max. DC shift
                Add this to the signal
```

Figure 3.14: Algorithm to filter data

This was validated by checking that the baseline values lay within the maximum and minimum, and by counting the number of DC shifts within a set time period. One file was generated with no probability of DC baseline shifts, and another file was generated with the probability of DC shifts set to 100% per second. All other parameters remained constant, and both files were generated for a 24 hr time period. These files were then compared, and the number of shifts in baseline were counted in the 24 hr period. The number of shifts counted, agreed (to 95 % confidence) with the number of shifts expected.

3.6 Code

It was envisaged that the simulation program would be used by others. Once again, this made the choice of a Microsoft Windows environment a good one. All the code was written using Borland ObjectWindows for the same reasons as its choice for the development of the analysis system (*see* section 2.2). The main display of the program can be seen in Figure 3.15. A user guide and a detailed breakdown of the object structure are presented in Appendices III and IV.

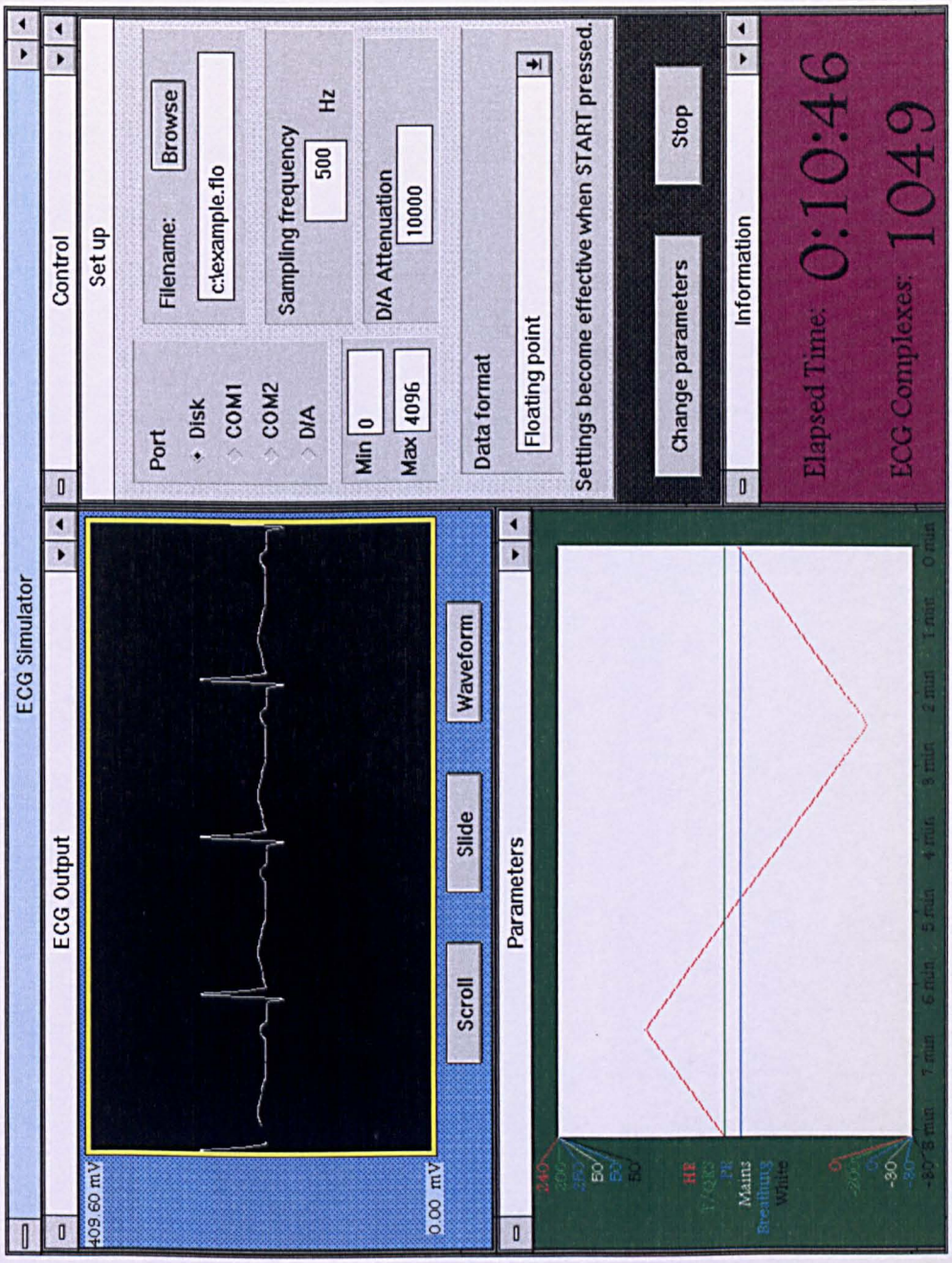


Figure 3.15: Simulation system display

3.7 Hardware

The simulated data could either be output as a data file[1], or converted to an analogue signal. An ADC-42 card [Blue Chip Technology], was chosen for this task. This was capable of output rates of 1 MHz, more than sufficient for this application, which generated 500 data points per second. The card has a 12-bit resolution, and came with the necessary library software to output data from the Microsoft Windows environment to the card.

The output range from the card was 0 to 5V, so that the signal had to be attenuated, using 2 resistors, to give the correct amplitude for the ECG. The resolution of the ECG complex dictated the range of the output, which was required to simulate a change in the baseline of the signal. For instance, if 8 bits were used for the resolution of the signal, and the R-peak amplitude was 0.25 mV, the output range would be 4 mV. The signal for the card would therefore need to be attenuated by 1250, using 2 resistors in the ratio 1249:1. The ranges for other ECG resolutions, with an R-peak amplitude of 0.25 mV, are given in Table 3.2.

ECG resolution (bits)	Output range (mV)	Attenuation
4	64	78.1
5	32	156.3
6	16	312.5
7	8	625
8	4	1250

Table 3.2: Output range of DAC card and attenuation required

[1] Data could be stored in 2 formats: as a 2 byte integer, Least Significant Byte first (generating data at a rate of 3.6 MB per hour), or as a float (7.2 MB per hour).

3.8 Discussion

A FECG simulation system has been developed which is capable of generating both digital and analogue signals. Changes in the morphology, and heart rate can be made, and different noise types can be added. The interface within the Microsoft Windows environment, allows easy user interaction.

However, a number of improvements could be made to the simulation system. Firstly, the adjustment of the ECG could be made more user friendly. At present, a limited number of parameters can be adjusted by setting control bars within the Windows package. By clicking the mouse on any node on the displayed ECG, and altering the morphology by dragging the mouse, the system would not only be easier to use, but would allow a larger number of parameters to be altered, without having to change the code. A history of all the parameters is displayed for the previous 8 minutes. A scroll-back facility for chosen parameters would prove helpful.

The signal generated need not be just an ECG. Test signals such as sinusoids, impulses and triangular waves would help validate equipment. Furthermore a second ECG signal could be added to the FECG, to simulate the signal received from the maternal abdomen, during labour.

The output range of the DAC was limited by the resolution of the ECG, and the total number of output bits of the card. If the resolution was not to be compromised, the range could be further increased by using a 16-bit (or higher)

DAC card. A 16-bit DAC card would give an output range 16 times as great as with a 12-bit card.

The versatility of generating the FECG signal in software, gave the additional benefit that the system could easily be replicated – the simulation software simply needed to be installed on an IBM-compatible PC, with a DAC card.

Now that the simulator had been developed, validation of the analysis system could be carried out. Validation of the software is described in Chapter 4, whilst validation of the full system is given in Chapter 6.

4. Validation of the FECG analysis algorithms

4.1 Introduction

In this Chapter, the limitations of the software algorithms, used in the FECG analysis system, are determined. The accuracy is found by applying a set of test signals from the simulator, and comparing the parameters generated by the analysis system, with those set within the simulator.

Testing was carried out in 4 main areas:

1. **R-peak detection** - the accuracy of the heart-rate was determined, together with the minimum and maximum detectable heart-rates. The ability to detect the R-peak, under different types of noise, was also investigated.
2. **Time-coherent averaging** - the improvement in signal-to-noise ratio (SNR) under different levels of noise is demonstrated, when the signal is time-coherent averaged. This is shown when different numbers of waveforms are averaged. An improvement in the SNR would be expected, when more complexes are used in the coherent averaging.
3. **Linear modelling and parameter extraction** - the ability of the linear model to determine parameters accurately is very important, especially when many measured parameters are used in calculating a derived parameter (such as the Conduction Index). Any errors are then propagated through to the derived parameter. Only the PR interval and T wave amplitude are currently used, so the accuracy of the linear model is therefore shown, by examining the accuracy of these two parameters. Although this gives only a limited validation of the

linear model, it will give an indication to the accuracy of measuring other time intervals and wave amplitudes.

4. **Parameter accuracy under noisy conditions** - finally, the effects on both the measured parameters (PR interval and T/QRS ratio) and any derived parameters (Conduction Index) are shown under different noise levels. An improved SNR, through a larger number of complexes in the coherent averaging, is shown to be a trade-off with the ability to track changes in the waveform.

Improved algorithms could be easily implemented in the analysis system, and then compared with the current algorithms, using the simulator.

Throughout the validation, it was necessary to know the usual values for various parameters of the fetal ECG. Both the mean value and the standard deviation of many of the ECG parameters have been previously found, in a study into the ECG of 114 fetuses [Mohajer, 1994]. The parameters used for this validation were based on these figures.

4.2 R-peak detection

The R-peak detection was tested in 3 ways: the accuracy was tested, by analysing static heart-rates (HRs); the HR limits were found by simply increasing and decreasing the HR; and the ability to detect peaks in noise was detected by increasing noise levels.

An ECG signal, with a constant heart rate and no noise was generated by the simulation system, and stored in a binary file. The data was then passed through the analysis software, which automatically stores various parameters measured from the ECG. The HR parameters were inspected, and the average HR and RMS error were calculated, for N complexes.

$$Error_{RMS} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(FHR_{Generated} - FHR_{measured}\right)^2}$$

This process was repeated for different HRs and the results are shown in Table 4.1. The RMS errors were small and as expected.

Generated	**Detected**	
Heart-Rate (b.p.m.)	**Average HR (b.p.m.)**	**RMS HR Error (b.p.m.)**
60	60.0	0
80	80.0	0
100	100.0	0
120	120.0	0
140	140.0	0.295
160	160.0	0.427
180	180.0	0.510

Table 4.1: Comparison of generated and detected HRs.

Since R-peak detection locates the R-peak to the nearest sample point, then with a 500 Hz sampling frequency, this is every 2 ms. With a HR of 160 b.p.m., the time interval between peaks is 375 ms. Consequently, the detection algorithm will give time intervals of 374 ms, half the time, and 376 ms, otherwise, and the error^2 will

be $\left(160-\frac{60}{0.374}\right)^2$ half the time, and $\left(160-\frac{60}{0.376}\right)^2$, the other half. Therefore the RMS error for the test set will be,

$$RMS_Error=\frac{1}{2}\sqrt{\left(160-\frac{60}{0.374}\right)^2+\left(160-\frac{60}{0.376}\right)^2}=0.427 \quad b.p.m.$$

This is indeed the measured value.

Such errors are not due to any error in the software, but due to the fact that, at a sampling rate of 500 Hz, the R-peak location is aligned to the nearest 2 ms. Hence they are inevitable, but would be reduced if either a higher sampling rate is used, or the R-peak position is determined between sample points by interpolation. The largest error in HR would be when the RR interval was odd (in milliseconds), causing each detected RR interval to have an error of 1 ms. The error is further increased at higher HRs. At an RR interval of 251 ms (an extreme 239 b.p.m.!), the analyser would give alternate RR intervals of 250 ms and 252 ms. The HR would therefore alternate between 238 b.p.m. and 240 b.p.m., giving a maximum RMS error of 1.34 b.p.m. Such errors in the system are not large enough to affect clinical decisions, and so are acceptable.

The next stage of R-peak validation was to determine the maximum and minimum value of HR that could be detected accurately. Two signals were generated with the simulator: one with an increasing heart-rate, the other, with a decreasing heart-rate. When this data was passed to the analysis software, it was found that as the HR decreased, and fell below 15 b.p.m., the software was unable to calculate the HR correctly. This was not actually a fault with the R-peak detection algorithm in the software, but due to the fact that two 2-second buffers are used to store the

raw FECG. This allows R-R intervals to be measured up to 4s, and hence HRs as low as 15 b.p.m. Since such low HRs should not be encountered, this is not a problem. The system was able to detect HRs of up to 300 b.p.m. Above this value, more than 10 heart beats occurred in a 2 second window, for which the system was designed, and the calculated HR becomes incorrect.

A mean FHR of 137 b.p.m. with a standard deviation of 15.3 b.p.m. is expected [Mohajer, 1994]. The R-peak detection in the analysis software has been validated to a given accuracy from 15 to 300 b.p.m., covering any possible fetal heart rate.

Finally, the detection algorithm was tested to determine the maximum noise levels of mains noise, white noise and low frequency noise, under which the R-peak could still be detected. Three ECG signals, each containing increasing noise (and, thus decreasing SNR) of each noise type, were generated by the simulator and processed by the analysis software. In order to measure the effectiveness of the detection, a measure of performance must be made. A performance of the detection, that has been used by several authors [Azevedo and Longini., 1980, Park et al., 1992], is defined as:

$$Performance(\%) = \frac{\#fetal_R\text{-}waves - (\#misses + \#false)}{\#fetal_R\text{-}waves} \times 100$$
$$= \%true_detections - \%false_alarms$$

Location of the R-peaks was determined by eye in these studies, as the test data were real, but this was not necessary in this work, as the simulated ECG was generated with a constant heart-rate, and thus the R-peaks occurred at predetermined time locations. A detection was said to be true, if the detection

occurred within 20 ms of the true simulated location. The figure of 20 ms would allow a correct detection, when the true R-wave position has been corrupted by noise. At a heart rate of 120 b.p.m., this equates to a heart-rate error of 5 b.p.m.

The performance under conditions of either white noise or mains noise is shown in Figure 4.1. To add noise to the signal, the power of the continuous signal is measured, and the power of noise calculated for a given SNR (*see* section 3.5).

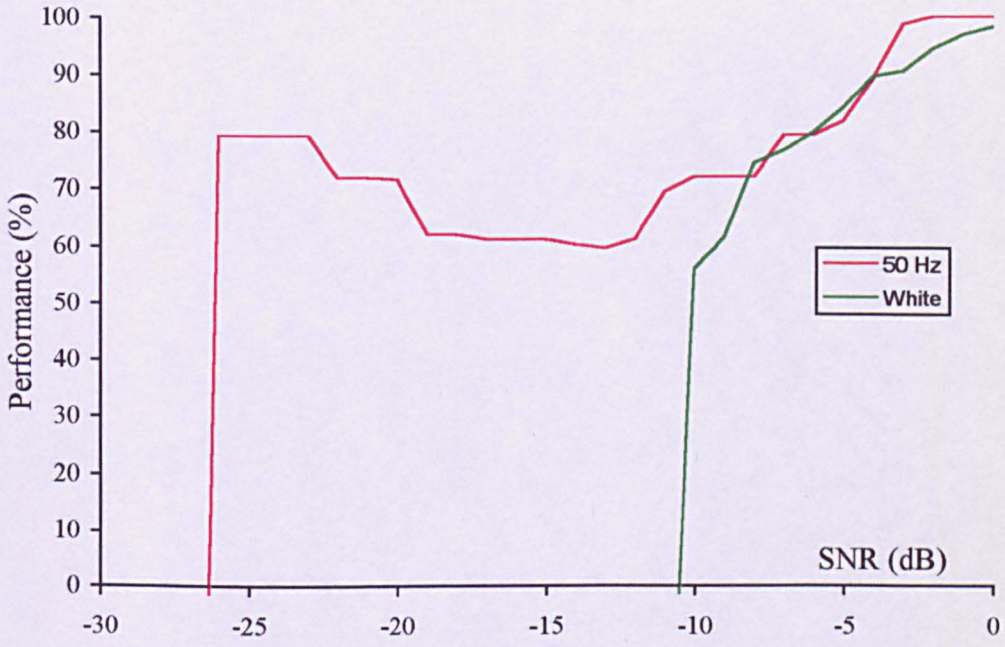


Figure 4.1: R-peak detection performance under different noise conditions

For an R-peak to be detected, the filtered R-peak had to exceed a given threshold (*see* section 2.4.1). The addition of the noise caused the filtered R-peak maxima to fluctuate around the correct (noiseless) value. With both types of noise, as the SNR decreased from approximately 0 dB, some of the maxima did not exceed the threshold for the R-peak detection, and the R-peak was missed. The R-peak

threshold was calculated from an average of the last 3 peak values (*see* section 2.4.1). Once these lower R-peaks were missed, the average value, and thus the threshold, increased, causing more R-peaks to be missed.

In the case of added mains noise below -15 dB, as more noise was added the performance increased slightly. In this case, for some of the previously missed R-peaks, the addition of noise caused the threshold to be exceeded, and the performance improved.

With respiration noise added up to a SNR of -70 dB, the performance of detection remained 100%.

The points at which the algorithms failed to detect R-peaks with 100% performance and 50% performance are shown in Table 4.2.

Noise type	**SNR_{min} for 50%**	**SNR_{min} for 100%**
Mains interference	-26 dB	-2 dB
White noise	-10 dB	4 dB
Respiration noise	< -70 dB	< -70 dB

Table 4.2: Performance of R-peak detection under noise

The noise rejection for mains interference is considerable higher than that for the white noise. This is not surprising as the R-peak detection is carried out by first filtering the signal with a pass-band around the R-peak constituent frequencies

(*see* section 2.4.1). The band-pass filter has 3 dB points at 15 Hz and 45 Hz. This attenuates the 50 Hz component by 10 dB, but much of the white noise passes through the pass-band of the filter. The effect of respiration noise was even less, as the 0.3 Hz frequency component used was filtered out. A SNR of -70 dB was not significant enough to affect the R-peak detection.

4.3 Time-coherent averaging

The time-coherent averaging algorithms needed to be validated to show that an improvement in SNR occurred, and to quantify by how much it improved.

The simulator was used to generate signals with different noise levels added, for each noise type. These signals were then passed through the analysis program and the averaged waveforms were stored. A waveform, with no noise, was subtracted from each average to leave just noise. The power of this noise, and the power of the average were calculated, and used to find the SNR. In this way, an improvement in the SNR was demonstrated when using time-coherent averaging. Furthermore, the time-coherent averaging was carried out using a different number of complexes to obtain the averaged waveform.

Figure 4.2 shows the SNR of the average waveform, when different levels of white noise are added to the signal. Each detected FECG is validated before it is included in the averaged waveform (*see* section 2.4.2), to reduce the noise in the averaged waveform. Below a SNR of -10 dB, the waveforms are considered too noisy to contribute towards an average; so that no average is obtained at these

noise levels. The time-coherent averaged waveform has been described previously (*see* section 2.4.2). It is a weighted, running average calculated as:

$$Average_n = \frac{1}{N} Complex_n + \frac{N-1}{N} Average_{n-1}$$

When averaging N complexes, this process improves the SNR by $\sqrt{2N-1}$ [Rhyne, 1968]. This is generally the case: an improvement of approximately 10 dB can be seen when N = 5, and up to 20 dB for N = 50. However, within the range 12 dB to 20 dB, the reduction in noise is small, even when N is increased significantly above 10. Up to 20 dB, the noise is sufficient to cause the location of the R-peak, to be inaccurate, and the misalignment of noisy complexes, then causes the averaging technique to not be as efficient.

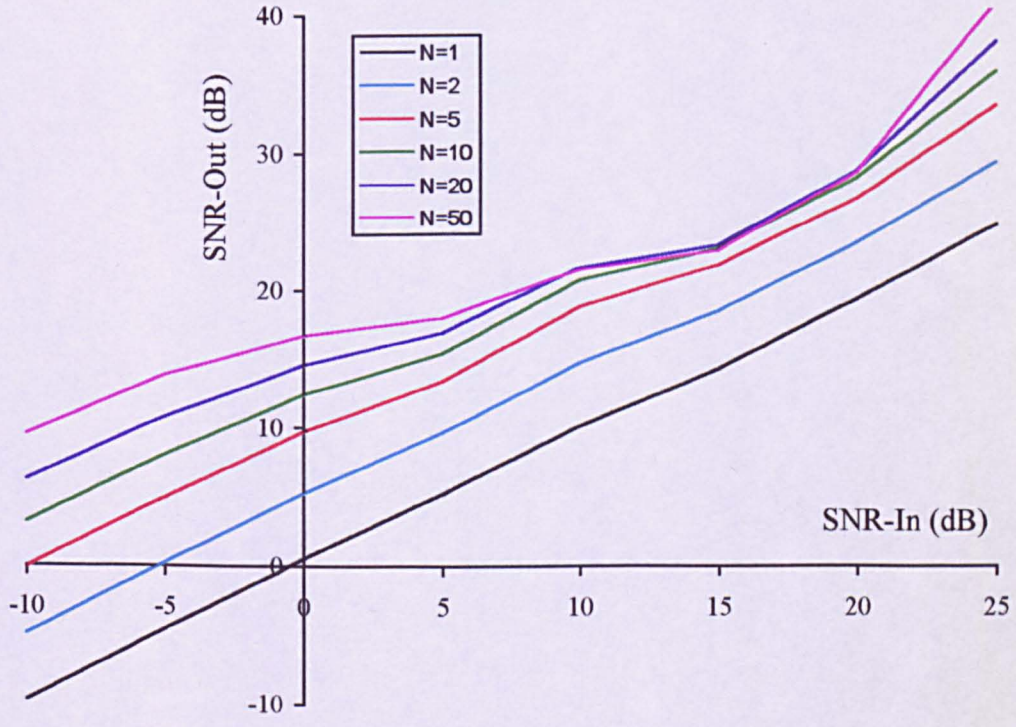


Figure 4.2: SNR improvement during coherent-averaging for white noise

The improvement is demonstrated in Figure 4.3, where an ECG signal containing white noise of SNR 0 dB is averaged.

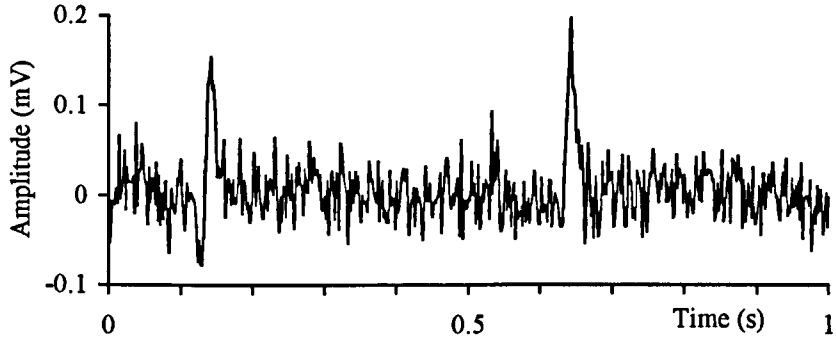


Figure 4.3(a): ECG signal with 0 dB white noise

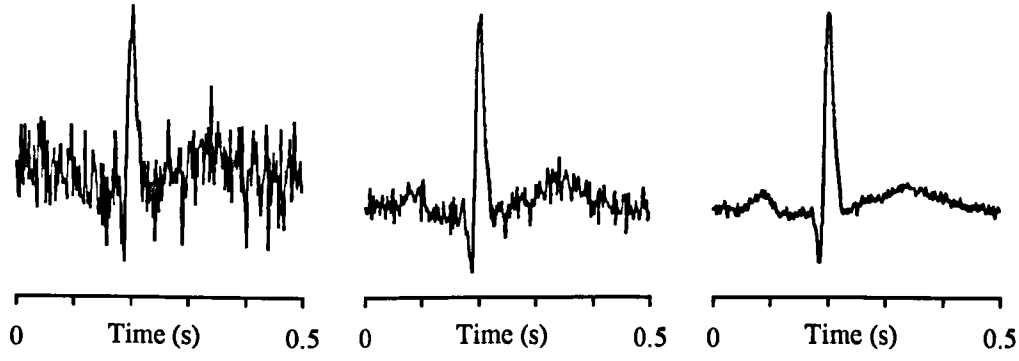


Figure 4.3(b): Averaged waveform for 1, 5 and 50 averages

Figure 4.4 shows the SNR of the average waveform, when different levels of baseline drift are added to the signal. Below -15 dB, some of the complexes are rejected by the FECG validation process.

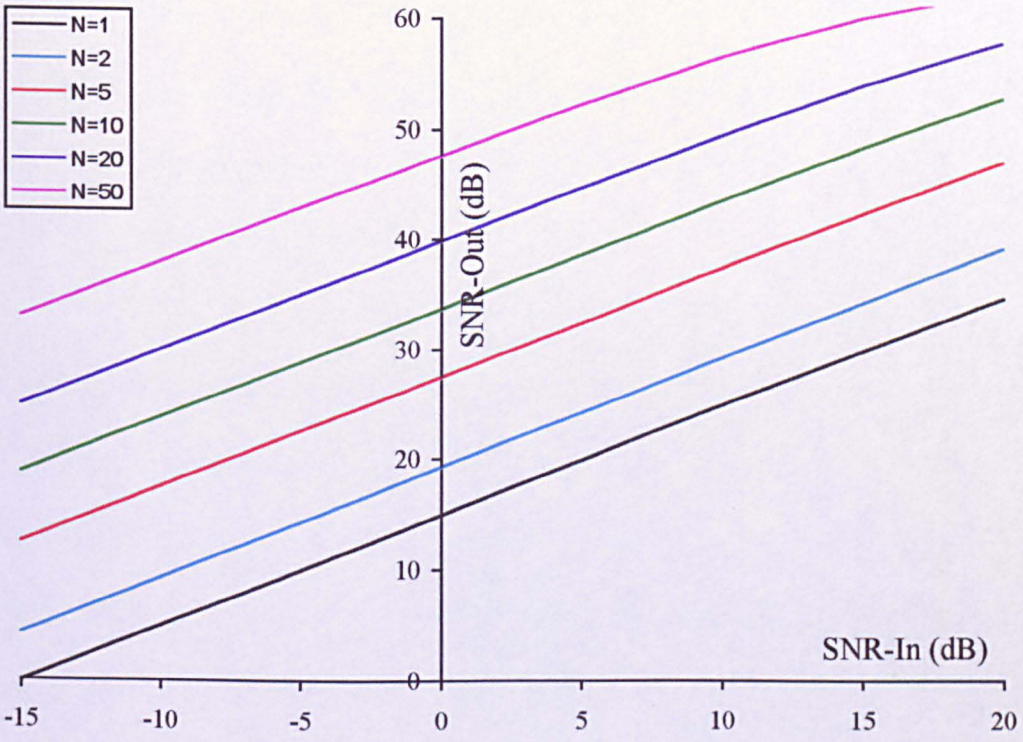


Figure 4.4: SNR improvement during coherent-averaging for baseline drift

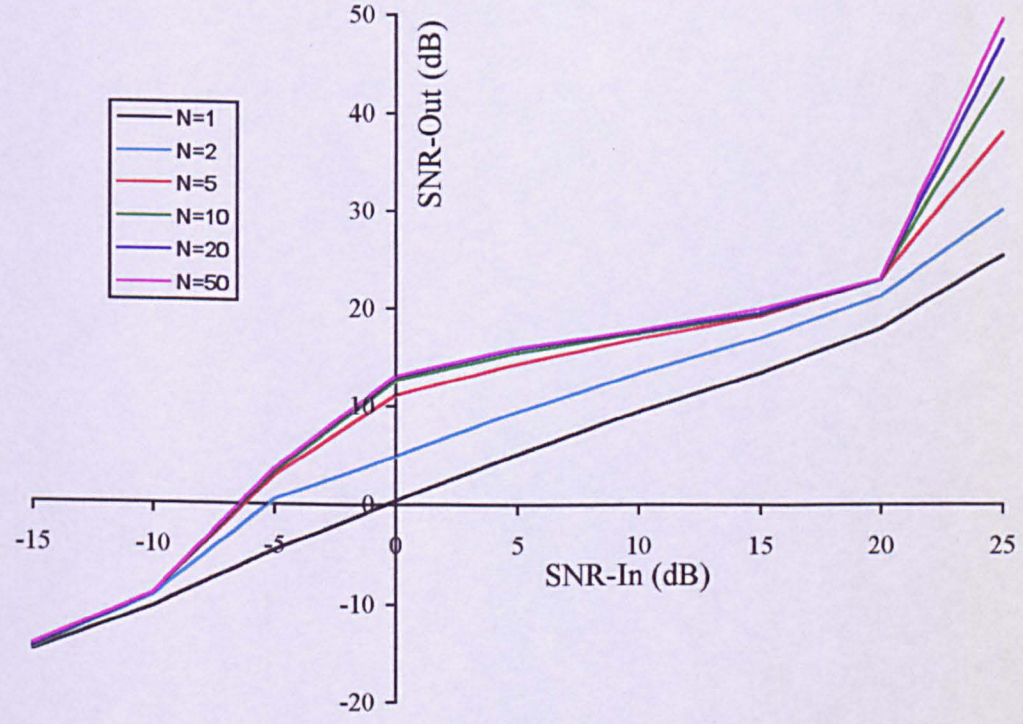


Figure 4.5: SNR improvement during coherent-averaging for mains noise

Figure 4.5 shows the SNR after averaging, compared to the SNR for a signal containing mains noise. Below -15 dB, the complexes are rejected by the FECG validation process, and no averaged waveform is obtained. No improvement in SNR is obtained from -15 dB to -10 dB.

With the SNR between -10 dB and 20 dB, an improvement of up to 10 dB is achieved with 5 averages. However, using N > 5, in the averaging process, gives little improvement in the SNR. The theoretical reduction in noise when more averages are used, is only true if each complex is aligned correctly. The effect of misalignment can be seen up to 20 dB, but above this point, when no misalignment occurs, the improvement in the SNR is considerable.

The improvement in SNR is demonstrated in Figure 4.6, which shows the raw ECG signal, with 10 dB of mains noise added, together with ECG complexes obtained using different numbers of averages.

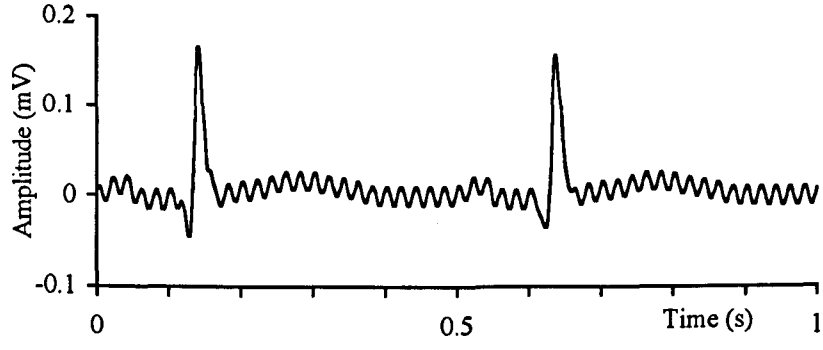


Figure 4.6(a): ECG signal with 10 dB mains noise

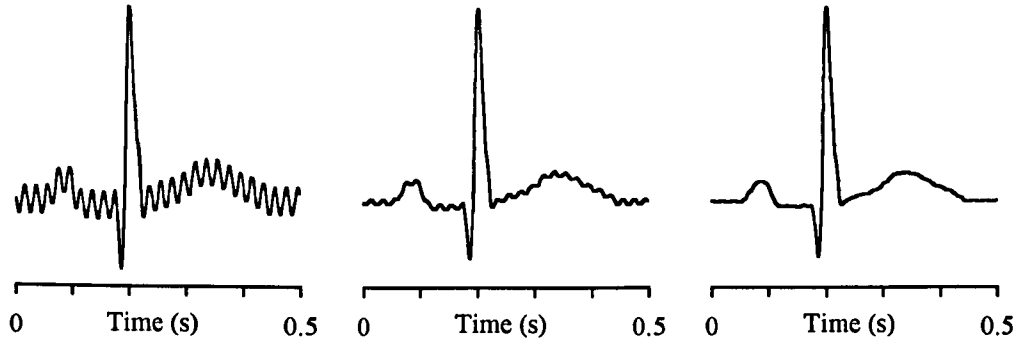


Figure 4.6(b): Averaged waveform for 1, 5 and 50 averages

Clearly to reduce noise, N should be as large as possible. However, this would not allow dynamic changes in the waveform to be observed as quickly. The transition time for the ECG parameters to settle will be investigated in section 4.5.

4.4 Linear modelling and parameter extraction

It was necessary to validate the linear model algorithms, to determine the accuracy of measurements obtained. Many parameters could be measured from the linear model, but only the PR time interval and the T height were used, as they were needed for the CI and T/QRS ratio. By validating the PR interval and T height calculations, only a limited validation of the linear model would be performed. However, they are an indication to the accuracy when measuring other time intervals and wave amplitudes.

4.4.1 PR interval measurements

In order to determine the minimum and maximum measurable PR intervals, the simulator was then used to generate two signals, one with an increasing PR

interval, and the other with a decreasing PR interval. The two signals were then passed through the analysis system. The system could determine the PR intervals between 70 ms and 134 ms. These values are more extreme than those that would be encountered. Beyond these values, the P wave lay outside the location specified in the linear model, so no P wave could be located.

The simulator was then used to generate signals with static PR intervals to determine the accuracy of these measurements. Part of a previous study has shown the PR interval to have a mean 102.7 ms, with SD 12.2 ms [Mohajer, 1994]. The PR interval was tested up to 2 SDs away from the mean (*see* Table 4.3).

Generated	**Detected**	
PR interval (ms)	**Average PR**	**RMS PR Error**
80	80 ms	0.0 ms
90	90 ms	0.0 ms
100	100 ms	0.0 ms
110	110 ms	0.0 ms
120	120 ms	0.0 ms

Table 4.3: Comparison of generated and detected PR intervals

4.4.2 T/QRS measurements

In order to determine the minimum and maximum measurable T/QRS heights, the simulator was used to generate signals with varying T heights. The two signals were then passed through the analysis system, with the result, that the minimum and maximum measurable T/QRS ratios were -0.80 and 0.83. The analysis system determines if the signal contains too much baseline wander by

measuring the average amplitudes of the P wave, QRS complex, and T wave (*see* section 2.4.2). If these values are significantly different, the waveform does not contribute towards an averaged complex. When these extremes of T wave are analysed, the average amplitude of the T wave is large enough for every complex to be rejected. No average can be calculated, and therefore no T height calculated.

The simulator was used to generate signals with static T heights to determine the accuracy of these measurements. Part of a study had determined the mean T height to be 7.57 units with standard deviation 6.63 units; the mean QRS height was found to be 43.66 units, with SD 4.98 units [Mohajer, 1994]. The Plymouth Perinatal Research Group define the normal T/QRS range to be -0.05 to 0.24, when using the STAN system, and, clinical decisions are made using these figures. The analyser was thus tested with T/QRS ratios from -0.40 to +0.60 for a good representative test range (*see* Table 4.4).

Generated	**Detected**	
T/QRS	**T/QRS**	**Error**
-0.4000	-0.3951	0.0049
-0.2000	-0.1975	0.0025
-0.0500	-0.0493	0.0007
0.0000	0.0001	0.0001
0.0500	0.0502	0.0002
0.2000	0.1996	0.0004
0.4000	0.3994	0.0006
0.6000	0.5928	0.0072

Table 4.4: Comparison of generated and detected T/QRS ratios

4.5 Parameter extraction under noisy conditions

So far, these parameters have been validated with no noise added. As noise is added, the accuracy of these measurements will deteriorate. However, as previously demonstrated, the averaging technique will reduce the noise in the ECG, and so improve the accuracy. This will be at the cost of tracking changes in these parameters more slowly.

An ECG signal was generated with a fixed PR interval of 100 ms, and T/QRS ratio of 10%. An increasing amount of mains noise was then added to the signal, varying from an SNR of 20 dB to -15 dB. The noisy signal was then passed through the analysis software several times, each time using a different number of complexes in the averaging process. The PR intervals and T/QRS ratios were then compared to the correct values at each SNR, and an RMS error calculated. The RMS errors are shown in Figures 4.7(a) and 4.7(b).

An ECG signal was generated with the heart-rate varying from 100 to 160 b.p.m as the PR interval varied from 80 to 120 ms. This should give a conduction index value of +1. Again a slowly increasing amount of mains noise was added to the signal, and this was passed through the analysis program. The RMS errors for the CI were calculated and are shown in Figure 4.7(c).

The above procedure for testing the accuracy of parameter extraction of the analysis software, under mains noise, was repeated for white noise (Figure 4.8) and respiration noise (Figure 4.9).

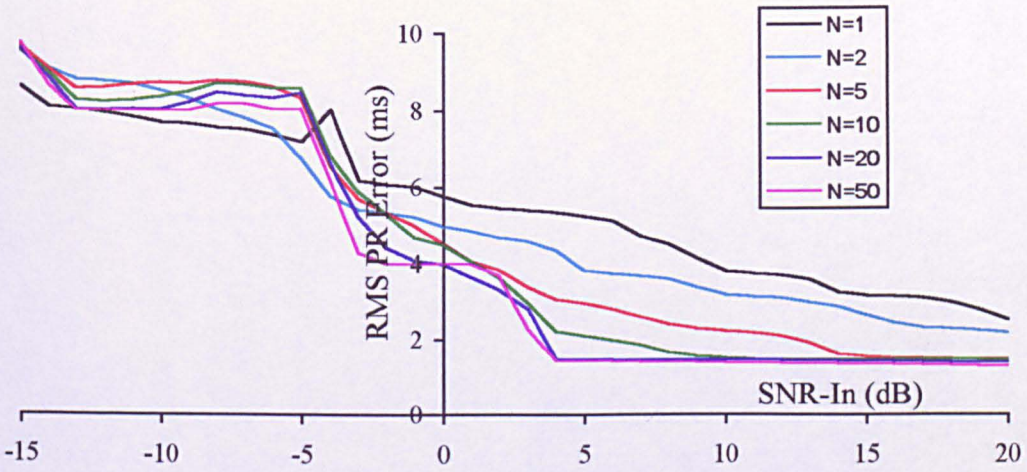


Figure 4.7(a): PR error with mains noise added

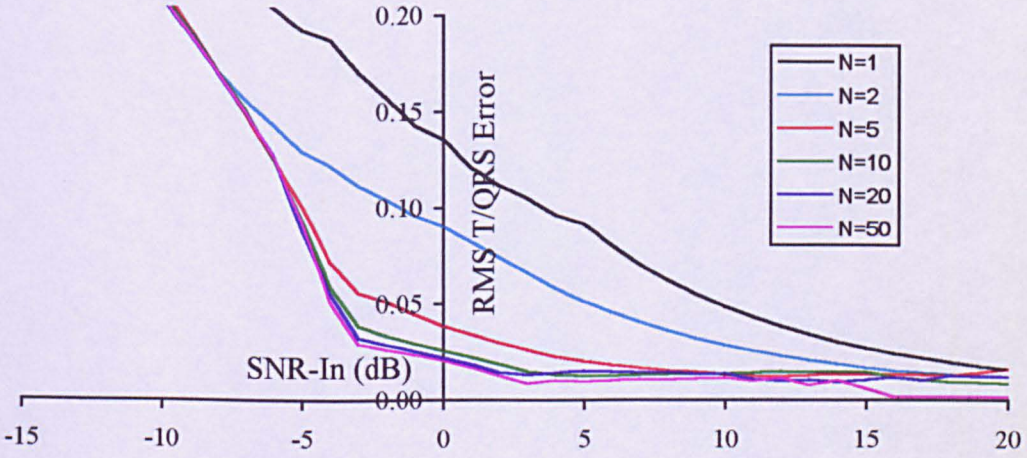


Figure 4.7(b): T/QRS error with mains noise added

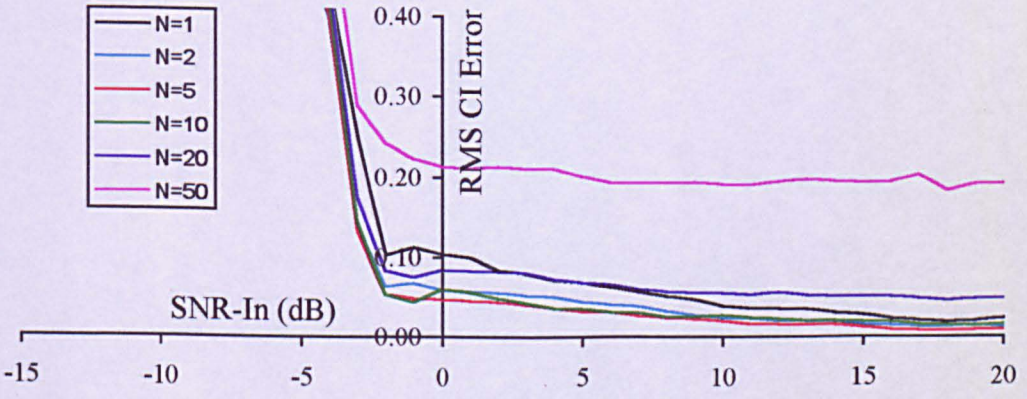


Figure 4.7(c): Conduction Index error with mains noise added

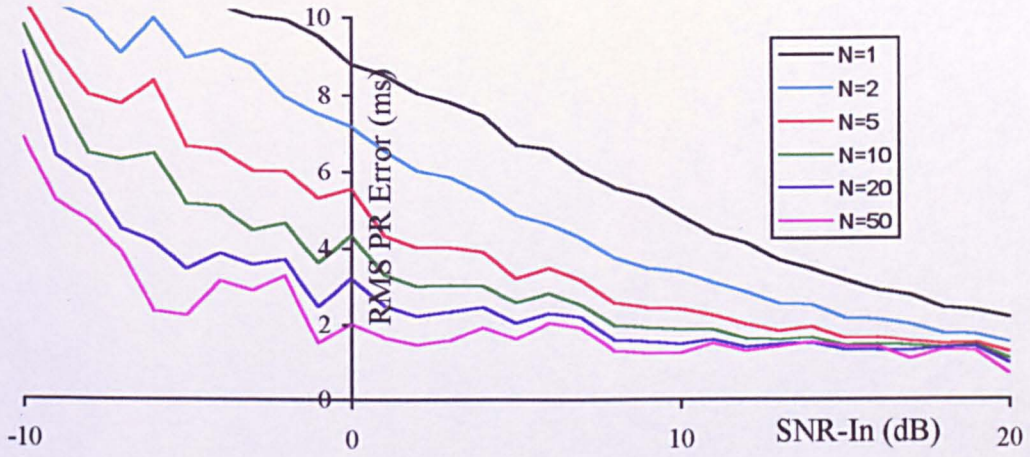


Figure 4.8(a): PR error with white noise added

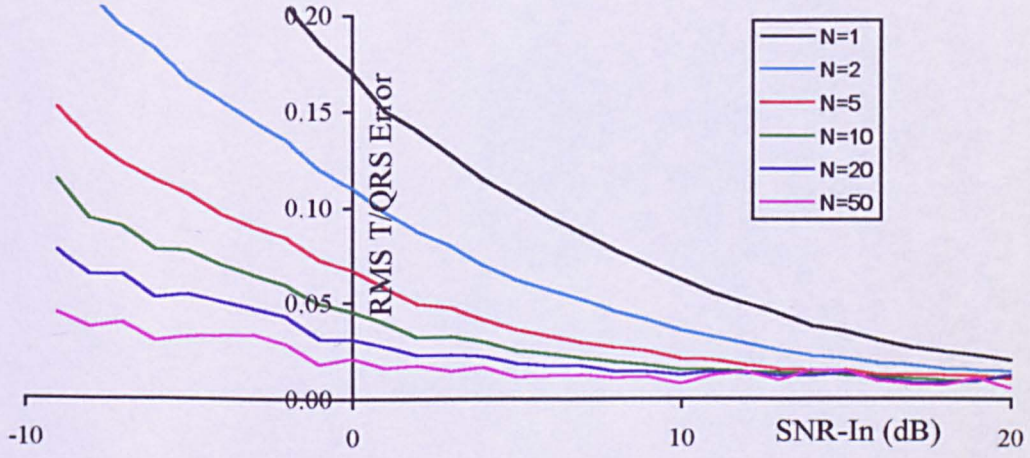


Figure 4.8(b): T/QRS error with white noise added

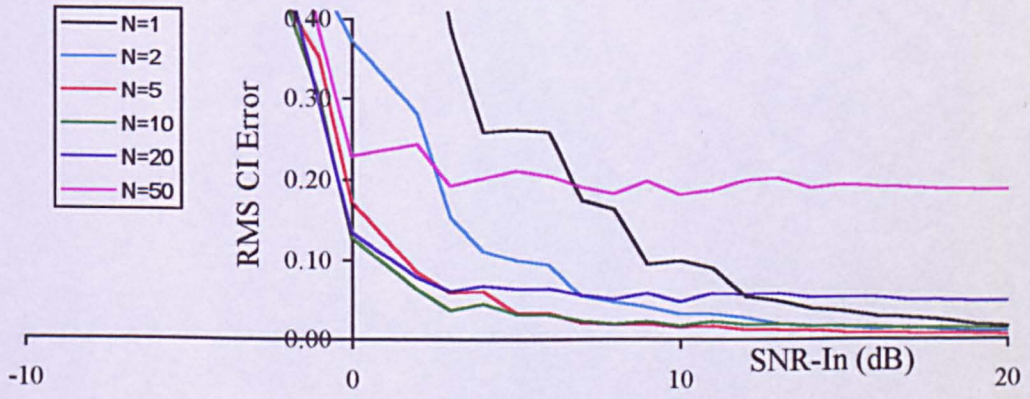


Figure 4.8(c): Conduction Index error with white noise added

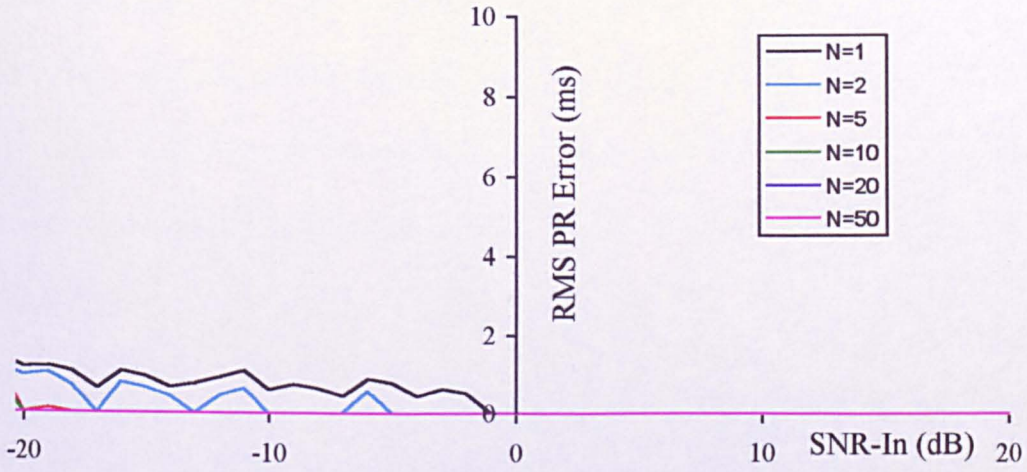


Figure 4.9(a): PR error with respiration noise added

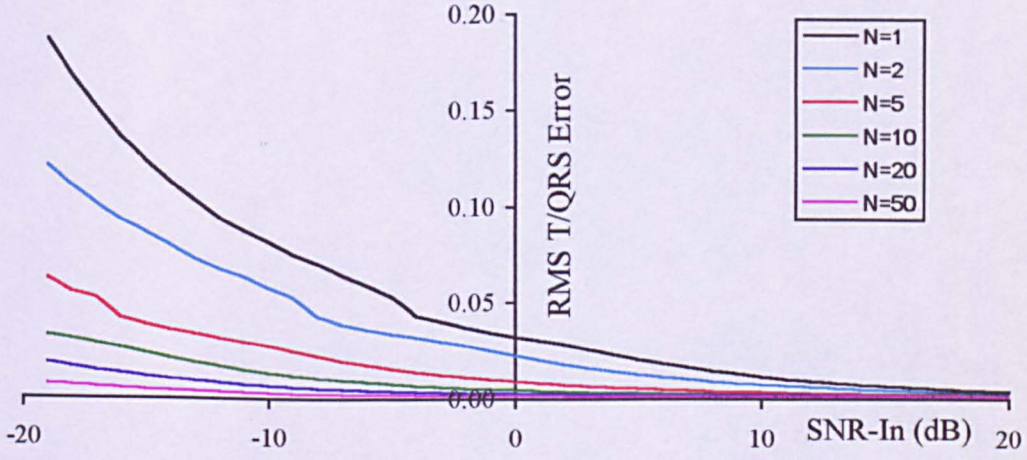


Figure 4.9(b): T/QRS error with respiration noise added

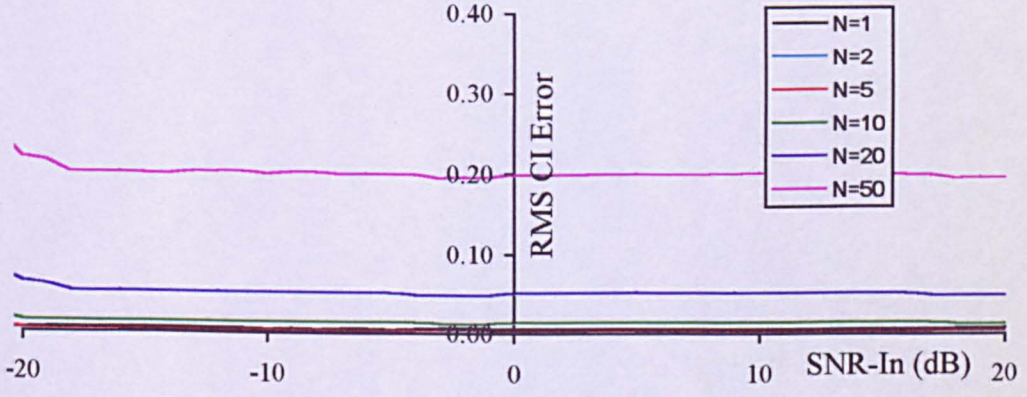


Figure 4.9(c): Conduction Index error with respiration noise added

From the graphs of PR and T/QRS errors, the more complexes used, the better the accuracy obtained. However, this does not allow dynamic changes in the waveform to be observed as quickly. This is demonstrated in the following figures, where the PR interval changes from 120 ms to 80 ms, and when the T/QRS ratio drops from 0.20 to 0.10.

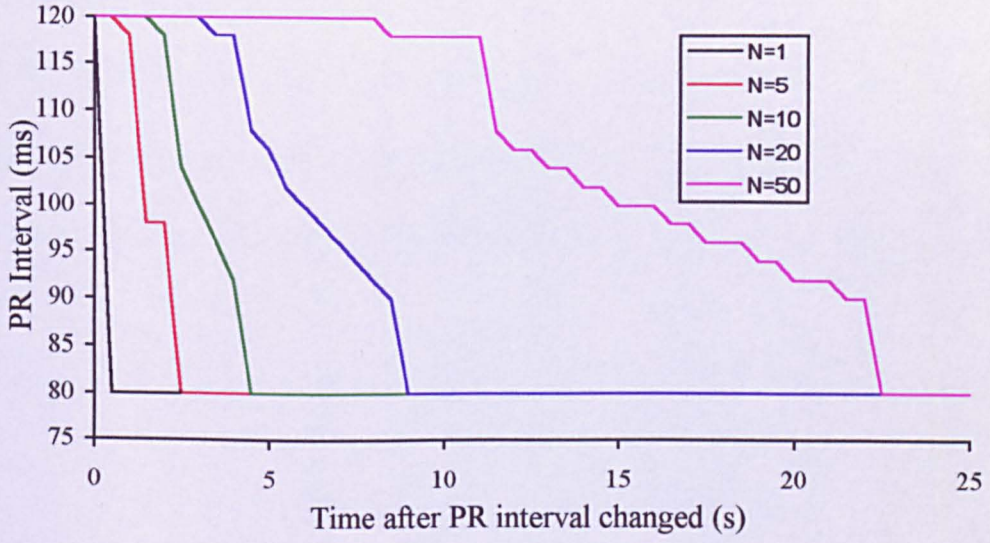


Figure 4.10: Tracking PR interval during averaging

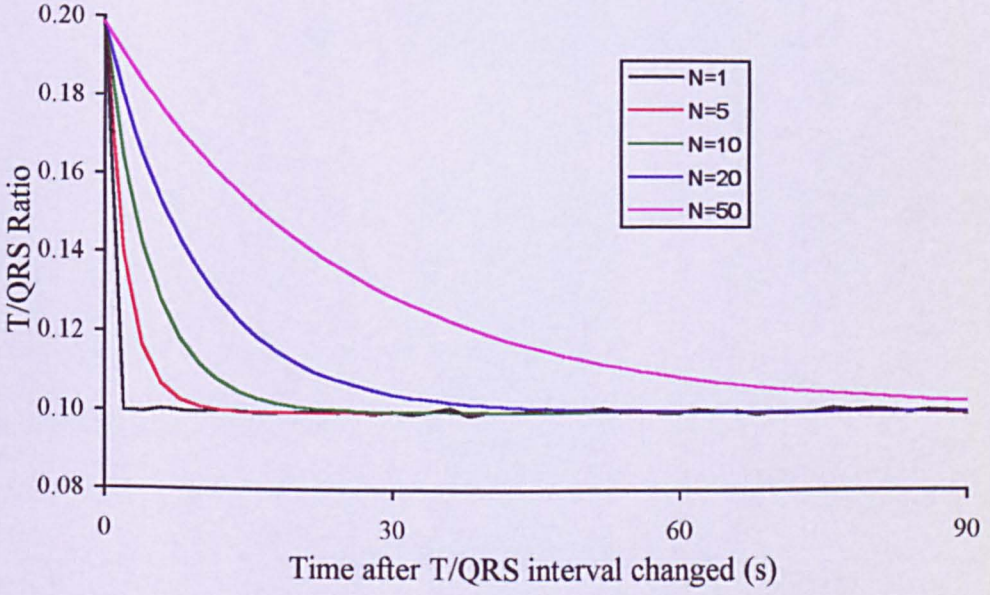


Figure 4.11: Tracking T/QRS ratio during averaging

The time-coherent averaging has previously been shown to be a low-pass filter [Scott, 1970], with a time constant of:

$$\tau = \frac{-T}{\ln\left(\frac{N-1}{N}\right)}$$

where T is the time between complexes. When a change in amplitude of any part of the underlying signal occurs, the amplitude in the averaged signal will reach 63 % of the value in τ and 86 % in 2τ. When the T/QRS ratio changes from 0.20 to 0.10, this relates to the ratio dropping 0.137 at time τ and 0.114 at time 2τ. Values of τ, corresponding to different values of N, are shown in Table 4.5.

N	τ(s)
5	2.2
10	4.7
20	9.7
50	24.7

Table 4.5: Values of τ for different values of N

Changes in the T wave height, clearly agree with these theoretical values. To measure the PR interval, the time between two waves is measured rather than an amplitude. The interval is measured with the aid of a linear model, so that the transient interval is no longer a simple exponential decay.

The effects of noise, when measuring the Conduction Index have been demonstrated, when the true CI value was +1. Clinical decisions are made upon the sign of the CI, so that, in this case, the errors have to be large before the measured CI becomes negative. However, errors when the CI is small may cause the measured value to be of the opposite sign to the true sign. Therefore the ECG simulator was used to generate signals with CI values of -0.3, -0.1, 0, +0.1 and +0.3, under different levels of white noise (N has been fixed at 10). The ranges of the CI, extracted by the analysis system, are shown in Figure 4.12.

Clearly, depending on the level of noise in the signal, it is possible to measure positive values, when the true value is negative and vice versa. At a noise level of 0 dB (which may be expected during a labour), when the *measured* CI value is between -0.3 and +0.3, the true value may be either positive or negative. This would indicate that in some cases, clinical decisions are made on measured CI values, which are, in fact, of the opposite sign.

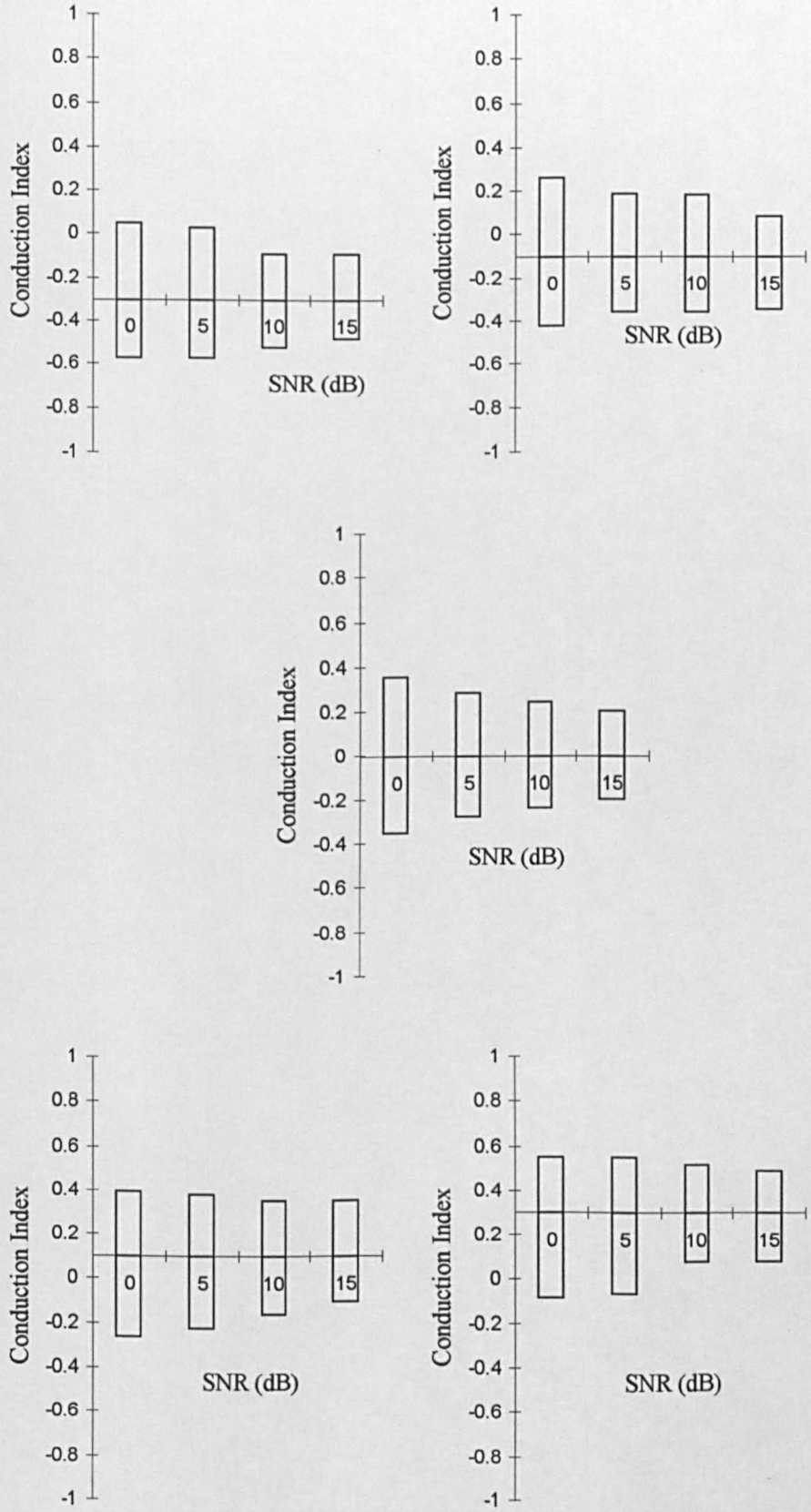


Figure 4.12: Ranges of measured CI values when true CI values are -0.3, -0.1, 0, +0.1, and +0.3.

4.6 Discussion

The accuracy of determining the heart-rate and a selection of parameters have been shown to be very good when noise does not exist.

However, there will always be some noise in real recordings. The work in this chapter has demonstrated how the number of complexes, used in the coherent averaging, is linked to the improvement in SNR of the ECG waveform, and the improved accuracy of parameter measurement. This is at the cost of a delay for the measured parameters to settle. The choice of the number of complexes to average will always be a compromise between improved waveform enhancement and the ability to track changes. This translates to a compromise between the parameter accuracy, when the morphology is static, and when the morphology is changing.

The inaccuracy of the measured parameters, when the morphology is changing, is a consequence of the settling time due to the averaging of several complexes. If one is to decide upon the maximum time for any parameter to settle, one can determine the number of complexes required for averaging (*see* Figures 4.10 and 4.11). From this, the improvement in SNR can be seen (*see* Figures 4.2, 4.4 and 4.5) and the accuracy of measured parameters for a static morphology, under different noise levels (*see* Figures 4.7-4.9).

Therefore, for a measured parameter on which clinical decisions are based, it is important to decide upon over what time period these decisions will be made.

From this, the settling time of the parameter, and the number of complexes needed during averaging, can be determined. Furthermore, if the noise in the signal is measured, an indication to the accuracy of any parameter can be estimated. When clinical decisions are being made, it is obviously essential to know the accuracy of the data. A future improvement, therefore, to the analysis software would be to add an indication to the accuracy of the data being presented to the user.

However, in the case of a derived parameter, such as the Conduction Index, the best value of N can easily be determined. The example shown in Figures 4.7(c), 4.8(c) and 4.9(c) show that averaging too many complexes, will cause transient changes in the PR interval to be filtered, giving an error in the CI, whatever the noise. From these graphs, using N = 10 is a good choice for the time-coherent average. However, below -3 dB of mains noise, or 0 dB of white noise, the CI value should not be believed. Even when the noise levels decrease, but still at noise levels typical during a labour, a measured CI value with a magnitude below approximately 0.3, does not indicate if the true value is positive or negative.

Furthermore, in the example used to demonstrate the inaccuracies of measuring the CI (Figures 4.7 - 4.9), the PR interval and FHR both had a large range: The PR interval changed from 80 to 120 ms, as the FHR changed from 100 to 160 b.p.m. However, if the range of these changing parameters was not as large, the CI error would be greater. Clinical decisions are made on the CI if the CI remains positive for more than 20 minutes. Care should be taken not to act on an erroneous value.

A new index, the recurrence index, is presently being developed, to overcome the problems of the CI [Sayers, 1996]. Like the CI, it will rely on the relationship between the FHR, and PR interval, but the algorithm will be designed on a statistically-based technique. Hopefully, the recurrence index should therefore not be as erroneous as the Conduction Index.

If the T/QRS ratio is also averaged with N = 10, it has been shown that the T/QRS ratio error may be as large as 0.05 at noise levels of 0 dB.

Now that the analysis program has been validated, it is necessary to validate the full system, including the front-end. Chapter 5 explains the required frequency response of the front-end to leave the underlying ECG undistorted. The accuracy of the full system is then shown in Chapter 6.

5. Frequency content of the ECG and its relation to the design of the ECG front-end filter

5.1 Introduction

All ECG signals will contain both a DC offset and very low frequency noise, referred to as *drift*. With fetal ECGs, the amplitude of this drift may be much larger than the amplitude of the complexes. When the input signal is amplified, saturation will occur with loss of signal. A high-pass filter is used to remove this noise, but this may also remove some of the component frequencies within the signal. The cut-off frequency must be a compromise between removing unwanted noise, and distorting the underlying ECG signal. Therefore it is essential to know the frequency content of the ECG signal, in order to decide upon the cut-off frequency of the high-pass filter. Furthermore, knowledge of the frequency components will allow a better choice of Fetal Scalp Electrode, which has been shown to affect the FECG signal [Westgate et al., 1990, Fisher, 1993].

In 1967, the AHA recommended that the frequency response should have -1 dB cut-off points at 0.14 Hz and 30 Hz, and -3 dB cut-off points at 0.05 to 60 Hz for adult ECG equipment [Kossman et al., 1967, Pipberger et al., 1975]. The phase response should also be linear over the latter region. However, in an experiment where the ECG was filtered with an almost rectangular digital high-pass filter, it was found that the cut-off frequency may be as high as the heart-beat frequency [vanAlste and Schilder, 1981]. For adults, this implied a frequency of 0.8 Hz. The AHA recommendations were probably based on filter methods, which introduce

phase distortion above the cut-off point, such as analogue filtering. This Chapter is therefore concerned with determining the frequency above which any filtering should have near constant gain and linear phase.

5.2 Theory

Much confusion has arisen into the frequency content of the ECG signal. One might look at a single ECG, and decide that there are frequencies down to d.c. However, the whole signal, containing many ECG complexes, must be considered to resolve this problem. The notion that the ECG spectrum will always contain near-dc components can be rejected with the following example. Consider a *theoretical* ECG signal with fixed morphology and a constant heart-rate of *HR* b.p.m.(beats per minute). This is a **periodic** signal of duration 60/HR s, and thus, by Fourier, its lowest frequency component is *HR*/60 Hz - a DC component may be present, but is not required. A constant heart-rate can be as low as 60 b.p.m., i.e. the lowest frequency component would be 1 Hz. Therefore, the high-pass filter would need a constant gain and linear phase delay above 1 Hz, in order to avoid distorting the waveform.

However, a fixed morphology waveform and fixed heart-rate have been assumed in the previous example. In practice, changes in the morphology of the waveform, and variability in the heart-rate, introduce additional components in the spectra. To find the lowest possible frequency within a real ECG signal, one would have to obtain many noiseless signals, and analyse the frequency content of each of these.

Obtaining a totally noise free signal from a fetal scalp electrode is impossible, so simulated data was used.

Modelling of the ECG was carried out in 2 parts. Firstly, a series of pulses representing the ECG complexes (termed the cardiac event series) was considered, and its frequency components investigated; then the pulses were replaced with the fixed morphology ECG wave. Replacing these impulses with the ECG complexes, can be considered as the **convolution** (in the time domain) of the impulse series with a single ECG complex, to give the ECG series (*see* Figure 5.1a).

In the frequency domain, the frequency spectra of the impulse series must be **multiplied** by the frequency content of a single ECG complex to give the frequency spectra of the ECG series (*see* Figure 5.1b).

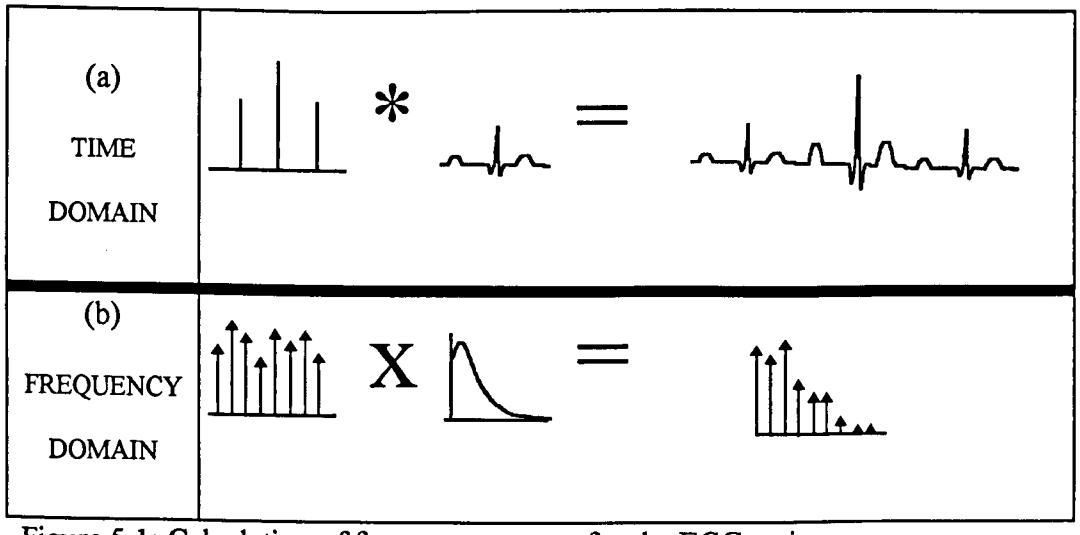


Figure 5.1: Calculation of frequency spectra for the ECG series

5.2.1 Modelling of HRV

It has been proposed that the neural modulation of the firing moments of cardiac pacemaker cells in the SA node can be modelled by integral pulse frequency modulation (IPFM) [Hyndman & Mohn, 1975]. IPFM consists of an integrator and a threshold detector as in Figure 5.2. This models the pacemaker frequency control via the rate of diastolic depolarisation. The IPFM input, *m(t)*, representing the neural influence on the SA node, is integrated. The integrated signal is passed to a comparator, and when this exceeds a fixed threshold, T_0, a pulse is generated, and the integrator reset. The output *s(t)* will then be the cardiac event series. T_0 will then be the mean interval between heart beats and the mean heart-rate, f_0, will be the reciprocal of T_0.

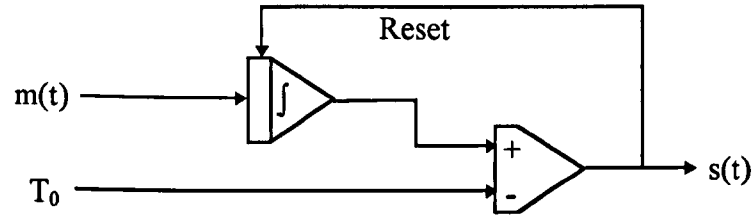


Figure 5.2: IPFM model

If we consider a single harmonic signal *m(t)*:

$$m(t)=1+m_p\cos\left(2\pi f_p t+\varphi\right) \tag{1}$$

where m_p is the pulse frequency modulation (PFM) index, f_p is the PFM frequency and φ is an arbitrary phase factor. An example of the impulse series, s(t), is shown in Figure 5.3.

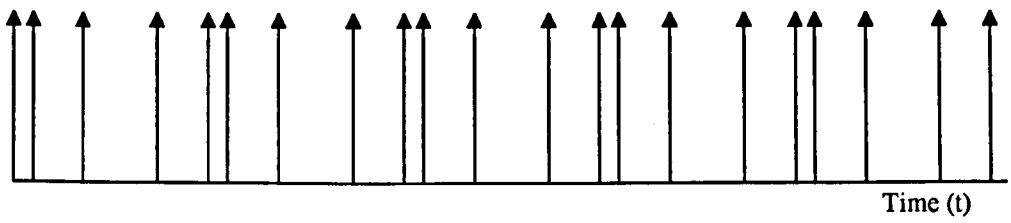


Figure 5.3: Impulse series response generated using IPFM model, using a single modulating sinusoid.

It has been shown [Bayly, 1968], that the output signal would be:

$$s(t) = If_0 + m_p If_0 \cos\left(2\pi f_p + \varphi\right) + 2If_0 \sum_{k=1}^{\infty} \sum_{n=-\infty}^{\infty} B_{k,n} \cos\left[2\pi\left(kf_0 + nf_p\right)t + \Phi_{k,n}\right]$$

$$\text{where} \quad \begin{cases} B_{k,n} = J_n\left(\dfrac{km_p f_0}{f_p}\right).\left(1 + \dfrac{nf_p}{kf_0}\right) \\ \Phi_{k,n} = 2k\pi f_0 \alpha + n\varphi + \dfrac{km_p f_0}{f_p} \sin\left(2\pi f_p \alpha - \varphi\right) \end{cases}$$

where I is the area of each pulse, $J_n()$ is the Bessel function of the first kind of the order n, and α is an arbitrary initial time instant. The complex two-sided IPFM spectrum is thus [TenVoorde et al., 1994]:

$$S(f) = If_0\delta(f) + \frac{m_p If_0}{2}\delta\left(f - f_p\right)e^{j\varphi} + \frac{m_p If_0}{2}\delta\left(f + f_p\right)e^{-j\varphi}$$
$$+ If_0 \sum_{k=1}^{\infty} \sum_{n=-\infty}^{\infty} B_{k,n}\delta\left[f - \left(kf_0 + nf_p\right)\right]e^{j\Phi_{k,n}} + If_0 \sum_{k=1}^{\infty} \sum_{n=-\infty}^{\infty} B_{k,n}\delta\left[f + \left(kf_0 + nf_p\right)\right]e^{-j\Phi_{k,n}}$$

Each term in the expression is of the form $A\delta(F)e^{j\theta}$. Each delta term gives an impulse at given frequency F, with amplitude A. The phase is given by the complex exponential term. The magnitude of the impulse spectra is shown in Figure 5.4.

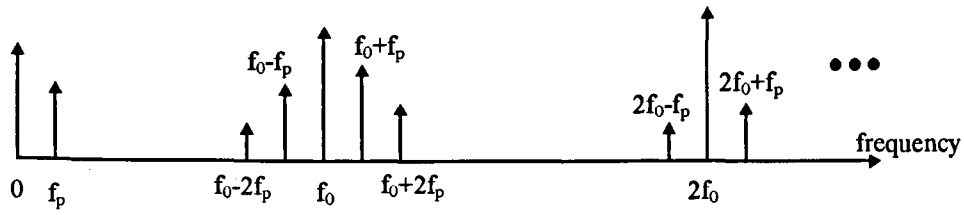


Figure 5.4: Impulse spectrum using IPFM model

Values of A can be both positive and negative and the above diagram shows the *magnitude* of A. In the following spectra, the magnitudes of the impulses could be similarly shown, with the addition of π to the phase whenever the magnitude is negative. However for changes in phase to be shown effectively, the phase must be left unaltered in this manner. Hence values of A will be plotted as both positives and negatives.

Example 1: Consider the case when f_0=2 Hz, m_p=0.2, and f_p=0.25 Hz, $\varphi=10^0$, I=1. This represents a mean heart-rate of 120 b.p.m., varying between 96 b.p.m. and 144 b.p.m. over a 4 second cycle. We are interested in the low frequency components, which are introduced by the modulation.

$$Amp(f_0 + nf_p) = 1*2*J_n\left(\frac{1*0.2*2}{0.25}\right).\left(1+\frac{n*0.25}{2}\right) = J_n(1.6)(2+0.25*n)$$

$$\Phi(f_o + nf_p) = 0 + 10n + \frac{180}{\pi}\frac{1*0.2*2}{0.25}\sin(-10) \approx -15.9 + 10n \qquad \text{deg}$$

Component	Frequency (Hz)	Amplitude	Phase (deg)
DC	0.00	2.000	0.0
f_p	0.25	0.200	10.0
f_0-5f_p	0.75	-0.002	-65.9
f_0-4f_p	1.00	0.015	-55.9
f_0-3f_p	1.25	-0.091	-45.9
f_0-2f_p	1.50	0.385	-35.9
f_0-f_p	1.75	-0.997	-25.9
f_0	2.00	0.911	-15.9
f_0+f_p	2.25	1.282	-5.9
f_0+2f_p	2.50	0.642	4.1
f_0+3f_p	2.75	0.199	14.1
f_0+4f_p	3.00	0.045	24.1

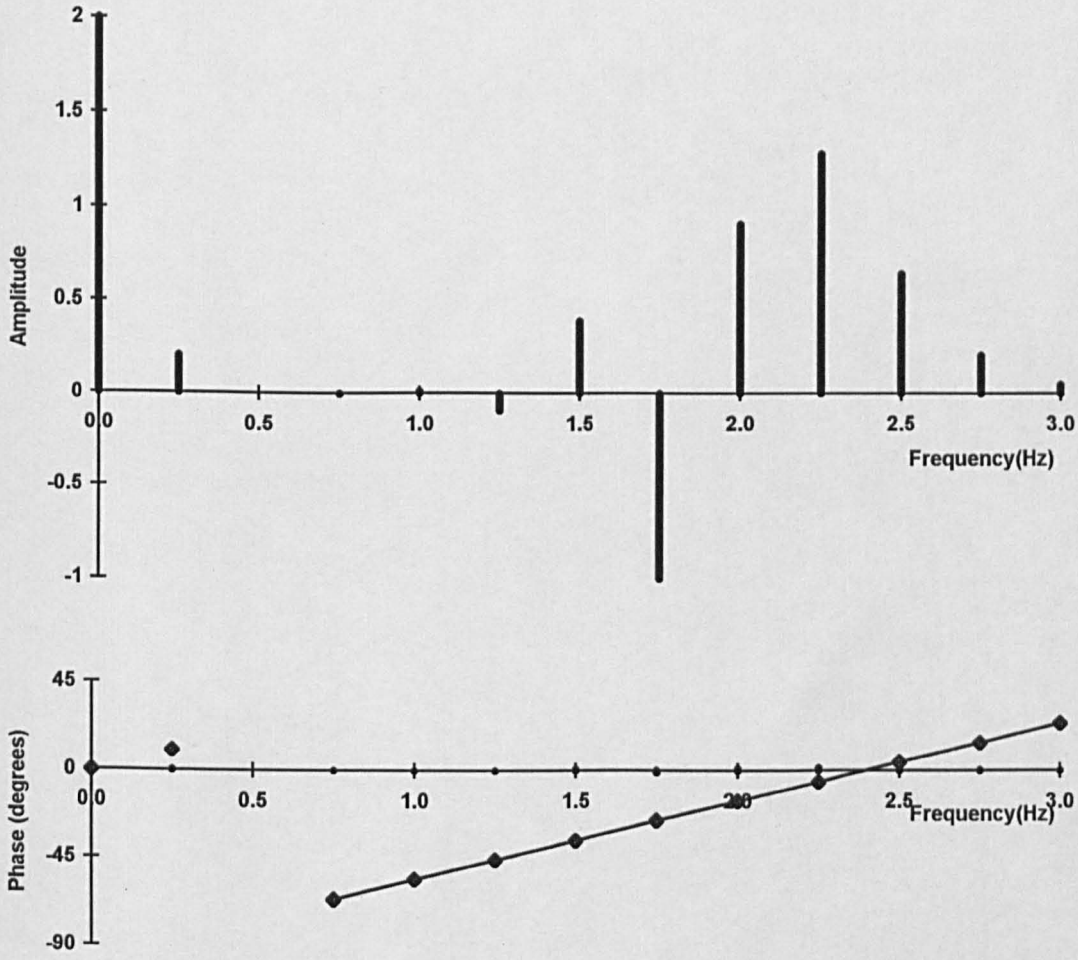


Figure 5.5: Frequency content of IPFM model example

To verify these theoretical results, the IPFM model was used to generate a series of pulses in the time domain for several cases. The data was output at 2 ms intervals; thus the amplitude was set to 500, so that the area remained unity. Within the IPFM model, integration of the continuous signal *m(t)* is carried out. However, a discrete signal was to be generated, and the integrator has to be modified. The integration at time *t*, over the previous sample interval, T_s, became:

$$I=\int_{t-T_s}^{t}\left\{1+m_p\cos\left(2\pi f_p\tau+\varphi\right)\right\}d\tau=\left[\tau+\frac{m_p}{2\pi f_p}\sin\left(2\pi f_p\tau+\varphi\right)\right]_{t-T_s}^{t}$$

$$=T_s+\frac{m_p}{2\pi f_p}\left[\sin\left\{2\pi f_p t+\varphi\right\}-\sin\left\{2\pi f_p\left(t-T_s\right)+\varphi\right\}\right]$$

The data was generated using the Windows ECG Simulator (*see* Chapter 3), which was adapted for the task. A DFT was then performed on this data, to obtain the amplitudes and phases of the components below 3 Hz. These results agreed with the above theoretical results.

5.2.2 Amplitude Modulation

The IPFM model has been developed to include sinusoidal changes in amplitude [TenVoorde et al., 1994]. These could then be used to model amplitude changes due to:

1. Physiological changes within the heart
2. Impedance changes from the electrode/patient contact.

The model is shown in Figure 5.6.

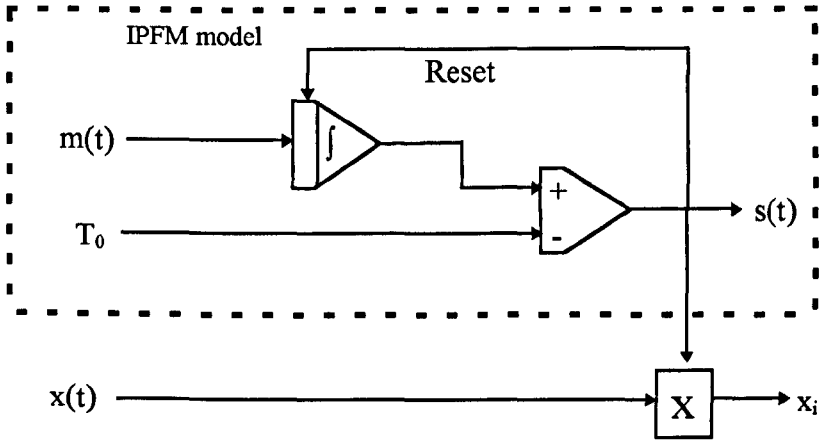


Figure 5.6: IPFM model with Pulse Amplitude Modulation

Output from the IPFM model is multiplied by x(t) to give the pulse amplitude modulated (PAM) signal x_i. The modulating sinusoid x(t) has DC level A_0, amplitude A_x, frequency f_x and an arbitrary phase factor θ.

$$x(t)=A_0+A_x\cos(2\pi f_x t+\theta)$$

The effect of the PAM on the impulse series is shown in Figure 5.7

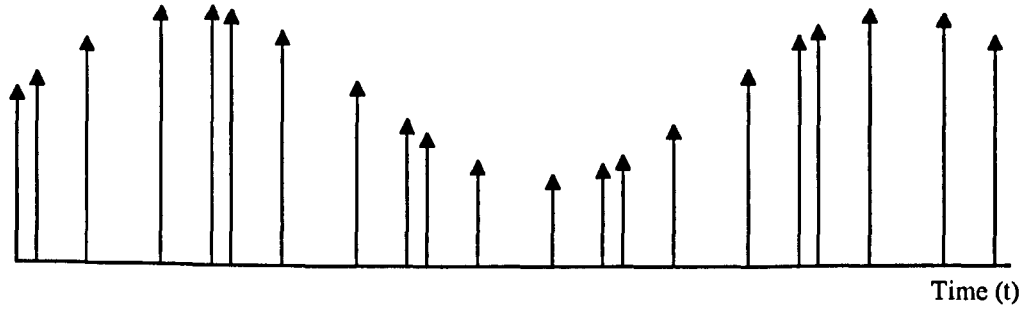


Figure 5.7: Impulse series generated by IPFM model with PAM

TenVoorde derived the frequency spectra to be:

$$X_s(f) = A(f)+B(f)+C(f)+D(f) \text{ where -}$$

$$\begin{cases} A(f) = \dfrac{A_0}{T_0}\delta(f) + \dfrac{A_0 m_p}{2T_0}\delta\left(f \pm f_p\right)e^{\pm j\varphi} + \dfrac{A_0}{T_0}\sum_{k=1}^{\infty}\sum_{n=-\infty}^{\infty} B_{k,n}\delta\left[f \mp \left(kf_0 + nf_p\right)\right]e^{\pm j\Phi_{k,n}} \\ B(f) = \dfrac{A_x}{2T_0}\delta\left(f \mp f_x\right)e^{+j\theta} \\ C(f) = \dfrac{A_x m_p}{4T_0}\delta\left[f \mp \left(f_p + f_x\right)\right]e^{\pm j(\theta+\varphi)} + \dfrac{A_x m_p}{4T_0}\delta\left[f \mp \left(f_p - f_x\right)\right]e^{\mp j(\theta-\varphi)} \\ D(f) = \dfrac{A_x}{2T_0}\sum_{k=1}^{\infty}\sum_{n=-\infty}^{\infty} B_{k,n}\delta\left[f \mp \left(kf_0 + nf_p + f_x\right)\right]e^{j(\theta\pm\Phi_{k,n})} \\ \qquad + \dfrac{A_x}{2T_0}\sum_{k=0}^{\infty}\sum_{n=-\infty}^{\infty} B_{k,n}\delta\left[f \mp \left(kf_0 + nf_p - f_x\right)\right]e^{-j(\theta\mp\Phi_{k,n})} \end{cases}$$

The magnitudes of the impulse spectrum are shown in Figure 5.8, with colour added to show the constituent parts of the equation.

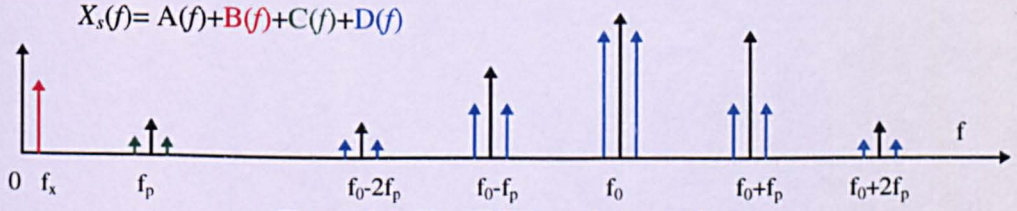


Figure 5.8: The IPFM model with PAM

Example 2: Consider example 1, but with the addition of PAM. A_0=1 for unity gain, A_x=0.5, and f_x=0.1 Hz, $\theta=0^0$. This represents the amplitude varying by 50% from the mean (e.g. a signal with a mean amplitude of 400 μV, will vary from 200 μV to 600 μV) over 10 seconds.

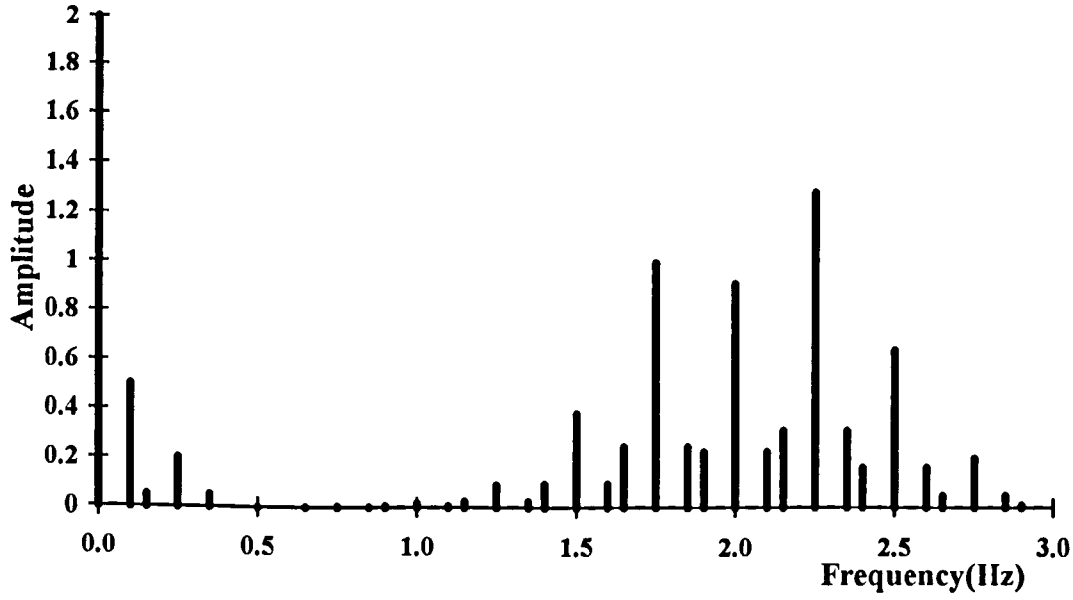


Figure 5.9: The IPFM spectrum with PAM

5.2.3 ECG spectrum

The above work has shown how the event-series can be predicted, but we are interested in the ECG spectrum. By using each of these pulses from the IPFM model to trigger an ECG wave, a full ECG signal can be created. As discussed earlier, this is, in fact, convolution in the time domain of the delta function with an ECG. Thus, to obtain the frequency spectrum of the continuous ECG, the IPFM frequency spectrum is multiplied by the ECG spectrum. Hence, the spectrum will consist of the same frequency components, but their magnitudes and phase delays will be altered.

The full ECG spectrum can be seen in Figure 5.10(a) and this is detailed for the frequencies we are particularly interested in Figure 5.10(b). Clearly this gives more weight to the impulses of lower frequencies.

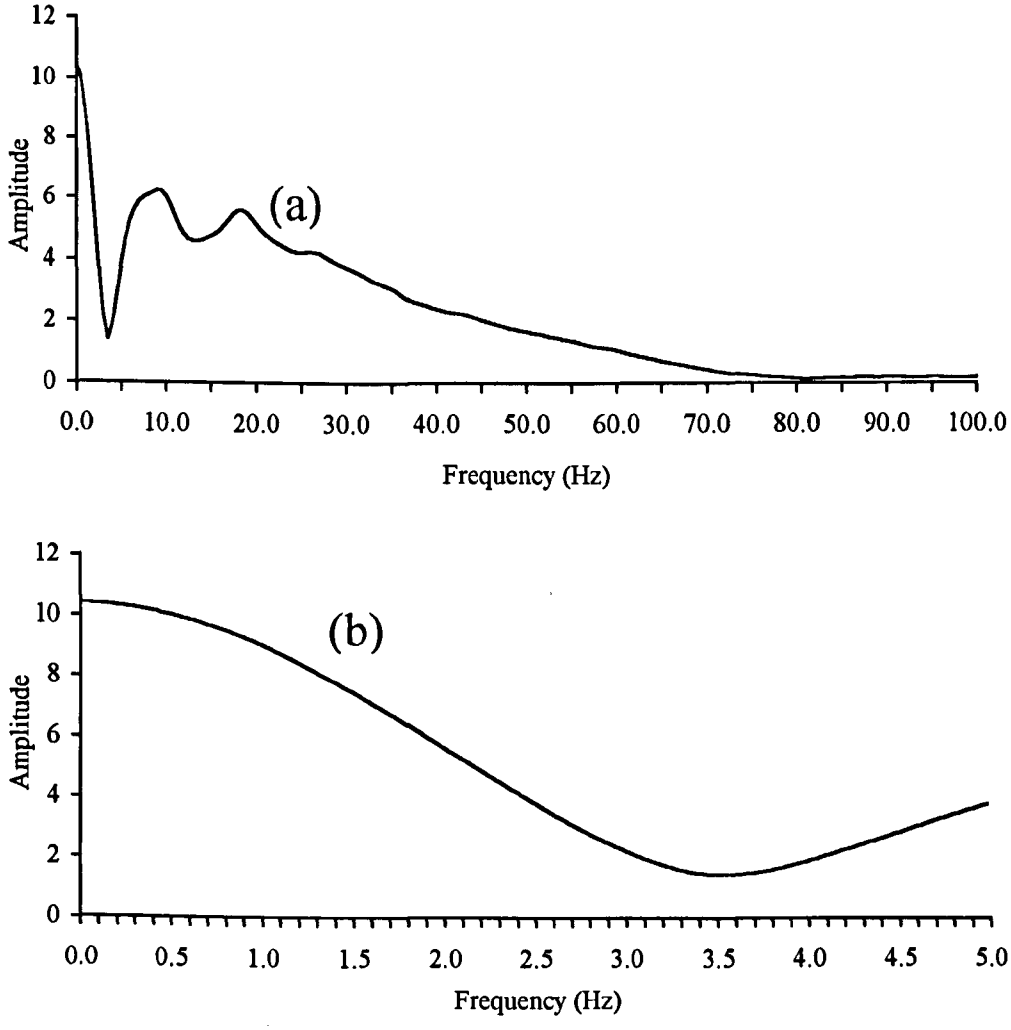


Figure 5.10: The frequency content of a typical ECG -

(a) From frequencies 0 - 100 Hz, (b) From 0 - 5 Hz.

Example 3: The above example using the IPFM model was then considered, but for a full ECG.

Each frequency component in the impulse spectra in Figure 5.9 is multiplied by the magnitude of the ECG spectrum (in Figure 5.10) to give the resultant spectrum in Figure 5.11 (black lines only).

Again, to verify these results, the IPFM model was used, within the Windows ECG simulator. Each impulse triggered the start of an ECG complex in order to obtain the full ECG signal. A DFT was then performed on the signal to confirm the predicted spectrum.

Furthermore, the ECG spectrum for example 2 was also calculated. This is also shown on Figure 5.11. The original PFM frequencies (in black) remain, with the addition of the PAM frequencies (in red).

Two distinct bands can be seen in the spectrum: the low-frequency band includes the PAM frequency(f_x), the PFM frequency(f_p), and the combination of both (f_p-f_x, f_p-f_x). The high frequency band includes all other frequencies, beginning with the sidebands of f_o.

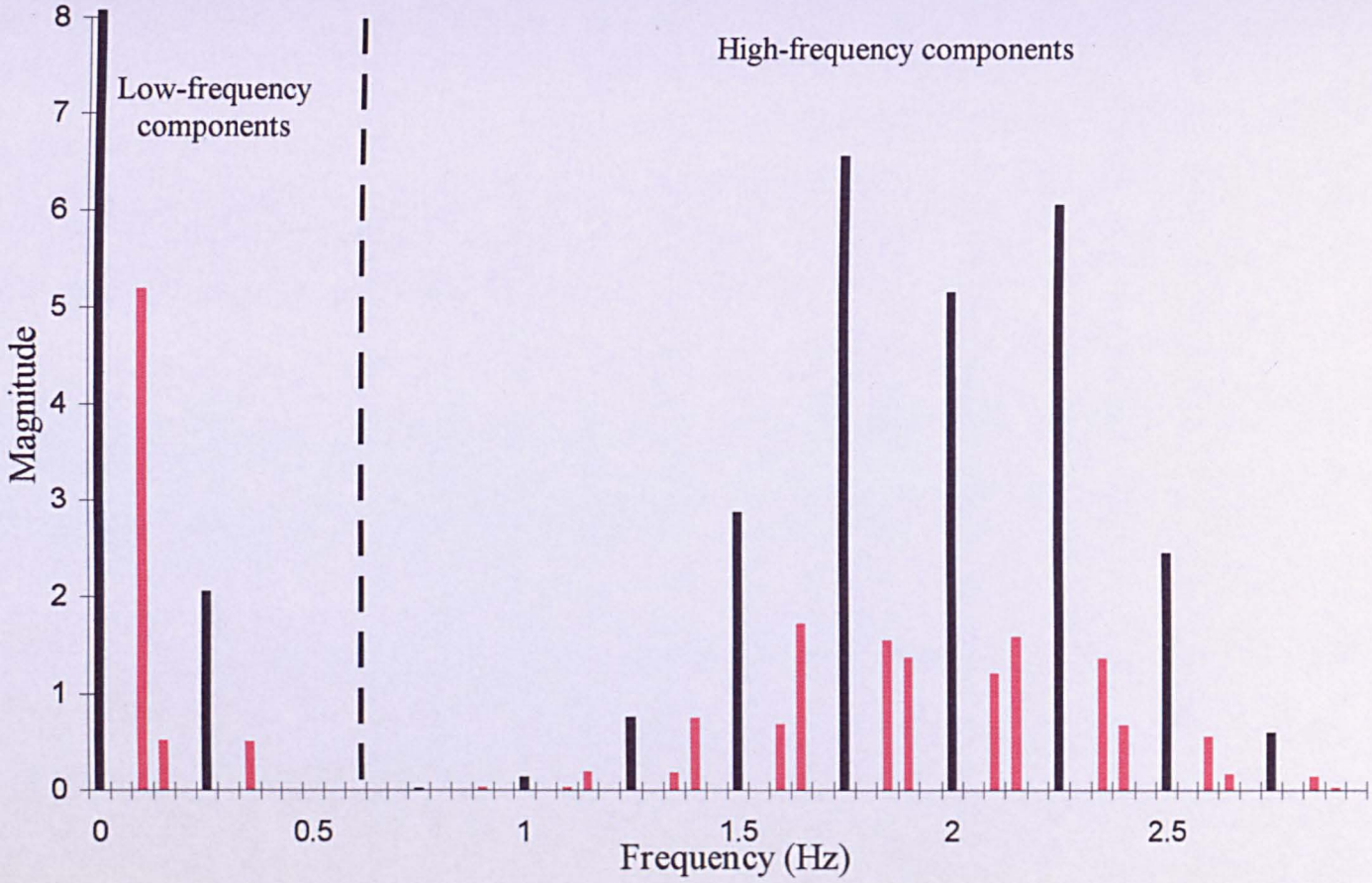


Figure 5.11: ECG spectrum applying IPFM model with PAM

5.3 Results - Effects on FECG parameters

The above work has been used to predict the theoretical frequency content of the FECG. However, filtering may occur in both the front-end electronics, and the electrodes. These effects can be modelled by adjusting the frequency spectrum.

The effects of filtering on the FECG can be studied by changing the FECG frequency components, and performing an inverse DFT. In the case of a *perfect* filter, the components can simply be removed. The effects of filtering on the parameters, can be determined by passing both the unfiltered and filtered signals through the analysis system, and comparing the results. The front-end filter could then be designed to pass those frequencies that left these parameters unaffected.

In order to determine the minimum filter cut-off frequency, it is necessary to determine the case in which the components are most significant. Furthermore, instead of just removing frequency components, the case has also been considered when the components are phase-shifted by 180^0, as this will have a more significant effect on the waveform.

Only the frequency components at these lower frequencies were required, and not the full spectrum. An FFT would be the fastest way of calculating the full spectrum, but for a small section, the DFT is quicker. All sinusoidal components up to the filter cut-off frequency, can then be subtracted from the original signal in the time domain. When phase-shifting the components by 180^0, the sinusoids are subtracted twice, clearly affecting the ECG the most.

There are many parameters that can be extracted from the FECG, and so it is necessary to define which of these are important. The effects on the following have been investigated:

1. R-R interval
2. P-R interval
3. T/QRS ratio

During analysis, the FECG waveform is time-coherent averaged before the parameters are extracted (*see* section 2.4.2). This will have the effect of averaging the beat-to-beat changes of the waveform. However, to determine the effects on the ECG, beat-to-beat changes have been investigated.

5.3.1 R-R interval

As discussed previously (*see* section 2.4.1), the R-peak detection algorithm is carried out by first passing the raw FECG through a bandpass filter. This has a centre frequency of 31 Hz and a bandwidth of 26 Hz. Therefore we would not expect the R-peak detection to be affected until frequencies of around 20 Hz are removed. Once the R-peak detection algorithm is affected, errors will occur in the calculated R-R intervals. The IPFM model of the ECG (example 3) was tested, and the ranges of R-R intervals when these frequencies are cumulatively removed, are shown in Figure 5.12.

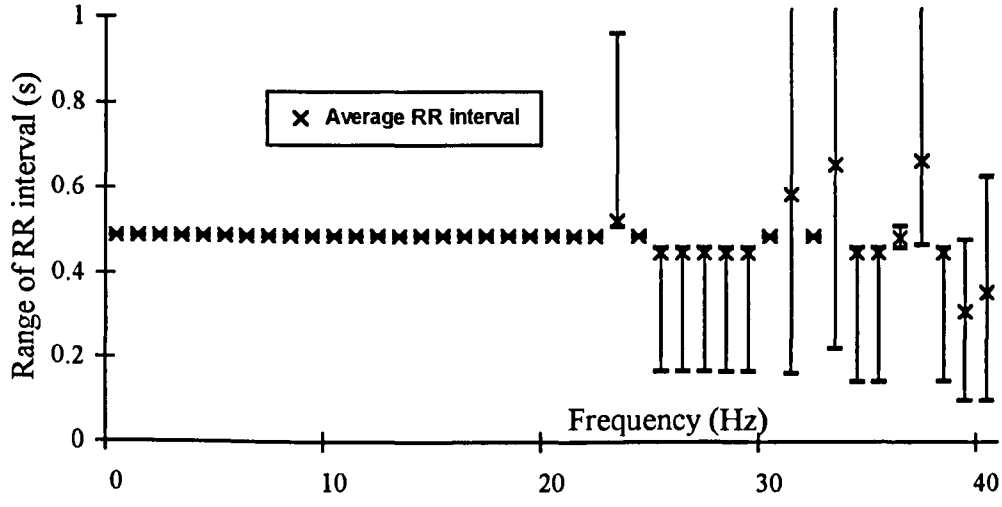


Figure 5.12: Effects of filtering on the R-R interval- both the average measured RR interval and the extremes of measured RR values are shown.

The RR interval is not affected until the signal is filtered at 23 Hz. This work concerns removing frequency components below about 2 Hz, so that the R-R interval will be unaffected, and will not be investigated any further.

5.3.2 P-R interval

First of all, the effects of removing the low-frequency band were investigated. To determine the worst case, the PFM and PAM components were filtered, when they were largest. The worst case is when $m_p f_o$ is at a maximum. With the assumption of minimum and maximum heart rates of 60 b.p.m. and 180 b.p.m. respectively, $m_p f_o$ is at a maximum when f_0=2 Hz, and the modulation depth m_p=0.5. Changes in the respiration amplitude from 200 μV to 600 μV are possible giving A_x=0.5 (A_0=1).

When these frequencies were removed, or phase shifted by 180^0, the PR interval was unaffected. Thus, the PR interval could only be affected if the higher frequency components were also removed. If frequencies under 2 Hz were to be of importance, they would originate from either f_0 or the PAM and PFM sidebands. It is necessary, therefore, to find the values in these models, which caused the largest components below f_0.

For the PAM, A_x=0.5 gave the largest components. In previous studies, it has been found that respiration has affected the ECG at 0.25 Hz, giving f_p=0.25 Hz [DeBoer et al., 1984, Cerutti et al., 1989, Novak and Novak, 1993]. This would affect the lowest frequencies. However, such large changes in signal amplitude in such short time periods (200 μV to 600 μV in 2 seconds), cause the R-peak detection on the analysis system to fail. Thus f_p was set to 0.01 Hz, and could be accounted for later.

In the case of the PFM model, the choice of f_0 and m_p was dependent upon the minimum heart rate, which we would assume to be 60 b.p.m. (1 Hz). This gave:

$$(1 - m_p) f_0 = f_{min} = 1 \text{ Hz}$$

Filtering was carried out for several choices of m_p and f_0, which met this criteria. One of the worst cases was with m_p=0.333 and f_0=1.5. f_p was also chosen at 0.25 Hz. The ranges of values of PR interval are shown in Figure 5.13. The PR interval is calculated to the nearest sampling interval (2 ms).

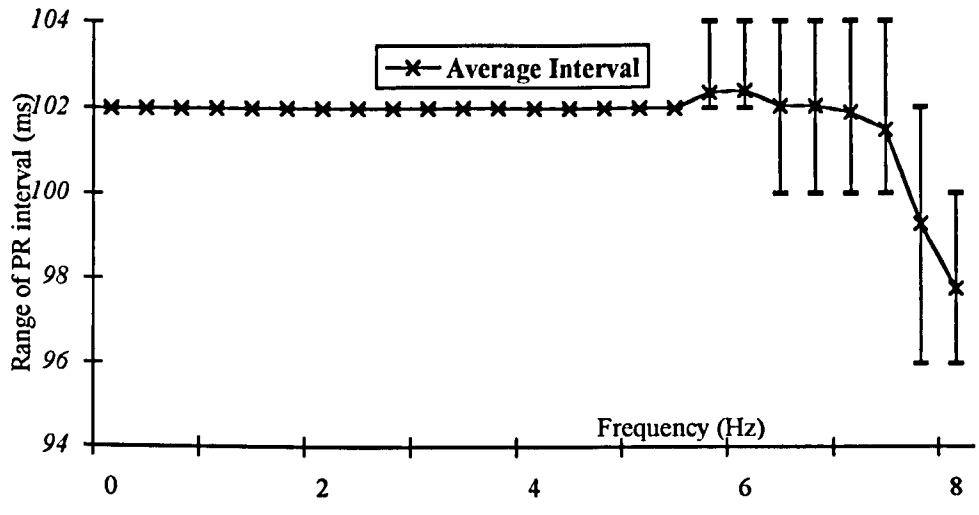


Figure 5.13: Effect of filtering on the PR interval - both the average measured PR interval and the extremes of measured PR values are shown.

Errors in the PR interval occurred once the frequency components from 0 to 5.66 Hz were filtered. As the cut-off frequency of the front-end will be significantly lower than 5 Hz, the effect of filtering on the PR interval is not a problem.

5.3.3 T/QRS ratio

Again, PFM and PAM components were filtered first of all. ECG waveforms produced at heart rates of greater than 160 b.p.m. do not have a complete T-wave. Thus the heart-rate must be limited to 160 b.p.m. In this case f_0=1.833 Hz (110 b.p.m.), and the modulation depth m_p=0.45. Again A_x=0.5. When these frequencies are removed, on a beat to beat basis, errors occurred in the T/QRS ratio. If the components were phase shifted by 180^0, errors occurred in the ratio by up to 0.04. However, the average T/QRS was only affected by a maximum of 0.005. In fact, when any of these low-frequency components were removed, an error occurred.

However, the T/QRS ratio is a slowly changing clinical measurement, and not acted upon over a beat-to-beat time period. Thus, it is the average T/QRS ratio that is important. Furthermore, the T wave is averaged over several waveforms during processing. Thus, it is only important that the average T/QRS ratio is left unaffected. Thus, the T/QRS ratio could only be affected if sidebands of f_o were removed. Choice of PFM and PAM parameters were chosen the same as the PR interval, except the heart rate had to be limited to 160 b.p.m., as complexes began to merge. An example of the effect on the average T/QRS ratio is shown in Figure 5.14, where f_o was set 1.5 and m_p to 0.333

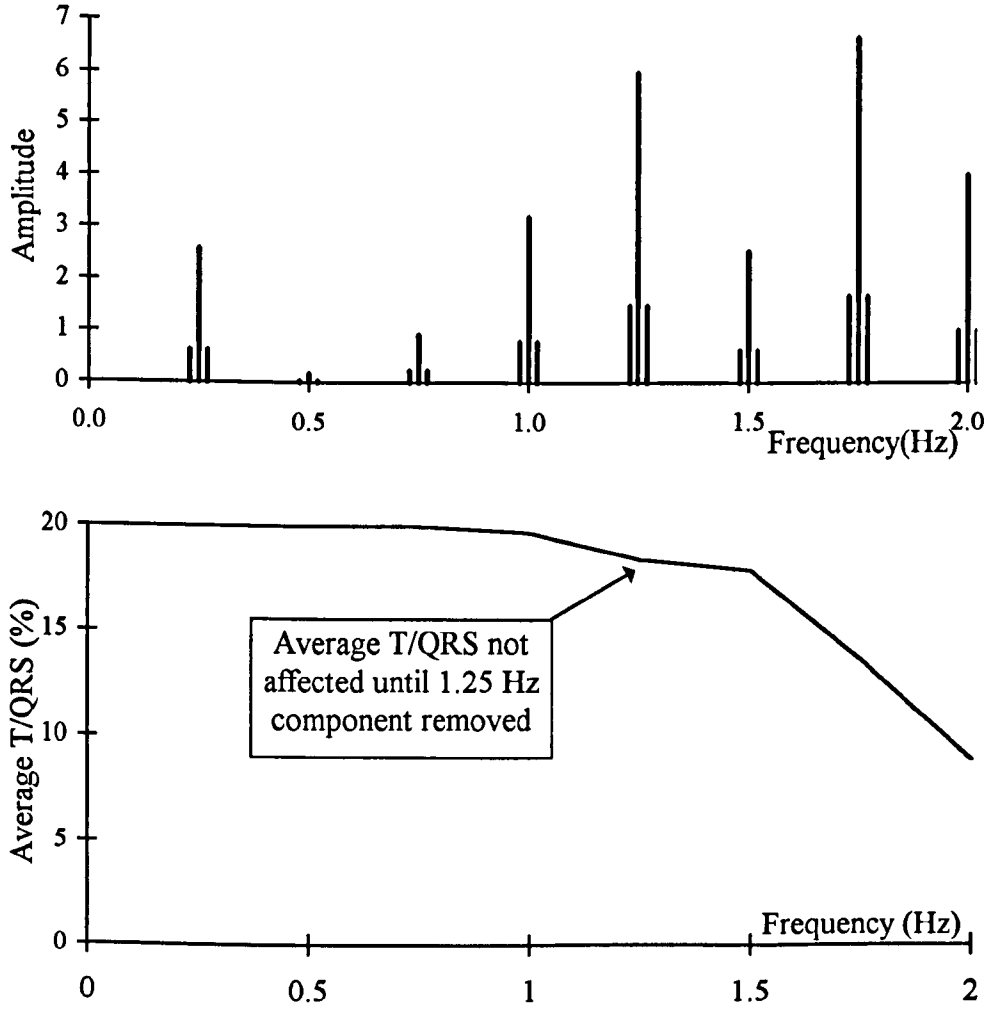


Figure 5.14: The ECG components, and the effect of filtering them on the average T/QRS ratio

The worst case occurred when there was actually no PFM, and the heart-rate was 60 b.p.m. In this case, the average T/QRS is affected by greater than 3% at 1 Hz.

5.3.4 Removal of mains frequency

Although this Chapter is mostly concerned with the filtering of low frequencies, the effect of removing the mains frequency is also of interest. Figure 5.15(a) shows a typical ECG complex. The 50 Hz component has then been calculated and removed from a signal containing these complexes, and is shown in Figure 5.15(b). As the removed component is, of course, a 50 Hz, sinusoid, the resultant waveform is Figure 5.15(a), with an apparent 50 Hz sinusoid added. If this 50 Hz component is altered in phase by 180^0, as in Figure 5.15(c), the resultant waveform, the effect is doubled.

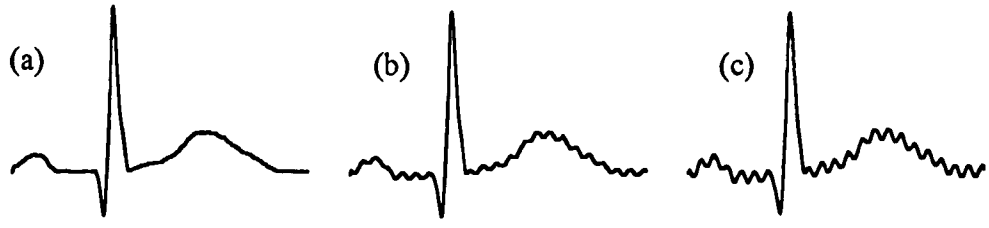


Figure 5.15: (a) An ECG signal, with (b) the50 Hz frequency component removed and (c), the component phase shifted by 180^0.

Therefore, if the 50 Hz component is arbitrarily removed, the effects will be similar to Figure 5.15(b). It is for this reason, that the Meridian front-end does not have a 50 Hz notch filter.

5.3.5 Effect of current front-end filtering electronics

As the frequency response of the front-end implemented in the analysis system is known, it is possible to simulate the effects of the front-end. The ECG spectrum may be multiplied by the frequency response of the filter. It has already been shown that the ECG is unaffected if:

1. the frequency response is constant above the fundamental frequency (> 1 Hz)
2. the gain ≤ 1 below the fundamental frequency

The front-end in this system does, however, have an increased gain at 0.1 Hz. This could only be significant if the signal contained frequency components near 0.1 Hz.

The case was therefore considered, when these components occurred at 0.1 Hz. Again to consider the effects of the PR interval: f_0=2 Hz, m_p=0.5 and A_x =0.5 A_0. For the components to coincide at the maximum gain: f_p=0.1 Hz, and f_x=0.01. The ECG spectrum, before the signal is passed through the front-end, is shown in Figure 5.16(a). This is multiplied by the front-end frequency response, to give the filtered frequency spectrum, shown in Figure 5.16(c). Although the component at 0.1 Hz was significantly increased, the PR interval was unaffected. A similar test on the T/QRS ratio showed that the average value was not significantly affected.

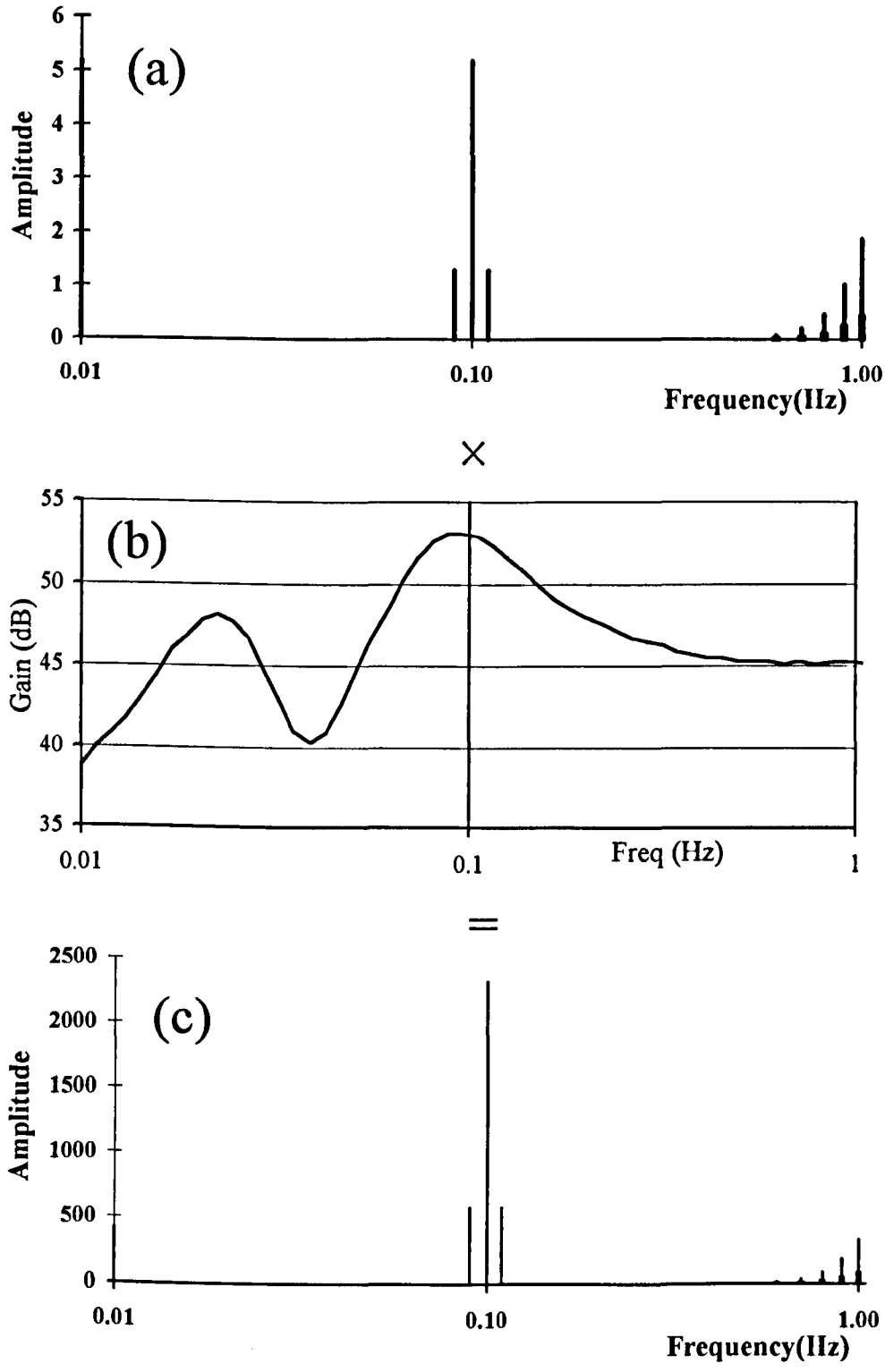


Figure 5.16: Each frequency component from the IPFM model with PAM (a), is multiplied by the respective frequency component in the filter response (b), to give the filtered spectrum (c).

5.4 Morphological changes and discussion of results

Using the model described, it has been shown that the RR and PR intervals are left unaffected when these lower frequencies are removed, or phase shifted by 180^0. The T/QRS is affected on a beat-to-beat basis, but as it is the average value that is important, this should not be a problem. The average value cannot be affected by high-pass filtering, unless the minimum heart-beat frequency is removed.

One of the main criticisms of this model, is the fact that the morphology of the waveform is fixed. An advancement to this model is to allow for changes in the morphology, and then to study the effects on the spectrum. The ECG may be considered as the sum of three constituent waves. Namely, the P-wave, the QRS complex and the T wave.

The frequency spectrum, H(f) has been calculated from the Fourier transform of a single ECG, C(f), and the impulse spectrum, I(f):

$$H(f) = C(f)I(f)$$

The magnitude of the frequency component, H(f), is therefore,

$$|H(f)| = |C(f)||I(f)|$$

An ECG complex is the sum of the P wave, QRS complex and T wave, so that

$$H(f) = C_p(f)I(f) + C_{qrs}(f)I(f) + C_t(f)I(f)$$

where C_p, C_{qrs} and C_t are the Fourier transform of the P wave, QRS complex and T waves.

The frequency spectra for these constituent waves are shown in Figure 5.17.

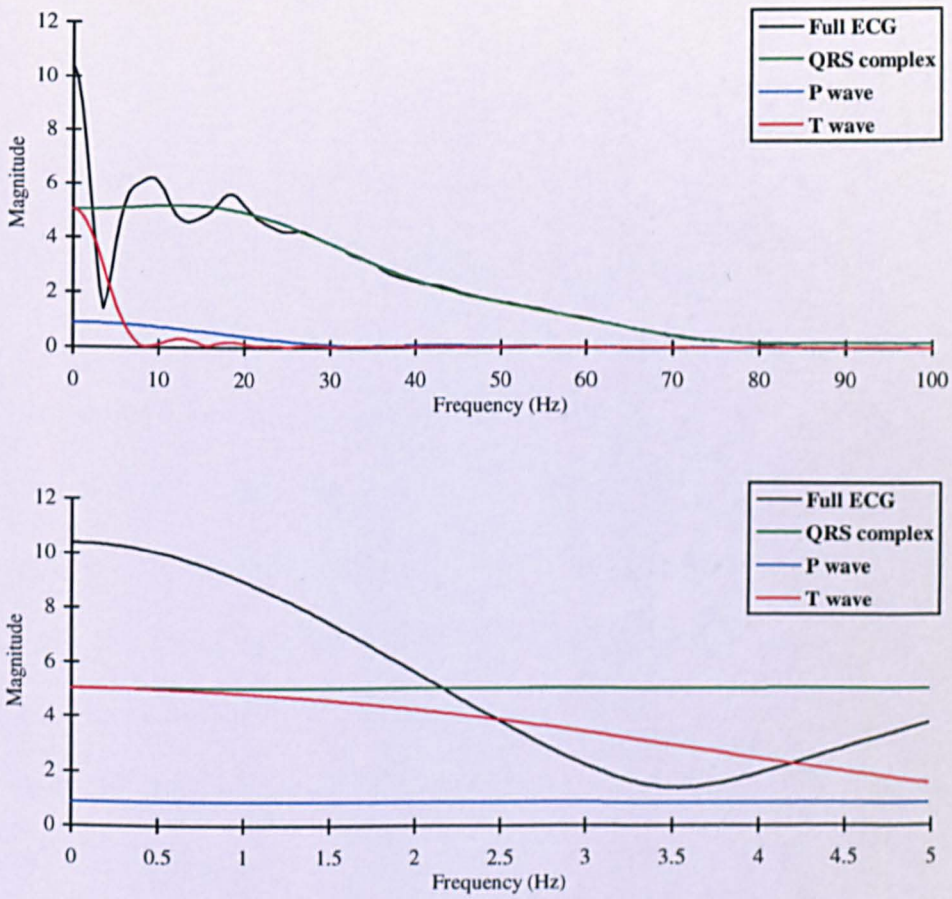


Figure 5.17: Frequency content of the constituent parts of the ECG

(This was calculated by taking the Fourier Transform of each constituent wave.)

In the above work, PFM and PAM have affected the whole ECG, but these ECG components may be affected separately. Consider the following example, where each component wave has a different set of PFM and PAM parameters (*see* Table 5.1).

Parameter	P wave	QRS complex	T wave
f_0 (Hz)	2.0	2.0	2.0
m_p	0.22	0.2	0.2
f_p (Hz)	10	10	10
A_x/A_0	0	0	0.5
f_x (Hz)	-	-	20

Table 5.1: PFM and PAM parameters for P wave, QRS complex, and T wave.

All 3 components are PFM modulated over a 10 second period. The slightly larger modulation depth of the P wave causes changes in the PR interval. Furthermore the T wave is PAM, causing changes in the T amplitude. The generated signals for the wave are then summed, and this is shown in Figure 5.18. Two ECGs generated in this example are shown in Figure 5.19. The morphological changes can clearly be seen. Between the two examples, the PR interval shortens, and the T/QRS ratio is reduced.

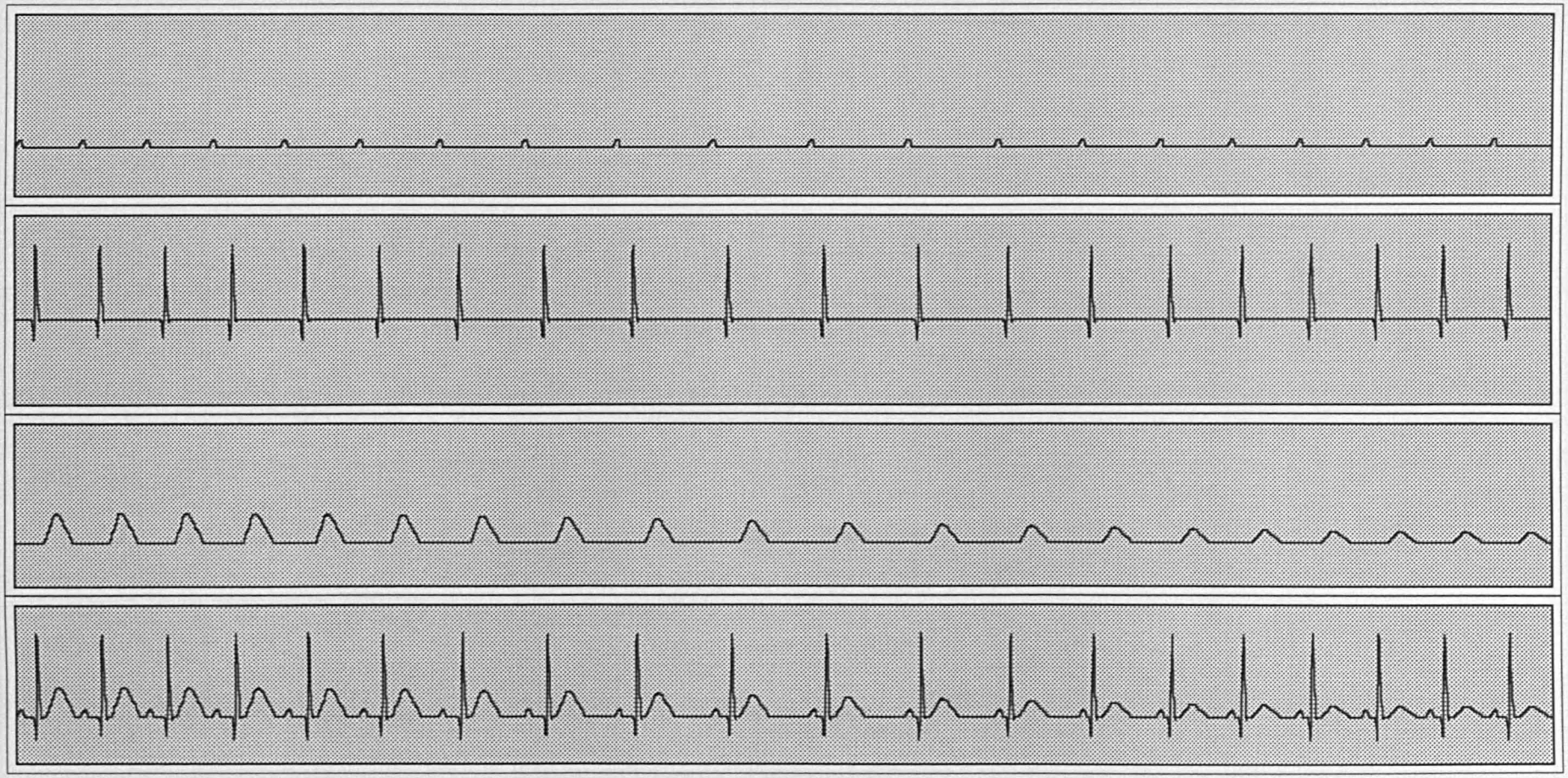


Figure 5.18: The ECG may be calculated as the sum of the P wave, QRS complex, and T wave.

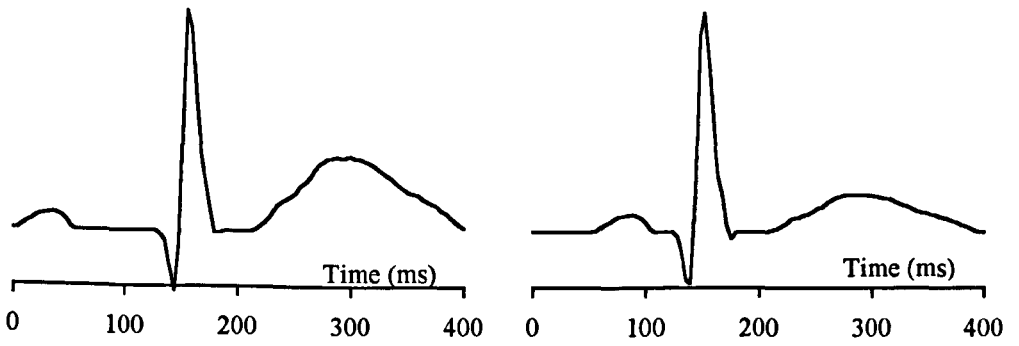


Figure 5.19: Morphological changes in the ECG complex

As each wave component has different PFM and PAM parameters, the impulse response is not the same for these waves, and the frequency spectrum is calculated using:

$$H'(f) = C_p(f)I_p(f) + C_{qrs}(f)I_{qrs}(f) + C_t(f)I_t(f)$$

where I_p, I_{qrs}, and I_t are the impulse responses for the respective waves.

Considering the magnitude,

$$|H'(f)| = |C_p(f)I_p(f) + C_{qrs}(f)I_{qrs}(f) + C_t(f)I_t(f)|$$

So that,

$$|H'(f)| \le |C_p(f)||I_p(f)| + |C_{qrs}(f)||I_{qrs}(f)| + |C_t(f)||I_t(f)|$$

Now the worst case for the impulse response has already been considered so that,

$$|I_p| \le |I|,\ |I_{qrs}| \le |I| \text{ and } |I_t| \le |I|$$

Thus,

$$|H'(f)| \le \{|C_p(f)| + |C_{qrs}(f)| + |C_t(f)|\}|I(f)|$$

At the low frequencies considered here,

$$|C(f)| \approx |C_p(f)| + |C_{qrs}(f)| + |C_t(f)|$$

Therefore,

$$|H'(f)| \leq C(f)|I(f)|$$

Comparing this to the fixed morphology frequencies,

$$|H'(f)| \leq H(f)$$

A changing morphology will cause the low frequency components to be never more than in the case of a fixed morphology. It has already been shown that the RR intervals, PR intervals, and average T/QRS ratio, are not affected by filtering the low-frequency components. It follows then, that these will not be affected, with a changing morphology.

However, the use of the IPFM model (with PAM) to model morphological changes has not been related to the physiological condition of the heart. Work would need to be carried out to justify this hypothesis. If this model is accurate, then it can be concluded that as long as the front-end, including electrodes:

1. passes all frequencies above, and including, the minimum heart beat frequency (approximately 1 Hz), without phase distortion, and
2. does not amplify the frequency components below the minimum heart rate,

then the RR interval, PR interval, and average T/QRS ratio will be undistorted.

6. Validation of the complete system

6.1 Introduction

The FECG analysis algorithms and software have already been validated in Chapter 4, using digitised data from the FECG simulator. It is nevertheless important to validate the whole system, including the analogue front-end. The front-end has been tested to ensure that it has an adequate frequency response, so that its addition, although introducing slight distortion to the FECG, should not affect the extracted parameters. This Chapter explains how both simulated and real data have been tested on the whole system.

The simulated data is generated from the FECG simulation program, described in Chapter 3. The addition of a D/A converter allows an analogue signal to be produced, which is then connected to the front-end. A database of FECG data, recorded from the fetal scalp, was also available and this would allow real data to be passed to the system. This raw analogue signal from the fetal scalp electrode has been passed through the front-end of the Conduction Index system (*see* Chapter 1), and been recorded onto magnetic tape. However the front-end, Sonicaid FM3R [Oxford Instruments], high-pass filtered the signal at 3 Hz. To replay the data, an equaliser must be used to boost those frequencies, which have been attenuated, thus restoring the original frequency components [Smith, 1983]. Although this method inevitably introduces more noise than occurred in the original recording, and causes phase distortion, the tapes were used to play real data into the Meridian.

6.2 Testing via use of simulator

In the previous Chapter, the IPFM model, with Pulse Amplitude Modulation, has been used to describe changes in the ECG. Techniques were given to find the frequency content of such a signal. The effects of the front-end filtering were also modelled. To further support this theory, a signal could be passed through the front-end, and the frequency content analysed.

The simulator was used to generate an IPFM signal, with PAM - the parameters for this are described in Example 3 of Chapter 5. This was then converted to an analogue signal and passed to the Meridian. The data was then recorded on the FECG analysis system, and the frequency spectrum analysed by taking a Fourier transform. This is shown in Figure 6.1.

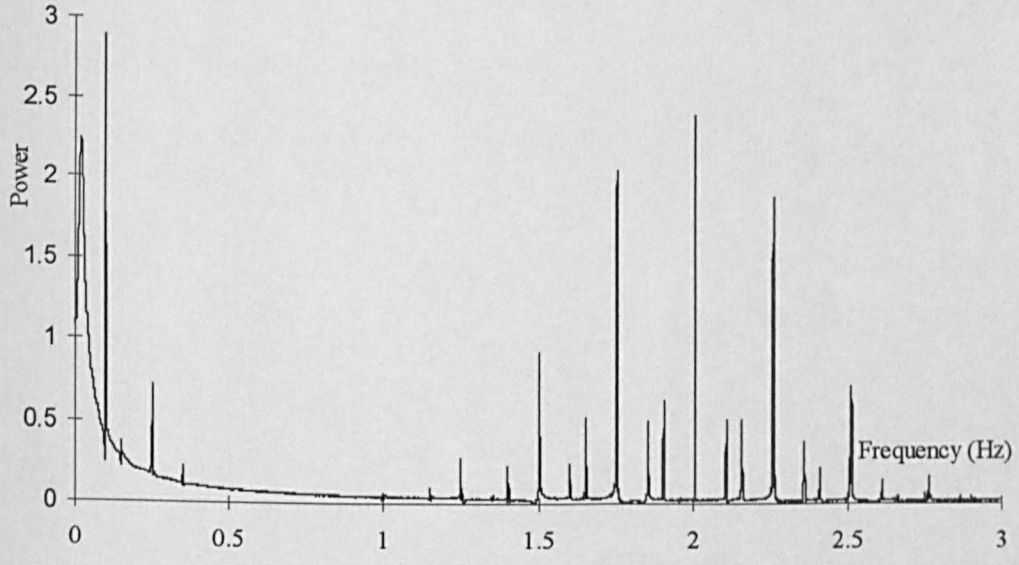


Figure 6.1: Frequency spectrum from IPFM model

As predicted in Chapter 5, the component at 0.1 Hz has been amplified by the front-end. The PFM frequency component can be seen at 0.25 Hz. Sidebands at multiples of this frequency can be seen around the (2 Hz) heart-beat frequency (i.e. at 1.25 Hz, 1.5 Hz, 1.75 Hz, 2.0 Hz, 2.25 Hz, 2.5 Hz, and 2.75 Hz). The PAM frequency is at 0.1 Hz, and other components can be seen on each side of the PFM components. This demonstrates the results in the previous Chapter.

In Chapter 4, it was detailed how signals with fixed HR, PR interval and T/QRS ratio were passed through the software system, to test the analysis software. When these signals were played through the front-end, the results produced from the software were consistent with those obtained previously.

6.3 Use of database

Finally, several cases were selected from the database. Both acidotic and non-acidotic were chosen, to demonstrate the changes in the Conduction Index value and the T/QRS ratio. Five of these cases are shown in the following figures.

Subject	pH indication	Figures
1	Non-acidotic	6.2 - 6.5
2	Non-acidotic	6.6 - 6.7
3	Non-acidotic	6.8 - 6.10
4	Acidotic	6.11 - 6.13
5	Acidotic	6.14

Table 6.1: Table of subjects shown in Figures 6.2 - 6.14.

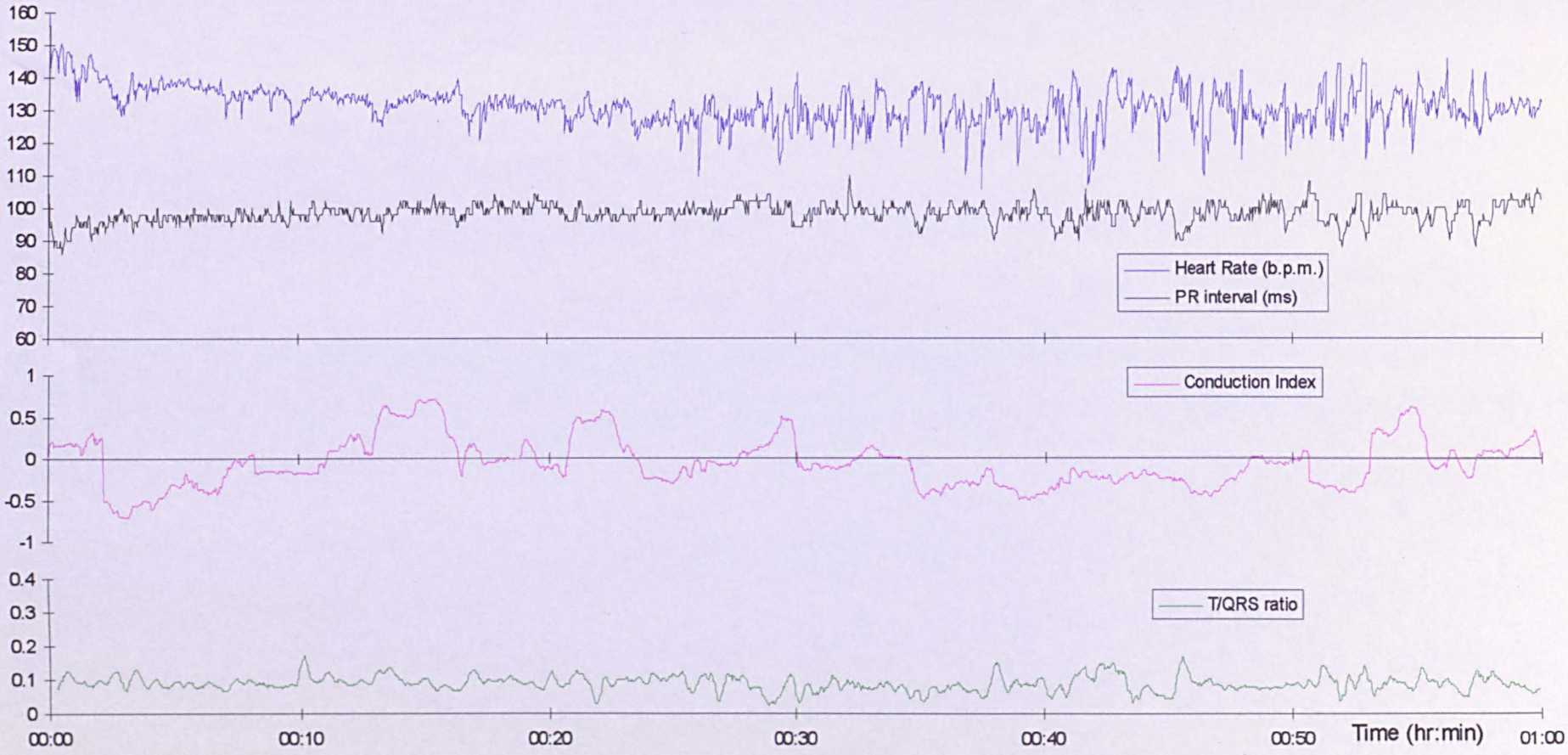


Figure 6.2: Subject 1 (Hour 1)

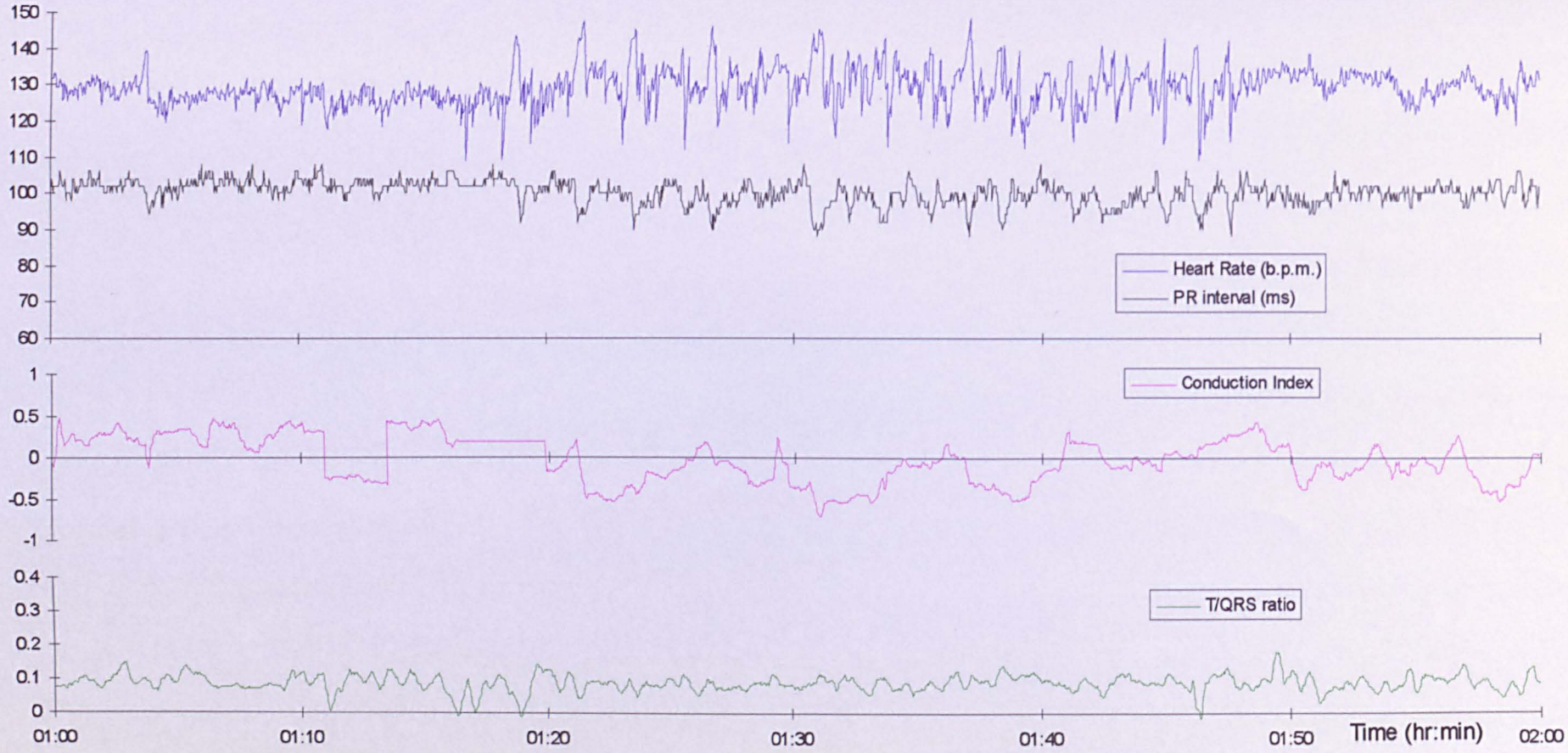


Figure 6.3: Subject 1 (Hour 2)

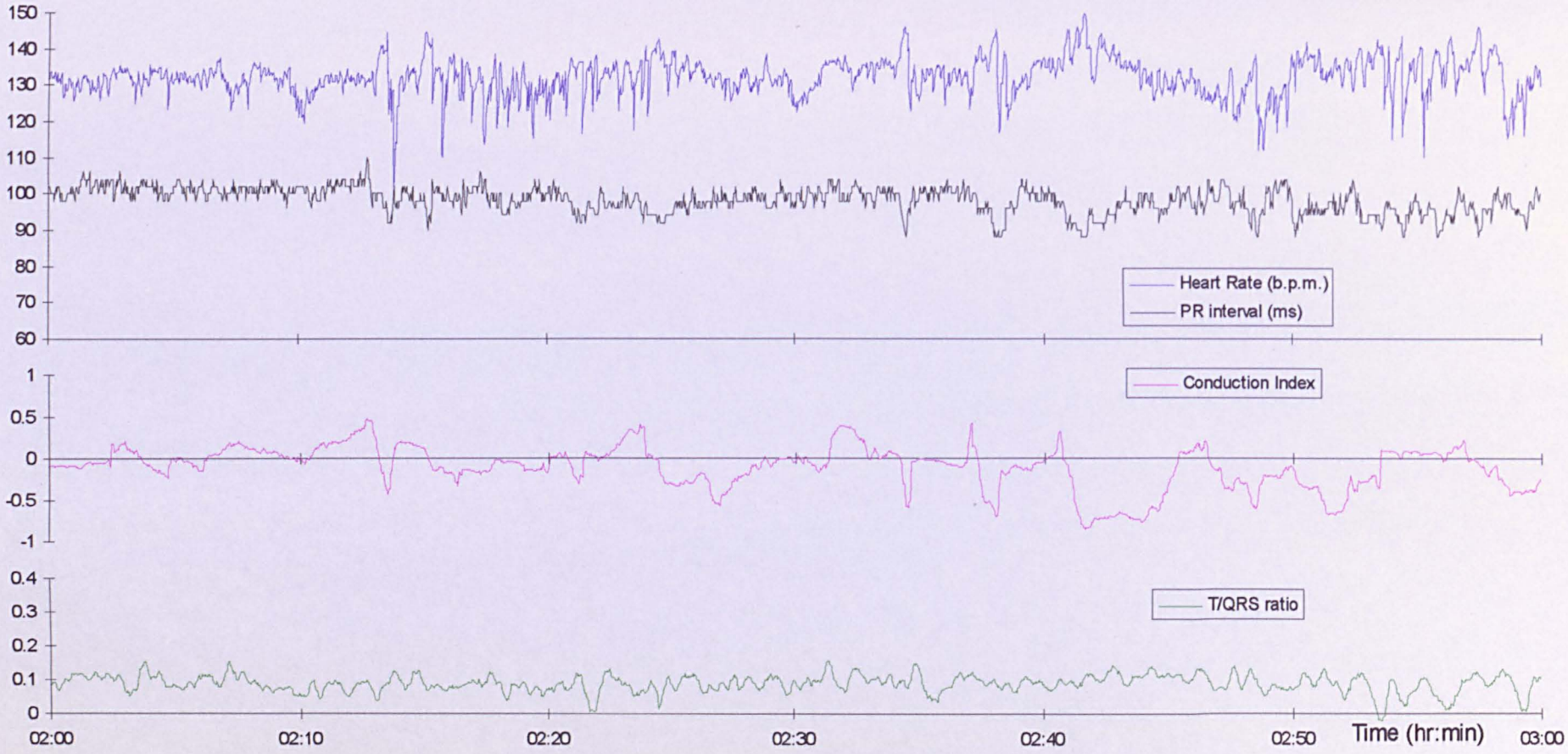


Figure 6.4: Subject 1 (Hour 3)

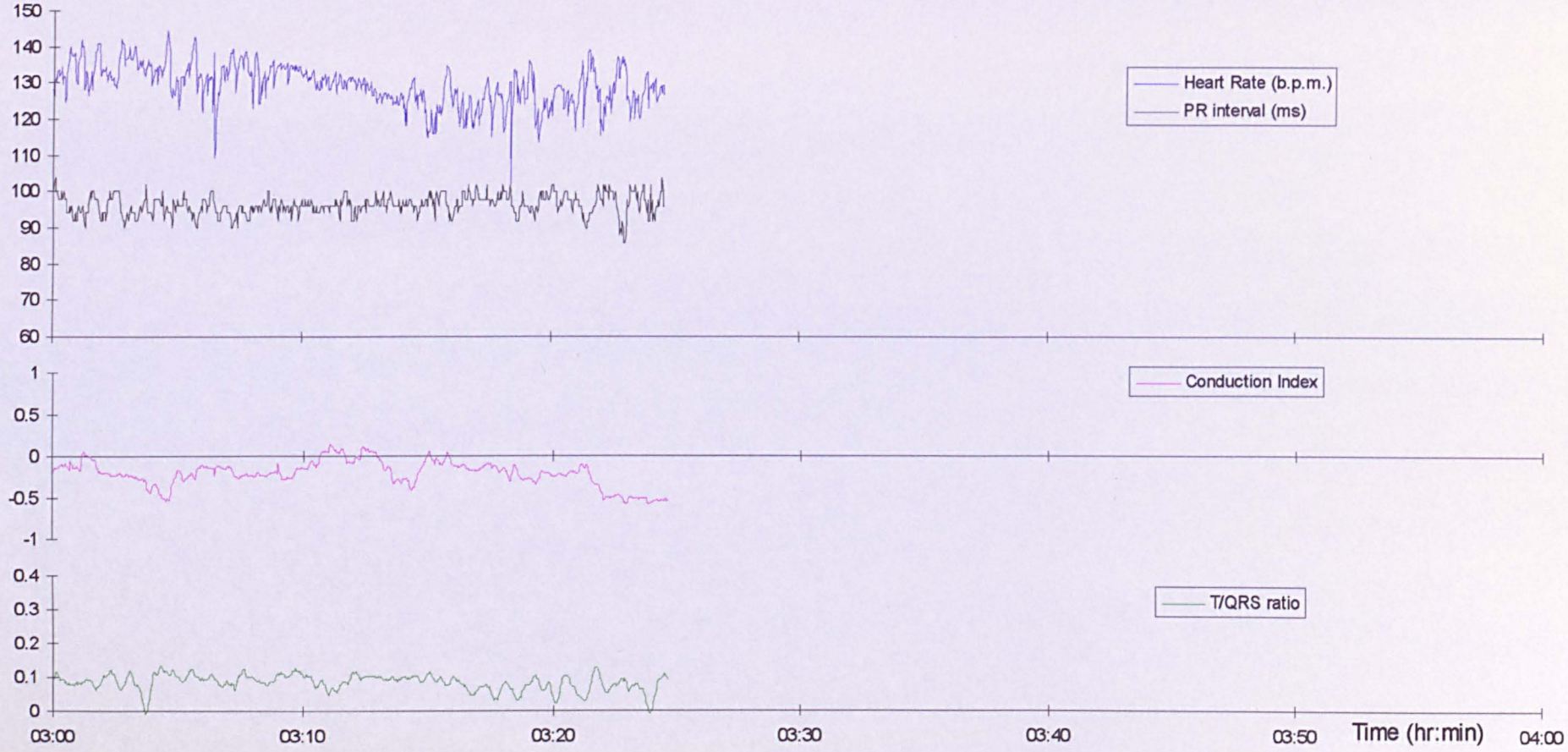


Figure 6.5: Subject 1 (Hour 4)

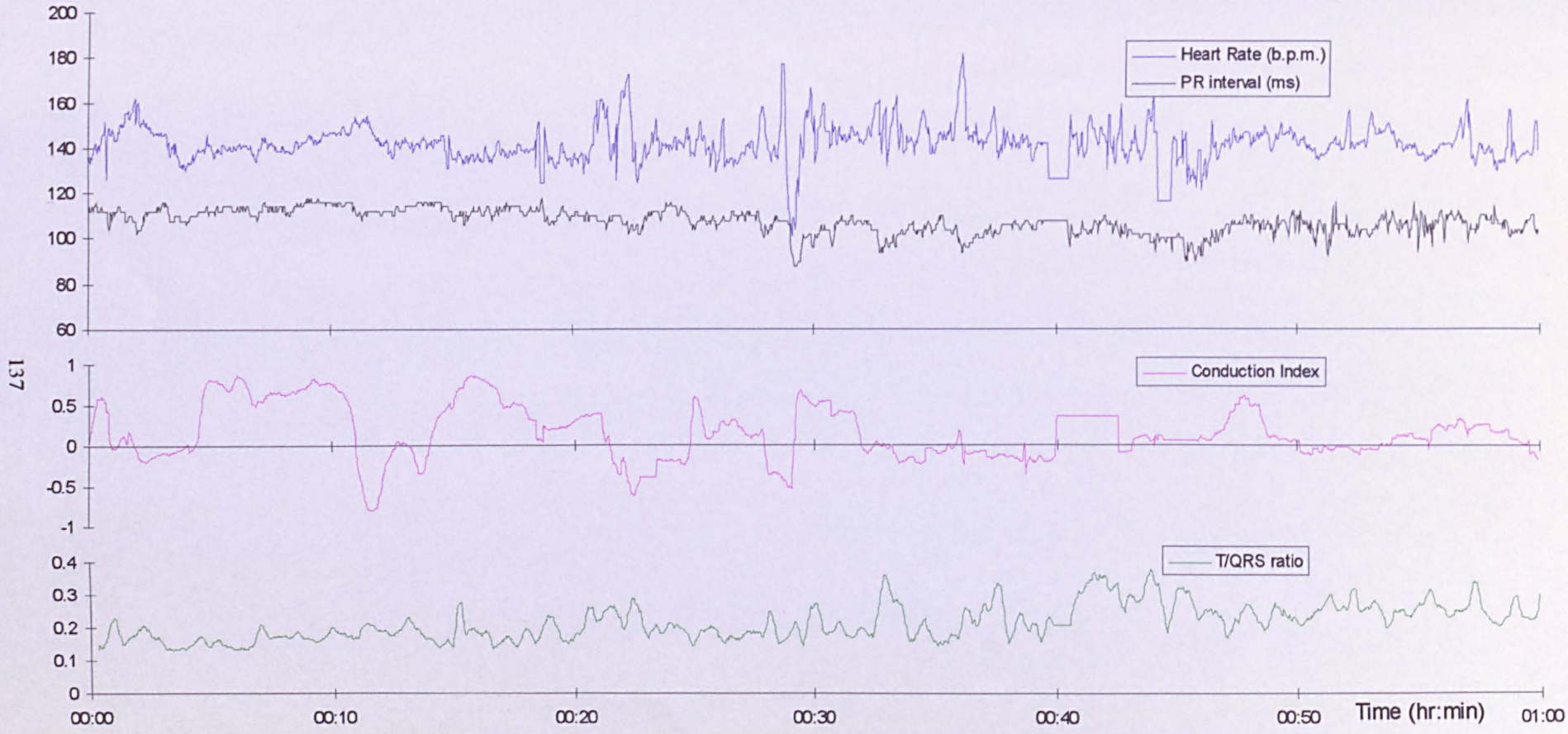


Figure 6.6: Subject 2 (Hour 1)

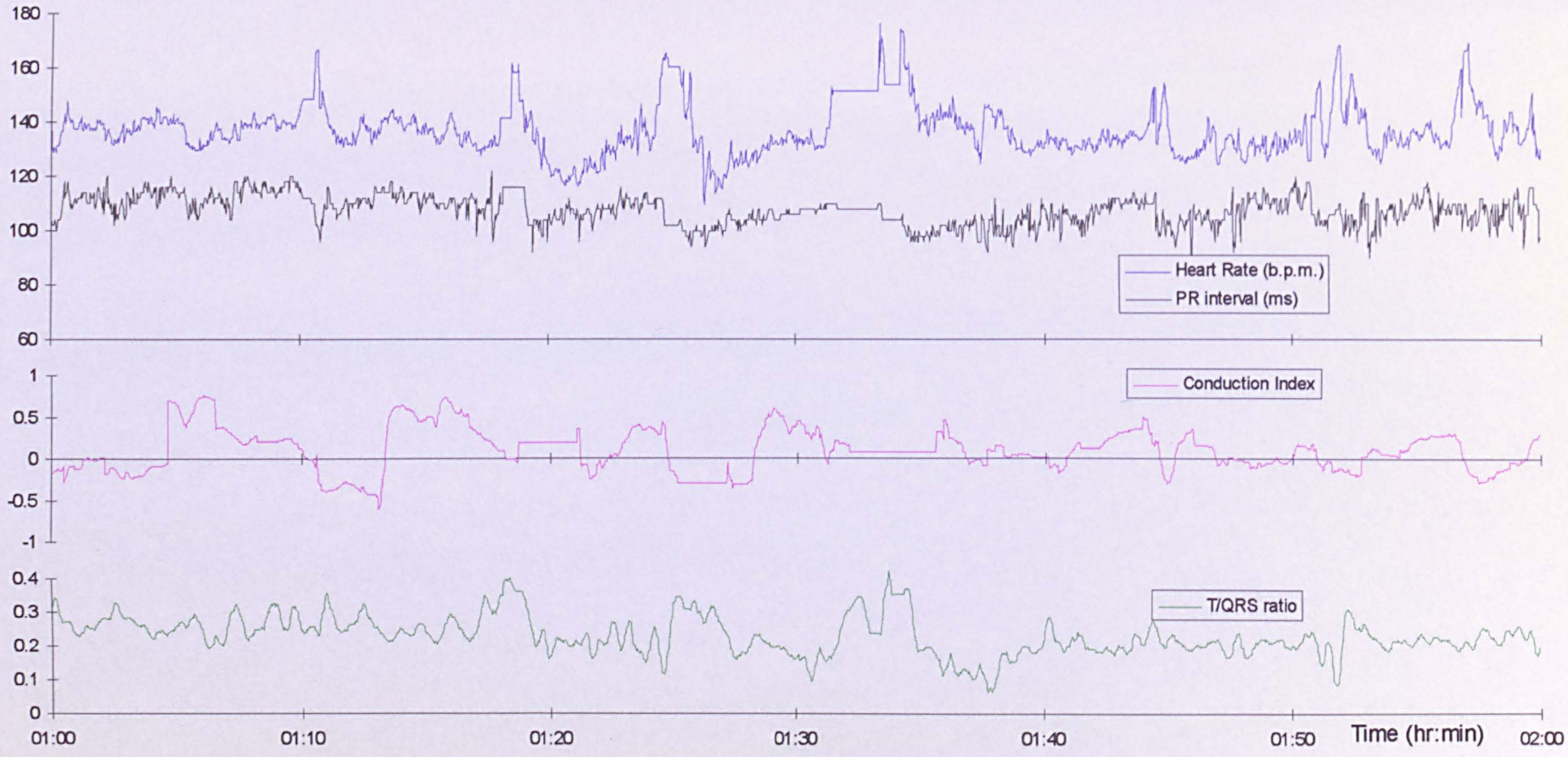


Figure 6.7: Subject 2 (Hour 2)

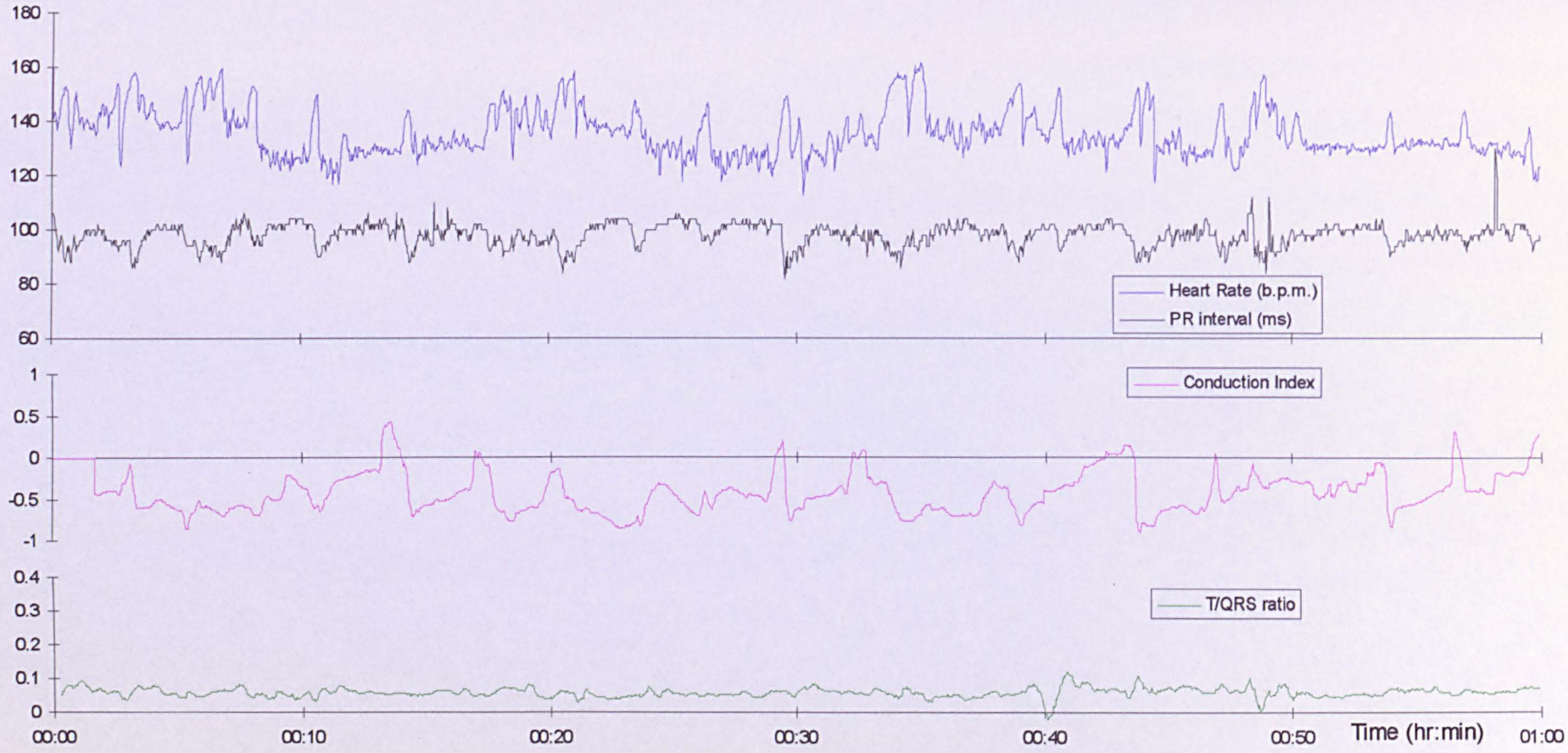


Figure 6.8: Subject 3 (Hour 1)

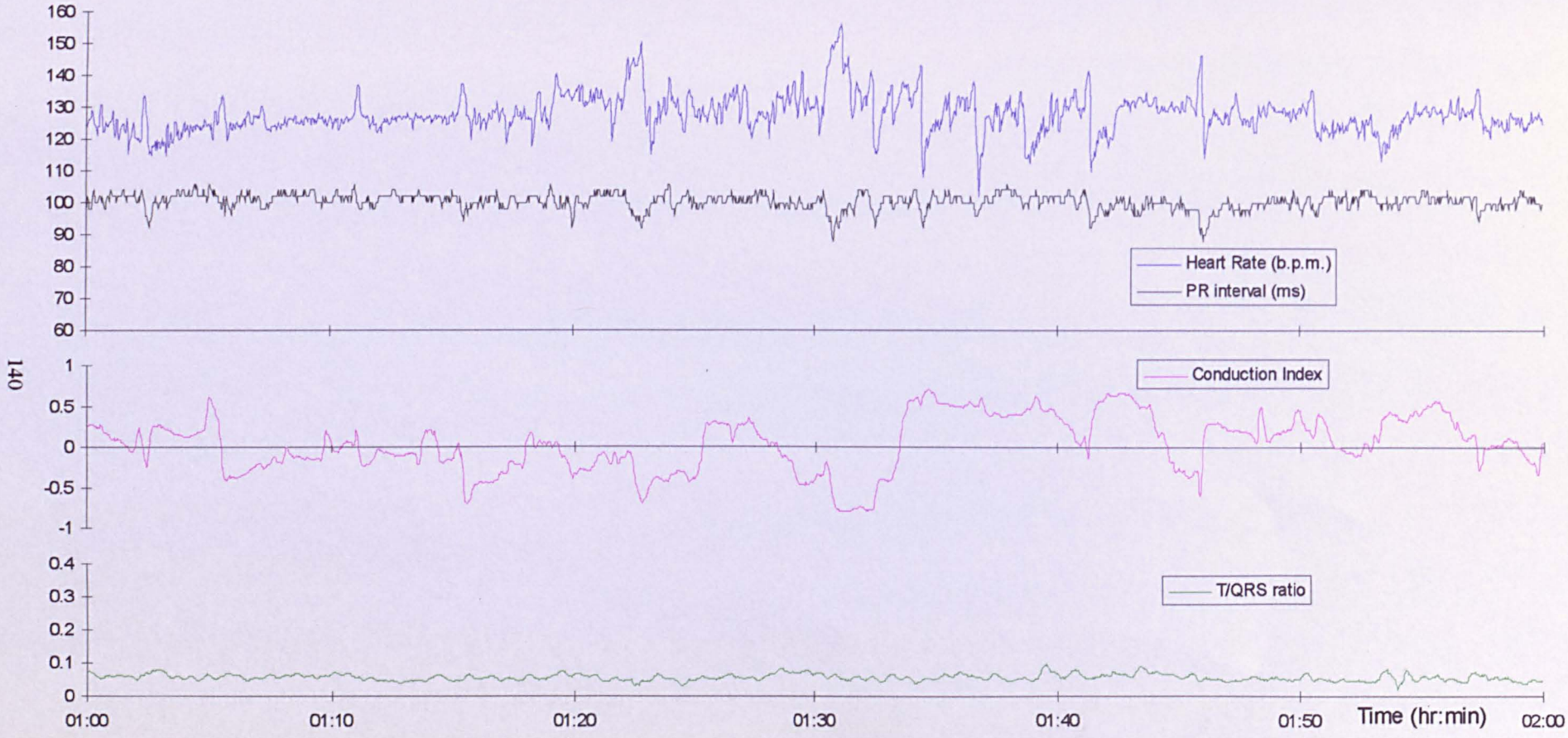


Figure 6.9: Subject 3 (Hour 2)

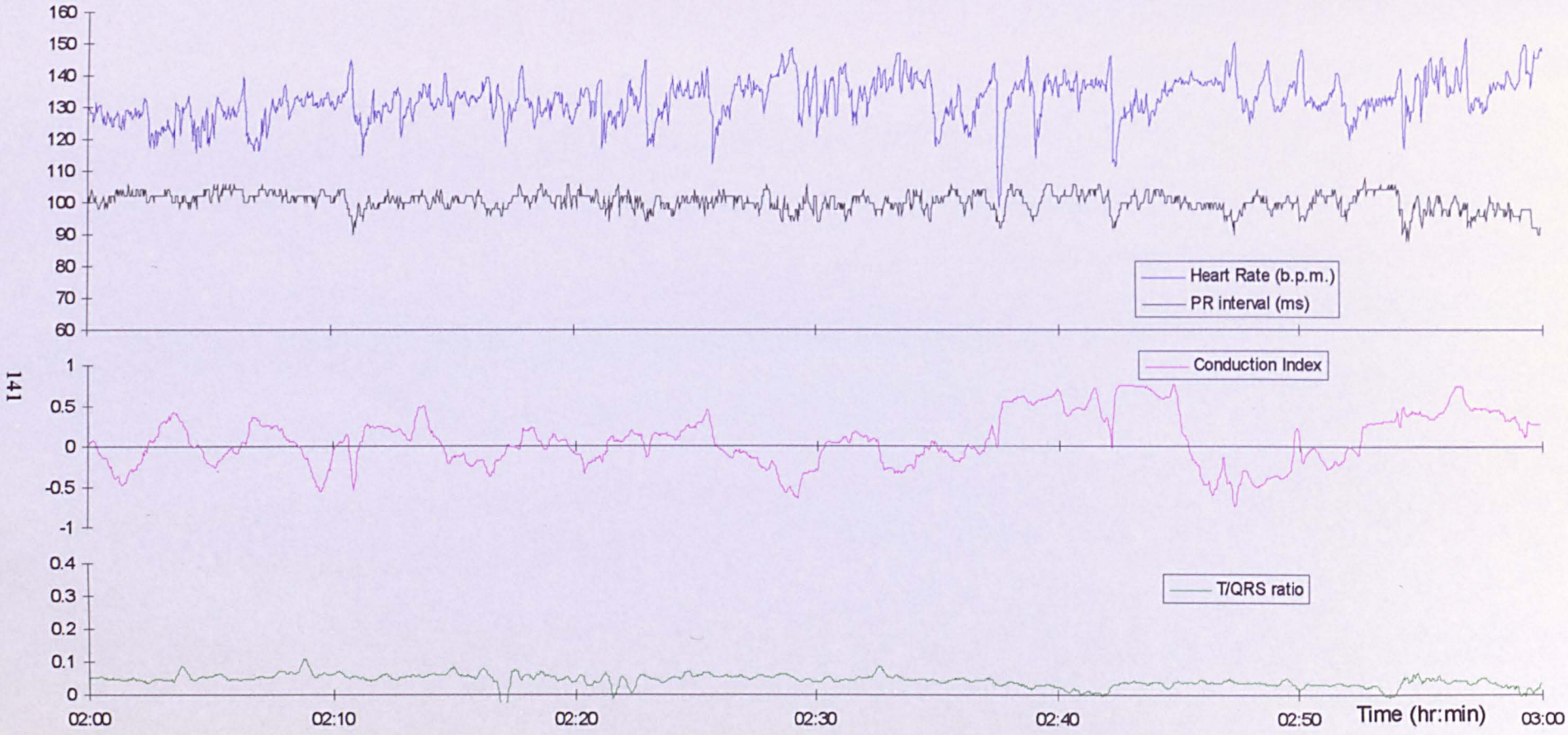


Figure 6.10: Subject 3 (Hour 3)

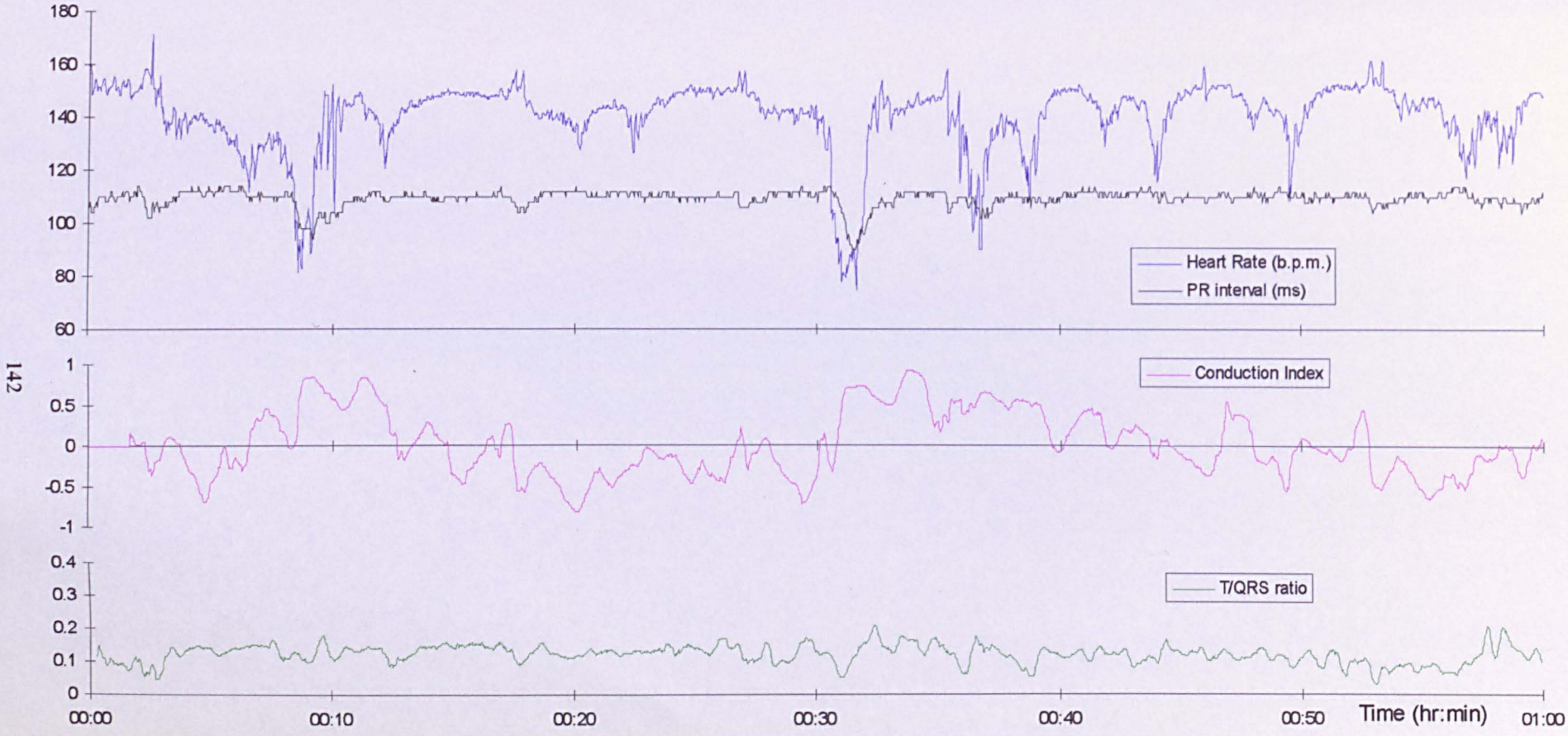


Figure 6.11: Subject 4 (Hour 1)

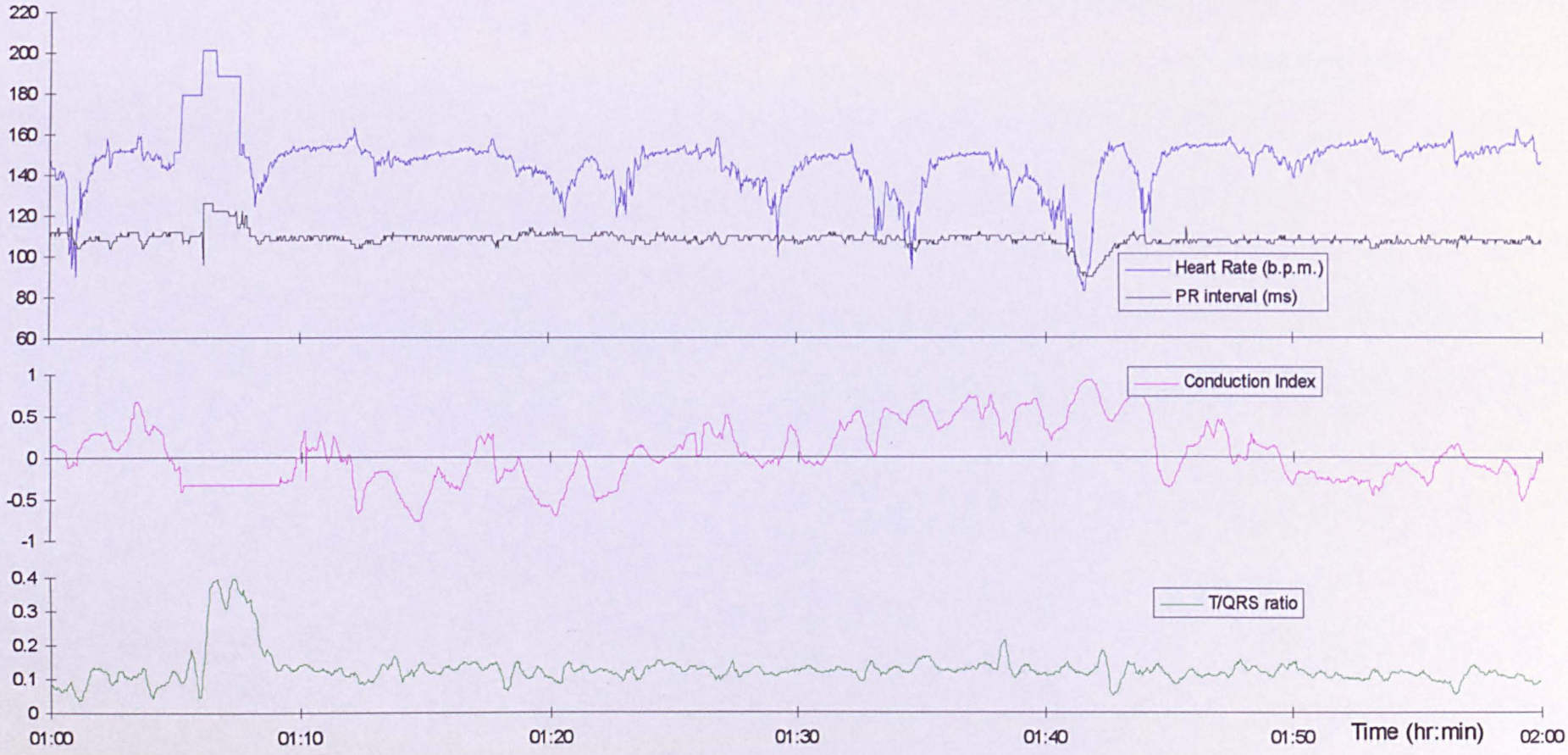


Figure 6.12: Subject 4 (Hour 2)

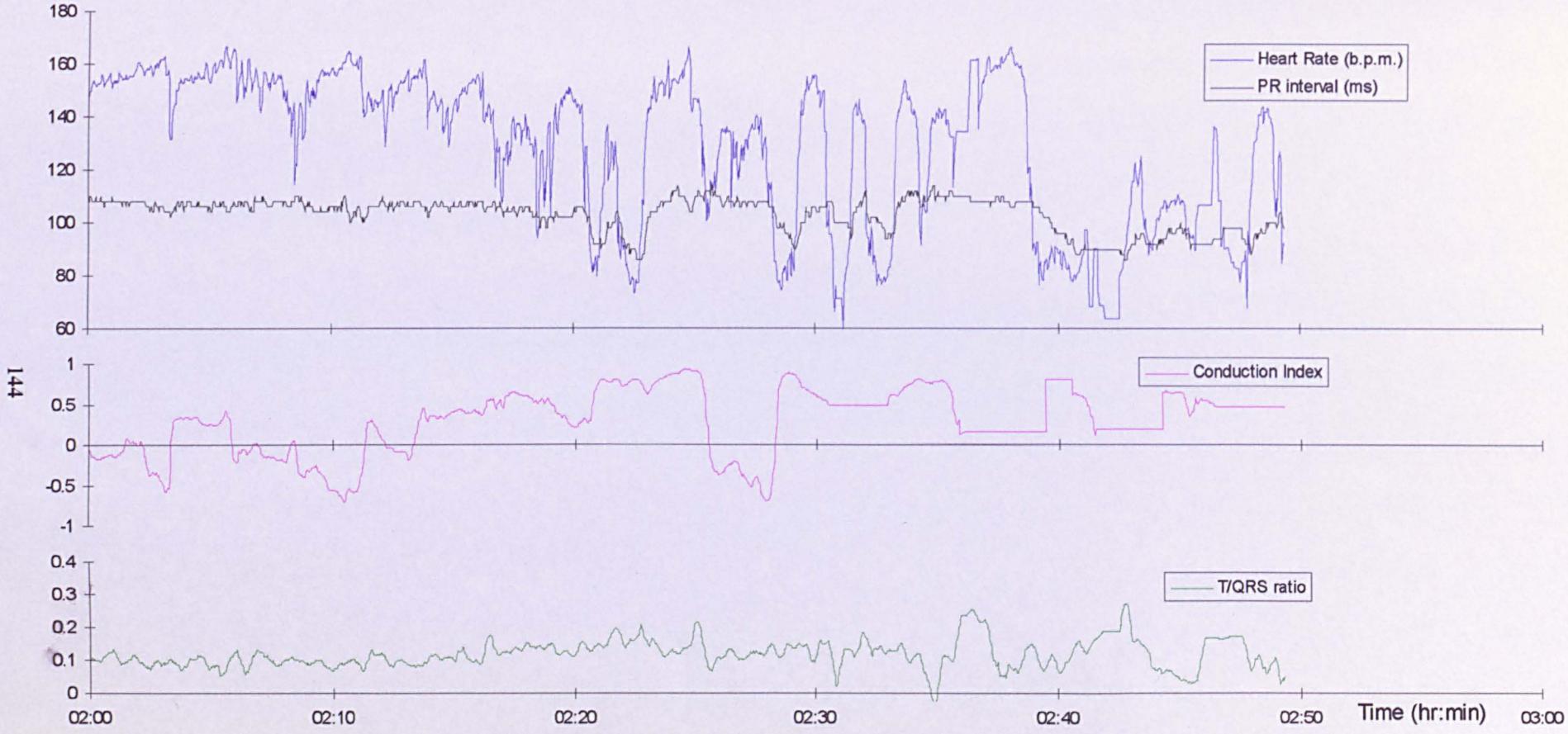


Figure 6.13: Subject 4 (Hour 3)

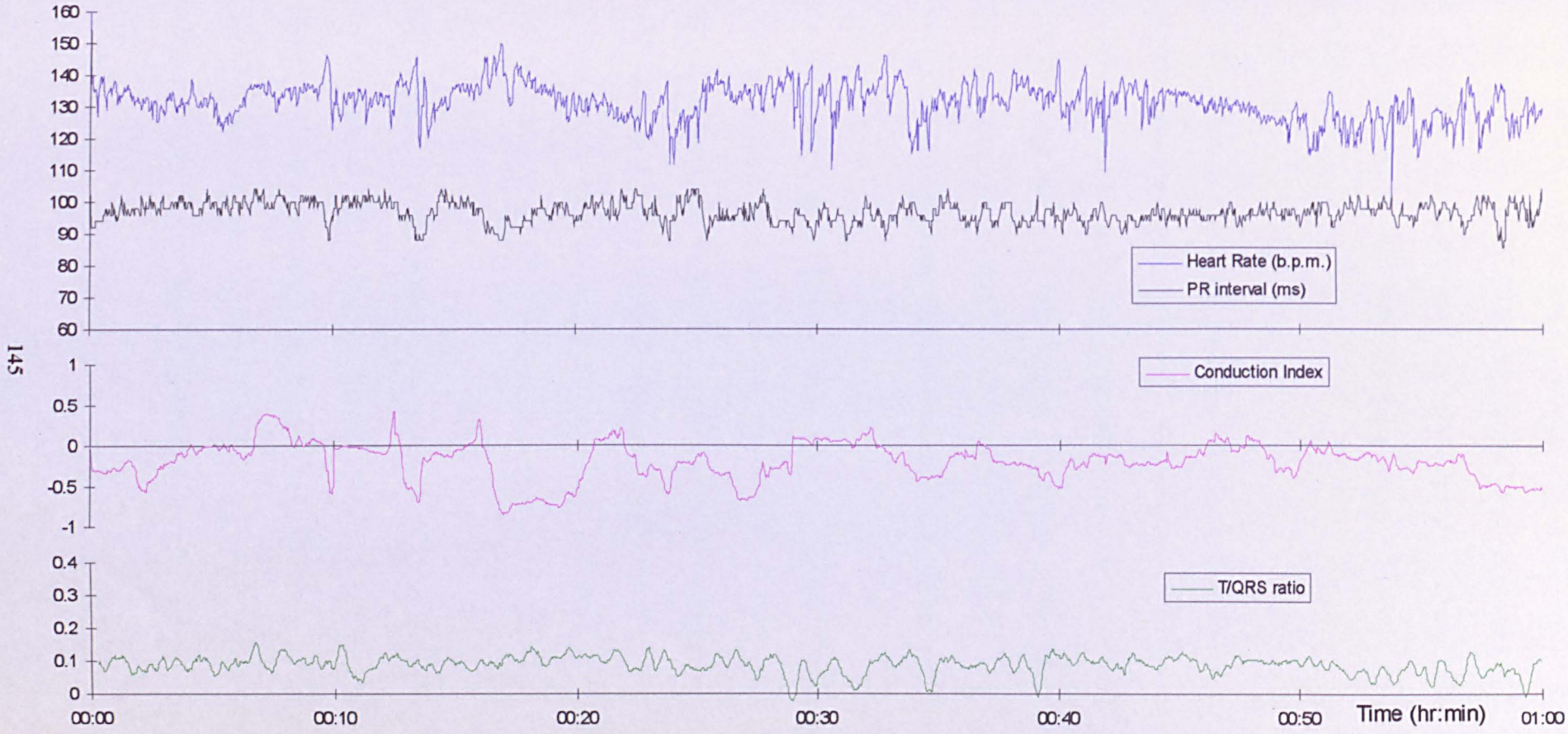


Figure 6.14: Subject 5 (Hour 1)

The CI and T/QRS ratio can now be studied, and, as both parameters have been obtained from the same FECG recording, they can be compared directly with each other. It must be emphasised that a more detailed analysis can only be carried out if many more labours are studied. Nevertheless, the opportunity now exists to compare the parameters.

Within the clinical guidelines for the STAN system, a T/QRS ratio is considered abnormal if it has a value less than -0.05, or greater than 0.24, for more than 20 minutes [Rosen and Luzietti, 1994]. Furthermore, a ratio, which rises more than 0.15, or, in severe case more than 0.40, in 20 minutes is also significant. For the Conduction Index system, a value which remains positive for more than 20 minutes is considered clinically significant [vanWijngaarden et al., 1996].

During an examination of the T/QRS ratio, only subject 2 showed signs of abnormality by the STAN criteria, when the T/QRS is high, exceeding 0.24. Subject 2 is, however not an acidotic case. However, from Chapter 4, it has been demonstrated that the T/QRS ratio has an error as large as 0.05 with 0 dB SNR. Thus, the noise levels must be considered, when observing the T/QRS ratio: within these 5 labours, a SNR of 0 dB is not uncommon.

For all 5 labours, the CI fluctuates between positive and negative values. In Chapter 4, it has been shown that at the noise levels encountered, a *measured* CI of between -0.3 and +0.3 could indeed have a true value that is either positive or negative. For most of the time, all 5 labours have a CI between -0.3 and +0.3, indicating an unknown true sign. Nevertheless, studying the traces for positive

durations in excess of 20 minutes, subject 4 does show signs of acidosis by the CI criteria, and this case is indeed acidotic.

There is no indication that both parameters are correlated: when one parameter indicates fetal distress, the other does not. However, for a more comprehensive comparison, many more labours must be analysed by the system.

It must be remembered that the data used in the 5 labours shown, have been recorded on a system which distorts the signal: the original signal is regenerated using a frequency equaliser, which boosts the lower frequencies. For a fairer comparison, recordings should be made, using the system described in this thesis. It has been shown that this system does not affect the ECG parameters, and offers the additional advantage that the digital data can be stored immediately, without the use of magnetic tape.

6.4 Discussion

This Chapter has described how simulated data was passed through the analogue front-end, and how real data, recorded from the fetal scalp, was analysed by the system. It was clearly shown that plausible plots of heart rate, PR interval, T/QRS ratio, and Conduction Index were obtained. This system would therefore allow these two parameters to be compared.

Using the FECG analysis system described in this thesis, signals from the fetal scalp can be recorded as digitised data for future reference. Undoubtedly, a database of undistorted recordings would be extremely useful when validating other systems.

7. Conclusions and further work

The research within this thesis concerns the monitoring of the fetus during labour, using the fetal electrocardiogram. A versatile FECG analysis system has been developed, implementing algorithms currently used in other FECG analysis systems, in order to allow various FECG parameters to be extracted. Although the algorithms used in this system have been demonstrated by several authors, it was felt they had been used with only minimal testing of them to find their limitations and determine their accuracy in controlled conditions.

Choice of performing the analysis on a PC ensured ease of standardisation (regarding data formats, storage media, and hardware). Furthermore, the system operates via a familiar Microsoft Windows interface, and use of Object Oriented Programming allows new algorithms to be easily implemented at a later stage.

The simulator, described in Chapter 3, was developed to check the ability of the currently used algorithms to extract the measured, and derived, parameters, and assess their accuracy under different noise conditions. This was also developed within the Windows environment for ease-of-use, and its modular design will allow the software to be improved more easily. A number of improvements for future work have been suggested: the parameters could be made more easy to adjust by dragging various points of the morphology with a mouse; test waveforms, such as a sinusoid, could be also generated, and the resolution of the analogue signal could be improved by changing the D/A PC card.

Using the FECG simulator system to generate a noise-free signal, the measured parameters were shown to be accurate as follows: PR intervals between 80 ms and 120 ms, could be calculated to the nearest 2 ms; T/QRS values between -0.40 and +0.60 had a maximum error of 0.008.

Since there will always be some noise in real recordings, coherent-averaging is used to improve the SNR of the ECG waveform and the parameter accuracy. However, there will always be a compromise between the parameter accuracy when the morphology is static, and parameter accuracy when the morphology is changing.

In the case of the Conduction Index, the optimum value of N, in the recursive coherent-average filter, is 10. If too many complexes are averaged, transient changes in the PR interval will be filtered, giving an error in the CI, whatever the noise. For a measured parameter on which clinical decisions are based, it is important to decide upon the time period over which these decisions will be made. From this, the settling time of the parameter, and the number of complexes needed during averaging, can be determined.

In Chapter 4, it has been shown that when the SNR is below -3 dB of mains noise, or 0 dB of white noise, the CI value should not be believed. At noise levels normally encountered, an RMS error of 0.05 would be expected in the T/QRS ratio, and the sign of the CI would be uncertain when its calculated magnitude was less than 0.3.

When data, recorded from the fetal scalp, was analysed by the system, plausible plots of heart rate, PR interval, T/QRS ratio and Conduction Index were obtained. However, the CI was found to lie in the region -0.3 to +0.3 for more that half the time, indicating that the CI value should be ignored. Additionally, as this system extracts both the T/QRS ratio and the CI, the abilities to determine the health of a fetus using these two parameters, can be compared.

As this work has shown degree of errors in both the CI and T/QRS, it is important that the clinician be aware of the system limitations and the accuracy of the data. If the noise in the signal is measured, the accuracy of any parameter can be estimated, as it has been measured in Chapter 4. Therefore, a future improvement to the analysis software must be to add an indication of the accuracy of the data being presented to the user, estimated from the measured noise. When a parameter is shown, the error should be displayed as either an error value, or more simply marked as good, intermediate, or inaccurate. It must be remembered that the clinician may have other information about the fetus: if a system is giving inaccurate information, other more reliable data from a different monitor may be used.

Using the analysis described in this work, signals from the fetal scalp can be recorded as digitised data for future use. Undoubtedly, a database of undistorted recordings would be extremely useful when validating other systems.

This thesis details the framework for evaluating FECG analysis systems. The FECG simulator allowed many parameters of the simulator to be altered, and different types of noise to be added, so that the FECG analyser (presented in Chapter 2), containing the currently implemented algorithms, could be tested. The versatility of the system would allow the algorithms to be tested in even greater depth in the future, or for other systems to be tested.

Having established the limitations of the algorithms, the task of developing new algorithms is made easier. First, the new algorithms can be compared with the present algorithms, to test their effectiveness. Furthermore, the modular method of encoding, together with the high level of software documentation, will allow the analysis software to be adapted quickly, as required, for other analysis methods. It will also allow the extraction of other parameters derived from the ECG. Ideas for improvements to each of the algorithms have been detailed in section 2.4.

There are many views about the correct bandwidth needed in the analogue front-end. Using the IPFM model, with PAM, it has been shown that as long as the front-end passes all frequencies above, and including, the minimum heart beat frequency (~1 Hz) without phase distortion, and does not amplify the frequency components below this frequency, then the RR interval, PR interval, and average T/QRS will be undistorted.

References

VANALSTE J.A., VANECK W. and HERRMANN O.E., *Methods for electrocardiography in patients unable to perform leg exercise: Rowing ergometry, robust averaging and linear phase filtering*, Proceedings of Computers in Cardiology, I.E.E.E. Computer Society, Florence, Italy (1981), pp465-468.

VANALSTE J.A. and SCHILDER T.S., *Removal of base-line wander and power-line interference from the ECG by an efficient FIR filter with a reduced number of taps*, I.E.E.E. Trans. Biomed. Eng. **BME-32**:12 (Dec 1985), pp1052-1060.

VANALSTE J.A., VANECK W., and HERRMANN O.E., *ECG baseline wander reduction using linear phase filters*, Comp. BioMed. Res. **19** (1986), pp417-427.

AMPLICON LIVELINE LTD, Centenary Industrial Estate, Hollingdean Road, Brighton, East Sussex, BN2 4AW.

ARULKUMARAN S., LILJA H., LINDECRANTZ K., RATMAN S.S., THAVARASAH A.S. and ROSEN K.G., *Fetal ECG waveform analysis should improve surveillance in labour*, J. Perin. Med. **18** (1990) pp13-22.

AZEVEDO S. and LONGINI R.L., *Abdominal-lead fetal electrocardiographic R-wave enhancement for heart rate determination*, I.E.E.E. Trans. Biomed. Eng **BME-27**:5 (1980) pp255-260.

BAYLY E.J., *Spectral analysis of Pulse Frequency Modulation in the nervous systems*, IEEE Trans. Biomed. Eng. **BME-15**:4, (1968), pp257-265.

BARRETT J.F.R., JARVIS G.J., MACDONALD H.N., BUCHAN P.C., TYRELL S.N. and LILFORD R.J., *Inconsistencies in clinical decisions in obstetrics*, Lancet **336** (1990), pp549-550.

BETTS J.A., *Signal processing, modulation and noise*, Hodder and Stoughton, Sevenoaks, Kent, UK (1976), pp91.

BLUE CHIP TECHNOLOGY, Hawarden Industrial Park, Manor Lane, Deeside, Clwyd, CH5 3PP, UK.

BORLAND INTERNATIONAL, *ObjectWindows for C++ -User's Guide* (1991).

BROADHEAD T.J. and JAMES D.K., *Worldwide utilization of caesarean section*, Fetal and Maternal Med. Rev., **7** (1995), pp99-108.

CERUTTI M., CIVARDI S., BIANCHI A., SIGNORINI M.G., FERRAZZI E., and PARDI G., *Spectral analysis of antepartum heart rate variablilty*, Clin. Phys. Physiol. Meas. **10, Suppl. B** (1989) pp27-31..

CHESTER E.G., SHARIF B.S., FURNISS S.S., and CAMPBELL R.W.F., *Classification of fragmented cardiac myopotentials for a fragmentation mapping system*, Colloquium on "Signal Processing in Cardiography", pp(4/1)-(4/5).

CHUNG T.K.H., MOHAJER M.P., YANG Z.J., CHANG A.M.Z. and SAHOTA D.S., *The prediction of fetal acidosis at birth by computerised analysis of intrapartum cardiotocography*, Br. J. Obst. Gynae. **102** (1995), pp454-460.

CINVENTA AB., Karantansgat, S-442 35 Kungalv, Sweden.

COCKBURN J., *ST-waveform analysis of the fetal electrocardiogram could reduce fetal blood sampling*, Br. J. Obst. Gynae. **99** (1992), pp783.

CONSENSUS IN MEDICINE, *Caesarean childbirth*, BMJ **282** (1981), pp1600-1604.

CREMER M., *Ueber die direkte albeitung der aktionströmme des mensclichen herzens von oesphagus and uber die electrokardiogramm des fötus*, Munch. med. Wschr. **53**, pp811 (1906).

DEANS A., GENEVIER E. and STEER P., *A comparative study of the FECG signal obtained from simultaneously applied single spiral and Copeland scalp electrodes*, 1996.

DEBOER R.W., KAREMAKER J.M., and STRACKEE J., *Comparing spectra of a series of point events particularly for heart rate variability data*, IEEE Trans.Biom. Eng., **BME-31**:4, pp384-387.

EVANS A.L., SMITH D.C., and WATTS M.P., *Microprocessor-controlled signal generator for the functional testing of electrocardiographs*, Medical and Biological Engineering & Computing (Sept 1984), pp468-470.

FAMILY J.M., *Cross-correlation techniques applied to the fetal electrocardiogram*, Ph.D. Thesis, Nottingham University (1982).

FISHER L.E., *The choice of fetal scalp electrodes for fetal electrocardiography analysis*, M.MedSci. Thesis, University of Birmingham (1993).

FRANCHI D., PALAGI G. and BEDINI R., *A PC-based generator of surface ECG potentials for computer electrocardiograph testing*, Computers and Biomedical Research **27** (1994), pp68-80.

GIBSON N.M., *A study of methods applicable to the analysis of fetal heart rate variability*, Ph.D. Thesis, Nottingham University (1995).

GIBSON N.M., WOOLFSON M.S., and CROWE J.A., *Detection of fetal electrocardiogram signals using matched filters with adaptive normalisation*, Med. Biol. Eng. Comput., **35** (1997), pp216-222.

GOOVAERTS H.G. and ROMPELMAN O., *Coherent averaging technique: A tutorial review*, J. Biomed. Eng. **13** (Jul 1991), pp275-280.

GRANT A., O'BRIEN N., JOY M.-T., HENNESSY E., and MACDONALD D., *Cerebral palsy among children born during the Dublin trial of intrapartum monitoring*, Lancet (1989), pp1233-1235.

GREANLEAF SOFTWARE INC., 16479 Dallas Parkway Suite 100, Dallas, TX 75248, USA.

GREENE K.R., DAWES G.S., LILJA H., and ROSEN K.G., *Changes in the ST waveform of the fetal lamb electrocardiogram with hypoxemia*, Am. J. Obstet. Gynecol **144** (1982), pp950-958.

GREENE K.R. and ROSEN K.G., *Long-term ST waveform changes in the ovine fetal electrocardiogram: the relationship to spontaneous labour and intrauterine death*, Clin. Phys. Physiol. Meas. **10** (1989), Suppl. B, pp33-40.

GUYTON A.C., *Textbook of medical physiology - eighth edition* (1991), pp97-123.

HON E.H. and LEE S.T., *The fetal electrocardiogram - II. The electrocardiogram of the dying fetus*, Am. J. Obst. & Gynec. **87**:6 (1963a), pp804-813.

HON E.H. and LEE S.T., *Noise reduction in fetal electrocardiography - II. Averaging techniques*, Am. J. Obst. & Gynec. **87**:8 (1963b), pp1086-1096.

HON E.H. and LEE S.T., *Averaging techniques in fetal electrocardiography*, Med. Electron. Biol. Engng 2 (1964), pp71-76.

HUANG X.B., CROWE J.A., HERBERT J.M., and WOOLFSON M.S., *A Windows application for real-time fetal ECG analysis*, Comp. Biomed. Res. **27** (1994), pp419-433.

HUNTER C.A., LANSFORD K.G., KNOEBEL S.B., and BRAUNLIN R.J., *A technic for recording fetal ECG during labour and delivery*, Obst. Gynec., **16** (1960), pp467-570.

HYNDMAN B.W. and MOHN R.K., *A model of the cardiac pacemaker and its use in decoding the information content of cardiac intervals*, Automedica (1975), pp239-252.

HYPERCEPTION, 9550 Skillman LB125, Dallas, TX 75243, USA.

JENKINS H.M.L., *Technical progress in fetal electrocardiography - a review*, J. Perin. Med. **14** (1986) pp365-370.

JOHANSON R.B., RICE C., SHOKR A., DOYLE M., CHENOY R., and O'BRIEN P.M.S., *ST-waveform analysis of the fetal electrocardiogram could reduce fetal blood sampling*, Br. J. Obst. Gynae. **99** (1992), pp167-168.

KAHN K.A. and SIMONSON E., *Changes of mean spatial QRS and T vectors and of conventional electrocardiographic items in hard anaerobic work*, Circulation **5** (1957), pp629-633.

KEITH R.D.F., BECKLEY S., GARIBALDI J.M., WESTGATE J.A., IFEACHOR E.C., and GREENE K.R., *A multicentre comparitive study of 17 experts and an intelligent system for managing labour using the cardiotocogram*, Br. J. Obst. Gynae. **102** (1995), pp688-700.

KIRK D.L. and SMITH P.R., *Techniques for the routine on-line processing of the fetal eletrocardiogram*, J. Perin. Med. **14** (1986) pp391-397.

KOSSMAN C.E., BRODY D.A., BURCH G.E., HECHT H.H., JOHNSTON F.D., KAY C., LEPESCHKIN E., PIPBERGER H.V., BAULE G., BERSON A.S., BRILLER S.A., GESELOWITZ D.B., HORAN L.G., and SCHMITT O.H., *Report of Committee on electrocadiography, American Heart Association - Recommendations for standardization of leads and of specifications for instruments in electrocadiography and vectorcadiography*, Circulation **35** (March 1967), pp582-602.

LILJA H., GREENE K.R., KARLSSON K., and ROSEN K.G., *ST wavefrom changes of the fetal electrocardiogram during labour - a clinical study*, Br. J. Obstet. Gynaecol. **92** (1985), pp611-617.

LILJA H., KARLSSON K., LINDECRANTZ K. and ROSEN K.G., *Microprocessor based waveform analysis of the fetal electrocardiogram during labour*, Int. J. Obstet. Gynaecol. **5** (1989), pp109-116.

LUI S.M., *A Conduction Index monitor*, Thesis, Nottingham University (1989) pp69-71.

MARINO A.R., *Electrocadiograph simulator*, Electronics World and Wireless World (Oct. 1993) pp852.

MARVELL C.J., *The normal condition in fetal electrocardiography*, Thesis, Nottingham University (1979) pp63-66.

MARVELL C.J., and KIRK D.L., *A simple software routine for the reproducible processing of the electrocardiogram*, J. Biomed. Engng. **2** (1980), pp216-220.

MARVELL C.J., KIRK D.L., JENKINS H.M.L., and SYMONDS E.M., *The normal condition of the fetal electrocardiogram during labour*, Br. J. Obstet. Gynaecol. **87** (1980), pp786-796.

McCORD J.W., *Developing Windows applications with Borland C++3*, SAMS, Indiana, 1992.

METRON AS, Thonning Owesens gt. 35 C, N-7044 Trondheim, Norway.

MIYAHARA H., AKIRI D., and TOSHIRO S., *The reproducibility of interpretation of 10 computer ECG systems by means of a microprocessor-based ECG signal generator*, Computers and BioMedical Research **17** (1984), 311-325.

MOHAJER M.P., SAHOTA D.S., REED N.N., CHANG A., SYMONDS E.M. and JAMES D.K., *Cumulative changes in the fetal electrocardiogram and biochemical indices of fetal hypoxia*, European J. Obst. Gynec. **55** (1994), pp63-70.

MOHAJER M.P., *The fetal electrocardiogram and intrapartum asphyxia*, M.D. thesis, University of Nottingham (1994), pp84-87.

MURRAY H.G., *The fetal electrocardiogram: Current developments in Nottingham*, J. Perin. Med **14** (1986), pp399-404.

MURRAY H.G., *Evaluation of the fetal electrocardiogram (ECG)*, M.D. thesis, University of Nottingham (1992).

MURPHY K.W., JOHNSON P., MOORCRAFT J., PATTINSON R., RUSSELL V., and TURNBULL A., *Birth asphyxia and the intrapartum cardiotocograms*, Br. J. Obstet. Gynaec. **97** (1990), pp470-479.

NATIONAL ADVISORY BODY, *Confidential enquiry into stillbirths and deaths in infancy: Annual report for 1993. Part 1: Summary of methods and main results*, Department of Health (1995), pp11-32.

NEWBOLD S., WHEELER T., CLEWLOW F., and FRANCES S., *Variation in the T/QRS ratio of fetal electrocardiograms during labour in normal subjects*, Br. J. Obstet. Gynaec. **96** (1989), pp144-150.

NEWBOLD S., WHEELER T., and CLEWLOW F., *Comparison of the T/QRS ratio of the fetal electrocardiogram and the fetal heart rate during labour and the relation of these variables to condition at delivery*, Br. J. Obstet. Gynaec. **98** (1991), pp173-178.

NEWBOLD S., WHEELER T., and CLEWLOW F., *The effect of uterine contractions on the T/QRS ratio of the fetal electrocardiogram* , J. Perin. Med. **23** (1995), pp459-466.

NOVAK P. and NOVAK V., *Time/frequency mapping of the heart rate, blood pressure and respiratory signals*, Med. & Biol. Eng. & Comput. **31** (1993), pp103-110.

OUTRAM N.J., IFEACHOR E.C., VANEETVELT P.W.J., and CURNROW J.S.H., *Techniques for optimal enhancement and feature extraction of fetal electrocardiogram*, I.E.E. Proc.-Sci. Meas. Technol. **142**:6 (Nov 1995), pp482-489.

OXFORD INSTRUMENTS, *Meridian 800 Cardiotocograph - Service Manual*, 1992.

OXFORD INSTRUMENTS plc., Medical Systems Division, 1 Kimber Road, Abingdon, Oxon, OX14 1BZ, UK.

PARDI G., TUCCI E., UDERZO A., and ZANINI D., *Fetal electrocardiogram changes in relation to fetal heart rate patterns during labor*, Am. J. Obstet. Gynecol., (Jan 1974), pp243-250.

PARK Y.C., LEE K.Y., YOUN D.H., KIM N.H., KIM W.K. and PARK S.H., *Communications on detecting the presence of fetal R-Wave using the moving averaged magnitude difference algorithm*, I.E.E.E. Trans. Biomed. Eng. 39:8 (1992).

PEASGOOD W., *Enhancement of the abdominal fetal electrocardiogram*, Thesis, Nottingham University (1993) pp145-150.

PIPBERGER H.V., ARZBAECHER R.C., BERSON A.S., BRILLER S.A., BRODY D.A., FLOWERS N.C., GESELOWITZ D.B., LEPESCHKIN E., OLIVER G.C., SCHMITT O.H., and SPACH M., *Report of Committee on electrocadiography, American Heart Association - Recommendations for standardization of leads and of specifications for instruments in electrocadiography and vectorcadiography*, Circulation 52 (1975), 'AHA Page' 11-31.

PRESS W.H., TEUKOLSKY S.A., VETTERLING W.T., FLANNERY B.P., *Numerical recipes in C - the art of scientific computing* (Second edition), Cambridge University Press (1992), pp105-110, 288-290.

REED N.N., MOHAJER M.P., SAHOTA D.S., JAMES D.K., and SYMONDS E.M., *The potential impact of PR interval analysis of the fetal electrocardiogram (FECG) on intrapartum fetal monitoring*, Eur. J. Obst. & Gynec. & Repr. Biolog. **68** (1996), pp87-92.

RHYNE V.T., *Digital signal enhancement of the fetal electrocardiogram*, Am. J. Obst. & Gynec. **102**:4 (1968), pp549-555.

ROETZHEIM W., *PC Magazine - Programming Windows with Borland C++*, Ziff-Davis Press, Emeryville, California (1992).

ROMPELMAN O. and ROS H.H., *Coherent averaging technique: A tutorial review*, J. Biomed. Eng 8 (Jan 86), pp24-35.

ROSEN K.G., *Alterations in the fetal electrocardiogram as a sign of fetal asphyxia - experimental data with a clinical implementation*, J. Perin. Med **14** (1986), pp355-363.

ROSEN K.G. and LINDECRANTZ K., *STAN - the Gothenburg model for fetal surveillance during labour by ST analysis of the fetal electrocardiogram*, Clin. Phys. Physiol. Meas., **10** (1989), pp51-56.

ROSEN K.G., and LUZIETTI R., *The fetal electrocardiogram: ST waveform analysis during labour*, J. Perin. Med. **22** (1994), pp501-512.

SAHOTA D., Private correspondence (1997).

SANDIGE R.S., FERRIS C.D., and BHASKARAN A., *Electronic ECG simulator*, Biomedical Sciences Instrumentation **28** (1992), pp21-25.

SAYERS B.McA., *Fetal monitoring using FECG morphology: the Nottingham project*, Private correspondance (1997).

SCHWID H.A., *Electrocardiogram simulation using a Personal Computer*, Computers and Biomedical Research **21** (1988), pp562-569.

SCOTT D.E., *Comparison of coherent averaging techniques for repetitive biological signals*, Med. Res. Eng. 9(1970), pp7-8

SHIELD J.E.A., *Surveillance for asphyxia with the fetal electrocardiogram*, Ph.D. thesis, University of Nottingham (1977), pp54-61.

SHIELD J.E.A. and KIRK D.L., *The use of digital filters in enhancing the fetal electrocardiogram*, J. Biomed. Engng (1981), pp44-48.

SKILLERN L., COCKBURN J., BENJAMIN M., PEARCE J.M., SAHOTA D., REED N., MOHAJER M., JAMES D., and SYMONDS M., *A comparative study of the fetal electrocadiogram recorded by the STAN and Nottingham systems*, Br. J. Obst. Gynaec. **101** (1994), pp582-586.

SMITH P.R., *An advanced fetal monitoring system*, Ph.D. Thesis, University of Nottingham (1983).

STRACKEE J. and PEPER A., *Effect of mains interference on time coherent averaging*, Med. & Biol. Eng. & Comput. **30** (1992), pp491-494.

STURBOIS X., TOURNAIRE M., RIPOCHE A., LeHOUZEC R., BREART G., CHAVINIE J., and SUREAU C., *Evaluation of the fetal state by automatic analysis of the heart rate*, J. Perin. Med. **1** (1973), pp235-244.

SUREAU C., *The history of fetal surveillance*, in: VANGEIJN H.P. and COPRAY F.J.A. (eds): Critical Appraisal of Fetal Surveillance, Elsevier Science B.V., Amsterdam (1994), pp3-10.

SYKES G.S., MOLLOY P.M., JOHNSON P., STIRRAT G.M., and TURNBULL A.C., *Fetal distress and the condition of newborn infants*, BMJ **287** (1983), pp943-945.

SYMONDS E.M., *Fetal monitoring: Medical and legal implications for the practitioner*, Current Opinion in Obst. & Gynec. 6(1994a), pp430-433.

SYMONDS E.M., *The P wave and the PR interval*, in: VANGEIJN H.P. and COPRAY F.J.A. (eds): A Critical Appraisal of Fetal Surveillance, Elsevier Science B.V., Amsterdam (1994b), pp381-387.

TENVOORDE B.J., FAES Th.J.C., and ROMPLEMAN O., *Spectra of data sampled at frequency modulated rates in application to cardiovascular signals*, Med. Biol. Eng. Comp. **32** (1994), 63-76.

TEPPNER U., LOBODZINSKI S.M., NEUBERT D., and LAKS M.M., *A high-fidelity multi-pathologic electronic ECG test patient for performance evaluation of computerized ECG analysis systems*, Proceedings of Computers in Cardiology, I.E.E.E. Computer Society, Los Alamitos, CA, U.S.A.(1988), pp247-250.

THALER I., TIMOR I.E., and GOLDBERG I., *Interpretation of the fetal ECG during labor: the effect of uterine contractions*, J. Perin. Med. **16**:4 (1988), pp373-379.

TREFFERS P.E. and PEL M., *The rising trend for caesarean birth*, BMJ. **307** (1993), pp1017-1018.

VOLTECH INSTRUMENTS LTD., 65 Milton Park, Abingdon, Oxfordshire, OX14 4RX.

WESTGATE J., KEITH R.D.F., CURNOW J.S.H., IFEACHOR E.C., and GREENE K.R., *Suitability of fetal scalp electrodes for monitoring the fetal electro cardiotocogram during labour*, Clin. Phys. Physio. Meas. **11**:4 (1990), pp297-306.

WESTGATE J., HARRIS M., CURNOW J.S.H., and GREENE K.R., *Randomized trial of cardiotocography alone or with ST waveform analysis for intrapartum monitoring*, Lancet (1992), pp194-198.

WESTGATE J., HARRIS M., CURNOW J.S.H., and GREENE K.R., *Plymouth randomized trial of cardiotocogram only versus ST waveform plus cardiotocogram for intrapartum monitoring in 2400 cases*, Am. J. Obstet. Gynecol. (1993), pp1151-1160.

VANWIJNGGAARDEN W.J., SAHOTA D.S., JAMES D.K., FARRELL T., MIRES G.J., WILCOX M., and CHANG A., *Improved intrapartum surveillance with PR interval analysis of the fetal electrocardiogram (FECG): A randomised trial showing a reduction in fetal blood sampling*, Am. J. Obstet. Gynecol. **174:4** (1996), pp1295-1299.

WOOLFSON M.S., HUANG X.B., and CROWE J.A., *Time-varying wiener filtering of the fetal ECG using the wavelet transform*, Colloquium on "Signal Processing in Cardiography", pp(11/1)-(11/5) (1995).

Appendix I : Analysis user guide

When the program starts, the user is presented with the title screen (*see* Figure I.1). After pressing a key, or pressing the mouse, the main menu is displayed (*see* Figure I.2). The 4 buttons allow easy access to the different sections of the program.

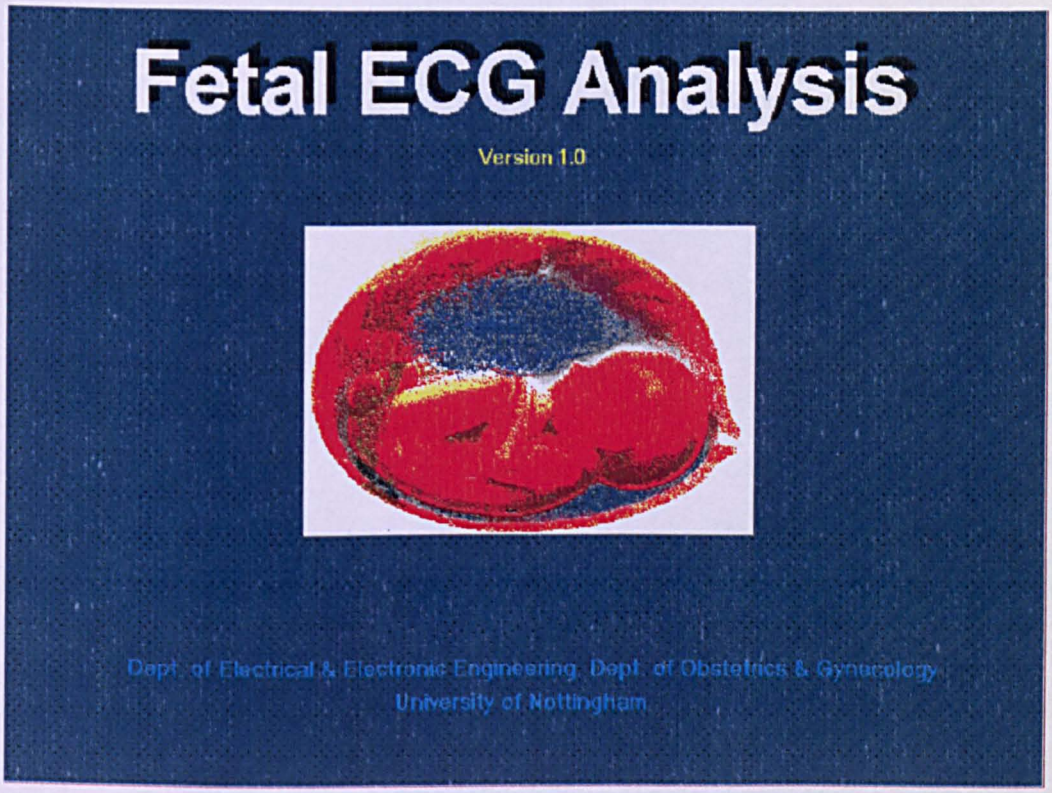


Figure I.1: Title screen

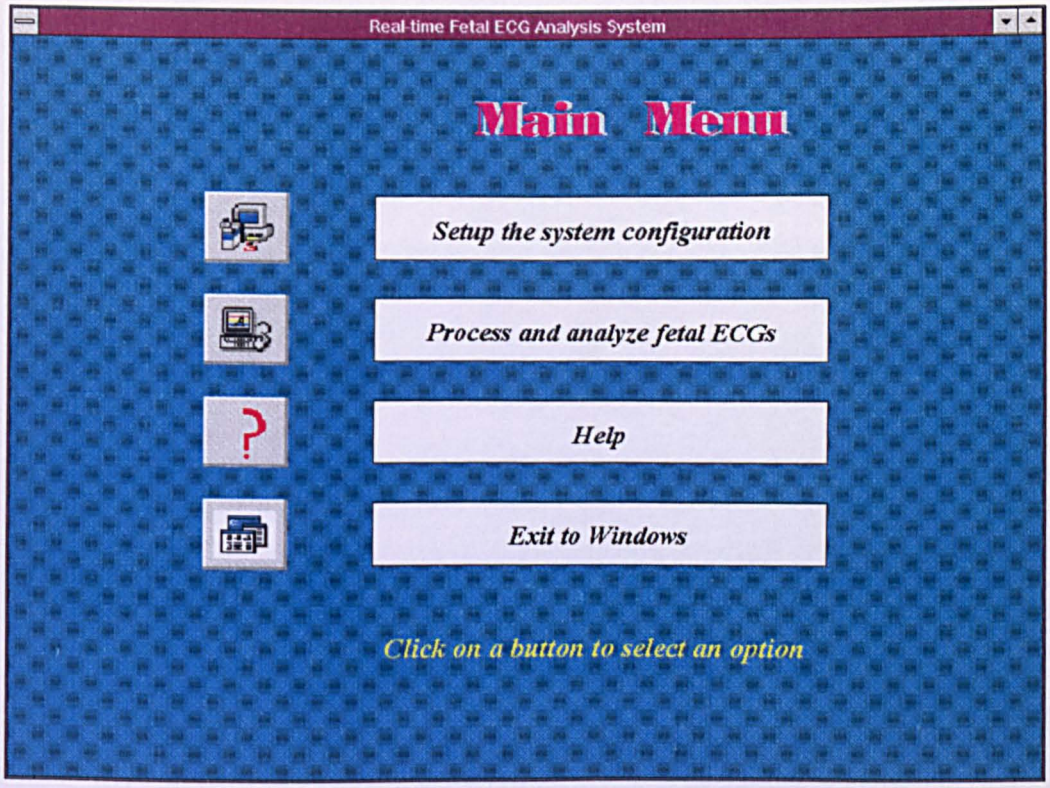


Figure I.2: Main menu

Each of these sections will now be discussed.

I.1 Setup the system configuration

From this screen, one can:

1. Enter patient details (*see* Figure I.3).
2. Choose processing options from the menu. This is limited to choosing either the Meridian or a specified file for the data source.
3. Display any raw ECG data, read in from a file.

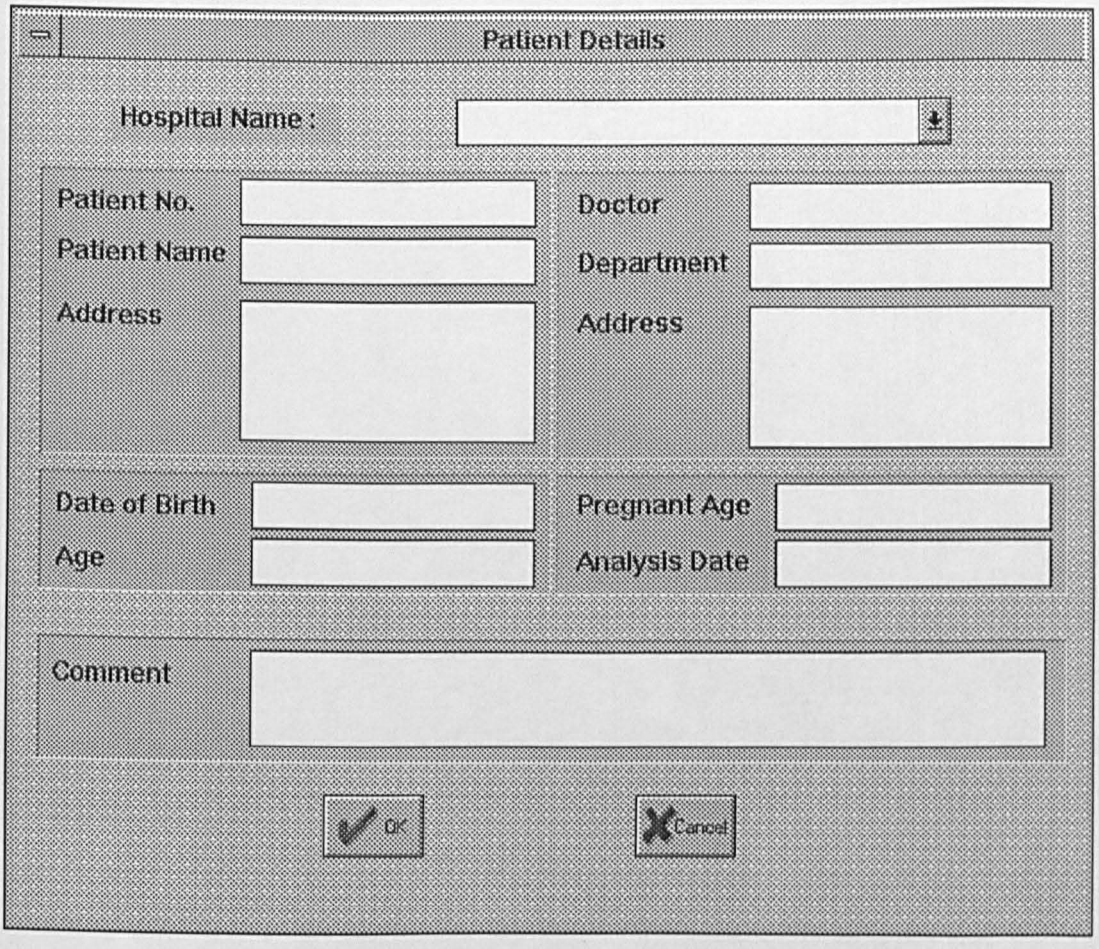


Figure I.3: Dialog box for patient details

I.2 Process and analyze fetal ECGs

Figure I.4 shows the analysis stage of the program. Once the start button is pressed, processing begins, as explained in Chapter 2. A button will also appear to allow the user to stop processing. The user may exit the analysis section at any time by pressing the *exit* button. Only the active buttons are displayed, whilst inactive buttons are hidden. For instance, leaving the *Start* button displayed may be confusing once it has been pressed, and processing commenced, so this button is hidden from the user. The figure shows an example of processing a file when the stop button has been pressed.

At this point the user could press:

1. *Continue*: to continue processing the file from the stop position;
2. *Restart*: to start processing from the beginning of the file;
3. *Redraw windows*: to redraw the windows in their original position (if they have been moved);
4. *Exit*: to exit the analysis stage.

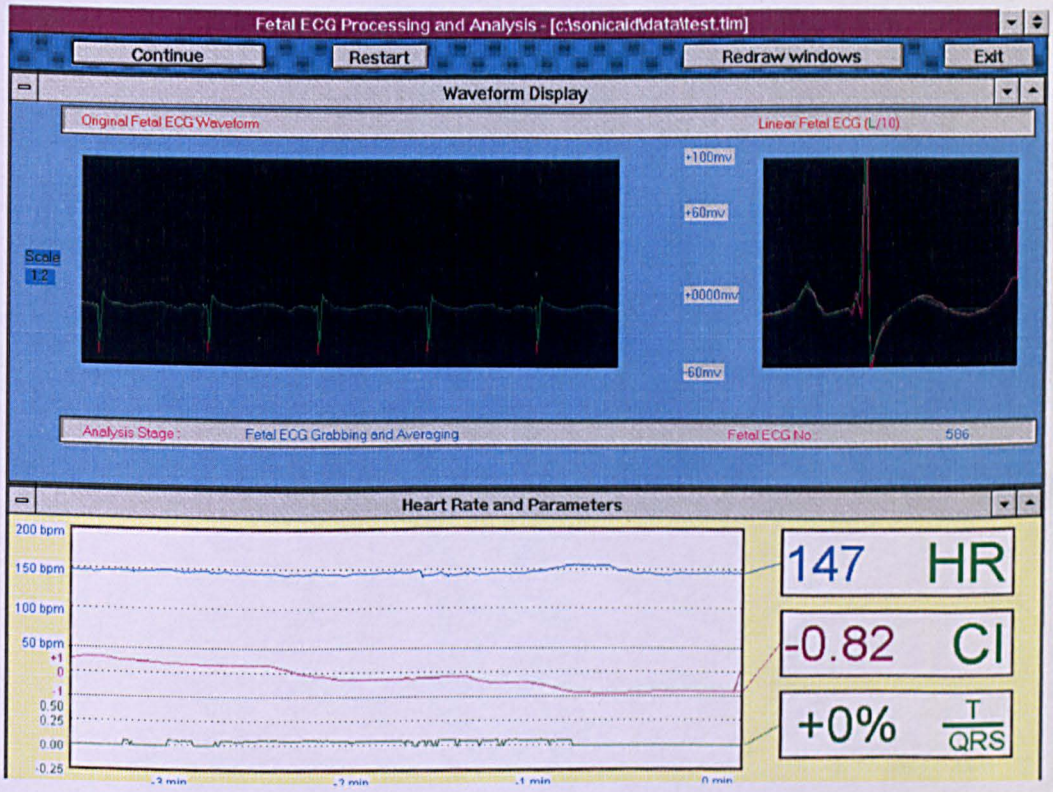


Figure I.4: Analysis window

The FECG Processing window contains 2 child windows: the *Waveform Display* window and the *Heart Rate and Parameters* window.

Waveform display window

On the left, 2 seconds of the raw signal is displayed, with each detected R-peak marked. On the right, the averaged waveform is displayed, and this is overlaid with the linear model, in red. A description to the processing stage, and the total number of detected complexes are also given.

Heart Rate and Parameters window

A trace of the heart rate, Conduction Index value and T/QRS ratio is shown, along with a clear display of their present values.

Appendix II : Analysis software

II.1 Development Files

Filename	Header filename	Description
FECGV1.PRJ		Borland C++ IDE project file
FECGV1.DSK		Borland C++ IDE desktop file
STARTUP.CPP	STARTUP.H	TBabyApp class
	IDENT.H	Contains list of identifiers
	SCREEN.H	Defines maximum screen coordinates
	COLOURS.H	Defines some program colours
MAINMENU.CPP	MAINMENU.H	TMainWindow class
	ENUMS.H	Defines an enum for the 4 types of window created from the main menu
POSTER.CPP	POSTER.H	TPoster class
MENUBTN.CPP	MENUBTN.H	TMenuButton class
SUB.CPP	SUB.H	TSubWindow class
SETUP.CPP	SETUP.H	TSetup class
	STRUCT.H	Defines structures for Setup class
	DEFINE.H	Defines maximum character lengths for various patient details
DISPLAY.CPP	DISPLAY.H	TDisplayWindow, TSlideWindow and TScrollWindow classes
ANALYSIS.CPP	ANALYSIS.H	TAnalysisWindow class
	TDATAIN.H	Base class for TDataMeridian and TDataInfile classes.
TDATAMER.CPP	TDATAMER.H	TDataMeridian class
TDATAFIL.CPP	TDATAFIL.H	TDataInfile class
ANASUB.CPP	ANASUB.H	TAnaSubWindow class
HEART.CPP	HEART.H	THeart class
WAVEFORM.CPP	WAVEFORM.H	TWaveform class
ECG.CPP	ECG.H	TECGProcessing class
	FILTER.H	Contains constants for filtering ECG
FECG.CPP	FECG.H	TFECGProcessing class
MODEL.CPP	MODEL.H	TModel class
RESOURCE.RC	UNIV.BMP FECGPOST.BMP SETUP.BMP ANALY.BMP REVIEW.BMP EXIT.BMP +	FETALBW.ICO HRWND.ICO WAVWND.ICO ANAWND.ICO SETWND.ICO FETAL.ICO ABOUT.ICO	Resource file, bitmaps and icons
DEFAULT.DEF		Module definition
BWCC.LIB		Borland C++ library
GCPBLW.LIB		Greenleaf library

II.2 Program run-time files

Filename	Description
BWCC.DLL	Borland dynamic link library
GCPBDW.DLL	Greenleaf dynamic link library

II.3 Objects

This section lists all the objects used. It is given in a format similar to the class reference in the *ObjectWindows User's Guide* [Borland International, 1991]. Many of the properties of a class are inherited from base classes. Rather than duplicate this information, only new data members are listed. Figure II.1 shows the class hierarchy. Classes taken from the ObjectWindows library are shown in **bold**. The structure for the classes is shown in Figure II.2.

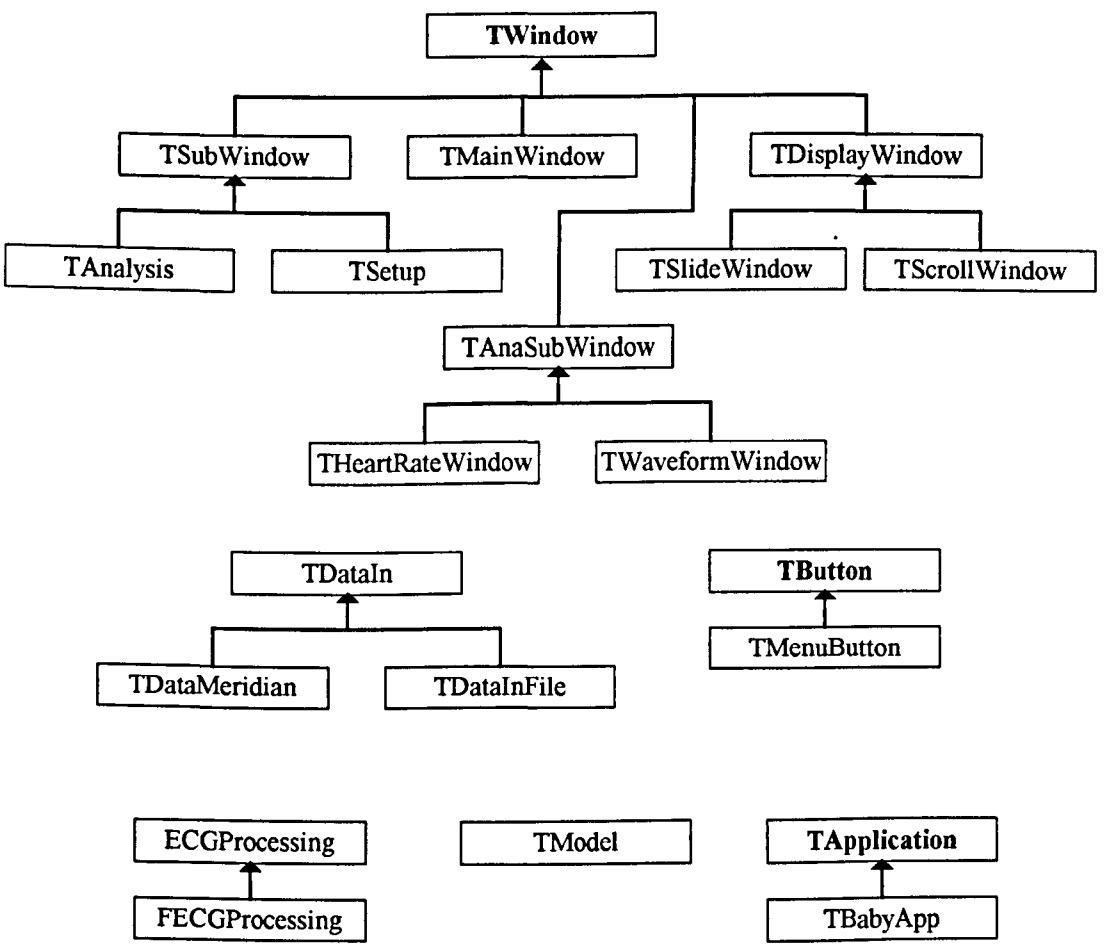


Figure II.1: Class hierarchy

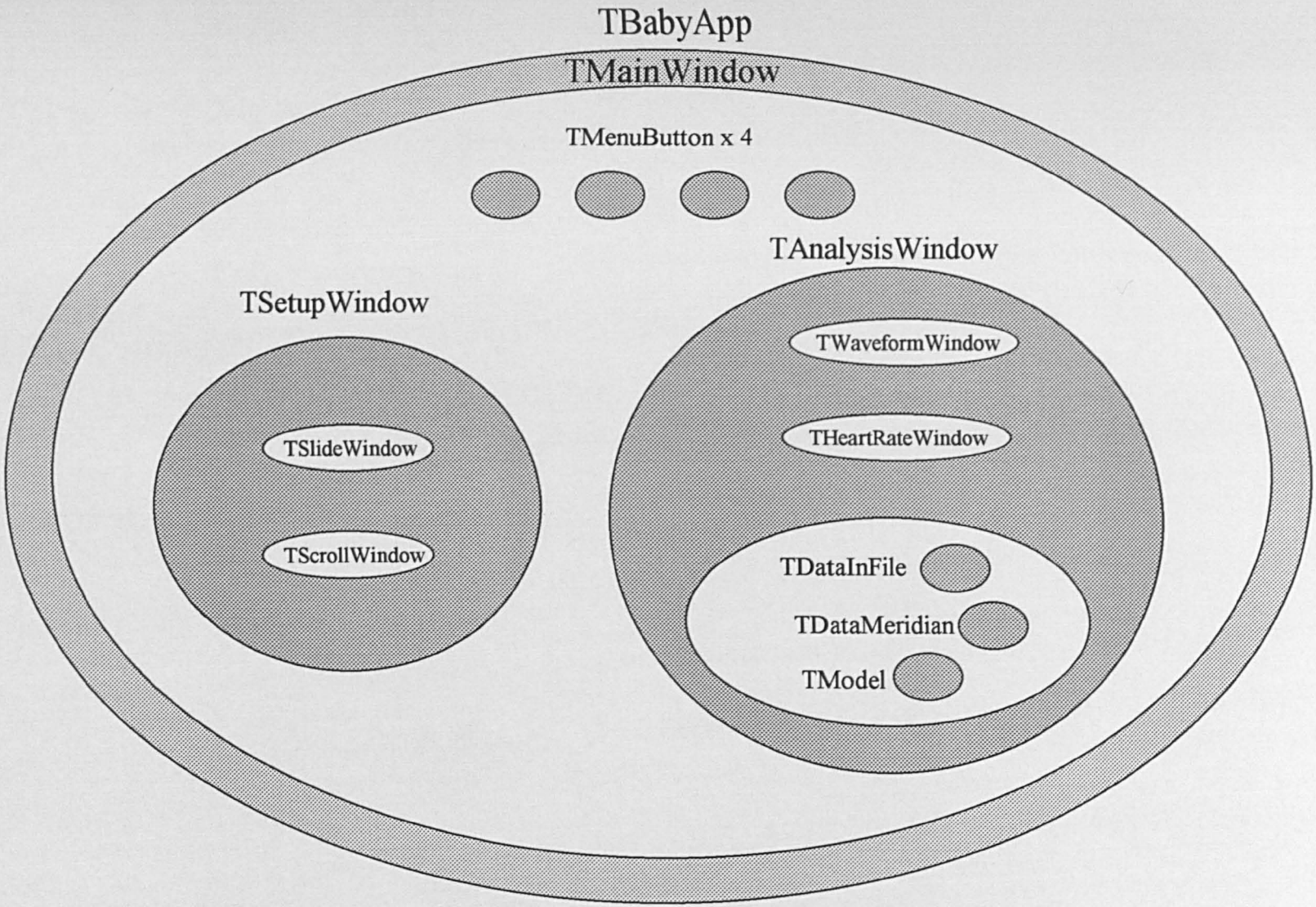


Figure II.2: Class structure

ECGProcessing ECG.CPP

Averaged_Data[400]	Grabbed_Data[400]
bAllQRSGrabbed	HRBuffer
bQRSDetectOff	HR_SmallArr[10]
bRDetected	Message[100]
bSetupDone	R_Value[10]
bSetupToDo	wFiltIndex
byCloseCount	wGrabEnd
byRPeakCount	wGrabStart
dwNoOfComplexes	wHRIndex
ECGBuffer	wLastRPeakPos
ElapsedTime	wRPos[10]
Filt_Data[32]	wStartAddr
ECGProcessing	Md
~ECGProcessing	Reset
Average	TransferHR
Initialize	

This is an abstract class, from which FECGProcessing is inherited. The code is written in this manner, so that a processing class for removing a Maternal ECG could easily be added.

Data members

Averaged_Data[400] `float Averaged_Data[400];`

Array to be used to store average ECG waveform.

bAllQRSGrabbed `BOOL bAllQRSGrabbed;`

Flag to indicate if all the QRS complexes, which were detected, have been processed. This flag should be set FALSE, if the R-peak occurs too late in the array, so that the whole complex cannot be processed.

bQRSDetectOff `BOOL bQRSDetectOff;` **(protected)**

Flag to be used to indicate if QRS detection is disabled. It should be set to TRUE for a fixed duration (i.e. 50 samples) after an R peak, to inhibit multiple R peak detection.

bRDetected, `BOOL bRDetected;` **(protected)**

Flag to be used to indicate if an R peak is currently being detected. It should remain TRUE until the R peak position has been established.

bSetupDone `BOOL bSetupDone;` **(protected)**

Flag to indicate if set-up phase has been executed.

bSetupToDo `BOOL bSetupToDo;`

Flag to indicate if set-up phase needs to be executed. It is the inverse of bSetupDone.

byCloseCount `unsigned char byCloseCount;` **(protected)**

Counter to be used to indicate the number of elapsed samples since an R-peak. When it reaches a given number (i.e. 50), it should be used to reset the bQRSDetectOff flag to FALSE.

byRPeakCount `unsigned char byRPeakCount;`

To contain the number of R-peaks detected in the ECG data presently being processed.

dwNoOfComplexes `unsigned long dwNoOfComplexes;`

To be used to store the total number of QRS complexes, which have been detected since the start of processing.

ECGBuffer `float *ECGBuffer;`

Pointer to buffer containing ECG data. This memory is allocated in the constructor, and is of size BUFFER_SAMPLES (2000 samples). Blocks of size PROCESS_SAMPLES (1000), containing new data, should be stored at index wStartAdd. Data blocks of size PROCESS_SAMPLES should then be processed from this array.

ElapsedTime `TimeType ElapsedTime;` **(protected)**

Elapsed processing time. This is incremented in Reset.

Filt_Data[32] `double Filt_Data[32];` **(protected)**

Circular buffer to be used for the most recent filtered ECG data. This buffer should be filled (at index wFiltData) with subsequent filtered ECG data, and wFiltData should be incremented accordingly. Previous filtered values may then be accessed.

Grabbed_Data[400] `float Grabbed_Data[400];` **(protected)**

Array to be used to store the latest ECG waveform.

HRBuffer `float *HRBuffer;`

Pointer to an array containing the previous Heart Rates. The array is created in the constructor and has size HRBUFFERSIZE (1024). The array should be updated by calling TransferHR.

HR_SmallArr[10] `float HR_SmallArr[10];`

Array to contain the Heart Rates for each R-R interval detected in the last 2 seconds.

Message[100] `char Message[100];`

Text message of present processing status.

R_Value[10] `float R_Value[10];`

Array to store R-peak amplitudes in latest processed block.

wFiltIndex `unsigned int wFiltIndex;` **(protected)**

To be used to indicate index, where next filtered data point should be placed in array Filt_Data.

wGrabEnd `unsigned int wGrabEnd;`

When the latest ECG waveform is stored in Grabbed_Data, wGrabEnd data points should be stored before the R peak. wGrabEnd should therefore be set before averaging is carried out.

wGrabStart `unsigned int wGrabStart;`

When the latest ECG waveform is stored in Grabbed_Data, wGrabStart data points should be stored after the R peak. wGrabStart should therefore be set before averaging is carried out.

wHRIndex `unsigned int wHRIndex;`

Index for HRBuffer, indicating where next heart rate will be entered.

wLastRPeakPos `unsigned int wLastRPeakPos;` **(protected)**

Index to indicate position of last R-peak detected in ECGBuffer.

wRPos[10] `unsigned int wRPos[10]`

Indexes to R-peaks locations in the array ECGBuffer, should be stored in wRPos for the processed data block.

wStartAddr `unsigned int wStartAddr;`

Index indicating where new blocks of ECG data should be stored. wStartAddr is updated automatically within Reset.

Member functions

Constructor `ECGProcessing();`

Allocates memory for ECGBuffer and HRBuffer; sets bSetupDone and bSetupToDo flags to FALSE and TRUE respectively; sets Message and sets dwNoOfComplexes to zero.

Destructor `~ECGProcessing();`

Frees up memory used for ECGBuffer and HRBuffer.

Average
```
void Average(float *NewECG, float *AvECG,
             long dwNumber, int iTotal,
             int iStart, int iEnd);   (protected)
```

This function calculates a running average of the ECG waveform. *NewECG* is a pointer to an array containing the latest ECG waveform, and *AvECG* is a pointer to the latest average ECG waveform. *dwNumber* is the total number of complexes found so far, and *iTotal* is the number of complexes over which one wants to average. *iStart* and *iEnd* contain the number of samples required before and after the R-peak. The new average is passed back to the array at **AvECG*.

Initialize `virtual void Initialize();`

This function should be called before any processing begins in order to initialise certain variables. The following variables are initialised: bSetupDone, bSetupToDo, dwNoOfComplexes, wLastRPeakPos, byRPeakCount, byCloseCount, wFiltIndex, wHRIndex, bQRSDetectOff, bRDetected, bAllQRSGrabbed and ElapsedTime. The following arrays are initialised: Filt_Data, HRBuffer, Grabbed_Data, Averaged_Data.

Md `int Md(int Index, int Max);`

Returns *Index* modulus *Max*.

Reset `virtual void Reset();`

This routine must be called before a new buffer is read in and processed. When bAllQRSGrabbed flag is FALSE, the last buffer contains part of an ECG complex, and certain variables need to be carried forward to be used in the next buffer. Also the timer in incremented, and the Message string is set.

TransferHR `void TransferHR();` **(protected)**

Transfers the calculated heart rates from HR_SmallArr to a buffer pointed to by HRBuffer. 6 heart rates are added at location wHRIndex. wHRIndex is incremented by 6. 6 heart rates will be stored irrespective of the number of complexes detected. Thus, some heart rates may be stored several times (when the HR is low), or not at all (if the heart rate is too fast).

FECGProcessing — FECG.CPP

AverageBased[250]	Param_SmallArr[2][10]
CIBuffer	TQRSBuffer
iInvert	wCIIndex
IsEndofFile	wTQRSIndex
Model	
FECGProcessing	Init_IO
~FECGProcessing	Is_IO_OK
End_IO	ProcessECG
Initialize	Setup

FECGProcessing is derived from ECGProcessing, and carries out all the analysis calculations described in chapter 2.

Data members

AverageBased[250] `float AverageBased[250];`

Array containing averaged FECG, with baseline removed.

CIBuffer `float *CIBuffer;`

Pointer to an array containing the previous CI values. The array is created in the constructor, and is of size CIBUFFERSIZE (1024). The array is updated when ProcessECG is called.

iInvert `int iInvert;`

Indicates orientation of the signal: +1 indicates the signal is normal; -1 indicates it is inverted.

IsEndofFile `BOOL IsEndofFile;`

Flag to indicate if the end of file has been reached, when input data is from a file.

Model `TModel *Model;`

Pointer to TModel, containing the linear model of the ECG. This is updated from ProcessECG.

Param_SmallArr[2][10] `float Param_SmallArr[2][10];`

Array containing the PR and T/QRS values for each complex detected in the last 2 seconds. Param_SmallArr[0] contains the PR intervals and Param_SmallArr[1] contains the T/QRS ratios.

TQRSBuffer `float *TQRSBuffer;`

Pointer to an array containing the previous T/QRS values. The array is created in the constructor, and is of size CIBUFFERSIZE (1024). The array is updated when ProcessECG is called.

wCIIndex `unsigned int wCIIndex;`

Index for CIBuffer, indicating where next Conduction Index value will be entered.

wTQRSIndex `unsigned int wTQRSIndex;`

Index for TQRSBuffer, indicating where next T/QRS value will be entered.

Member functions

Constructor `FECGProcessing();`

Allocates memory for CIBuffer, TQRSBuffer and various internal buffers, and opens parameter file. Creates instance of either **TDataInFile** of **TDataMeridian**, and an instance of TModel, pointed to by Model.

Destructor `~FECGProcessing();`

Closes parameter file and frees up memory.

End_IO `void End_IO();`

Terminates input stream (from either file or Meridian) and output file stream.

Initialize `void Initialize();`

This function should be called before any processing begins in order to initialise certain variables. Firstly ECG::Initialise() is called. Then the following variables are initialised: IsEndofFile, wStartAddr, wTQRSIndex, iInvert, wCIIndex, and other internal variables. The arrays AverageBased, TQRSBuffer, CIBuffer, Param_SmallArr and other internal arrays are also initialised.

Init_IO `unsigned char Init_IO();`

Initialises input file stream (from either file or Meridian) and output file stream for raw data.

Is_IO_OK `unsigned char Is_IO_OK();`

Checks if input data stream is OK.

ProcessECG `void ProcessECG();`

This processes the raw waveform to calculate the average waveform, the linear model, and the ECG parameters, and stores them to disk. This is achieved as follows:

Reset is called first. Then the ECG data is read into ECGBuffer from the input stream and this data is saved to disk. IsEndOfFile is set to TRUE if the input stream is from disk, and all the file has been read.

R-peak detection is then carried out on this data. Variables, bQRSDetectOff, bRDetected, byCloseCount, Filt_Data, wFiltIndex, HR_SmallArr, R_Value, wLastRPeakPos, and WRPos are set appropriately.

Each subsequent waveform is stored in Grabbed_Data and dwNoOfComplexes is incremented for each QRS complex detected. If the waveform is not too noisy, Average is called to produce the latest average waveform, which is stored in Averaged_Data. A copy of Averaged_Data is made to AverageBased (but no baseline drift is removed). A linear model is applied by calling Model→Fit. The ECG parameters are calculated and stored in CIBuffer, TQRSBuffer and Param_SmallArr (wCIIndex and wTQRSIndex are incremented accordingly).

If the HR appears erroneous for too long, the bSetupToDo flag is set to TRUE, and the bSetupDone flag is set to FALSE. If an R-peak is detected too late in the currently processed buffer for the whole waveform to be grabbed, flag bAllQRSGrabbed is set to FALSE. TransferHR is called to transfer the Heart Rates to array HRBuffer. These parameters are then saved to disk.

Setup `void Setup();`

This function must be called once at the beginning of ECG processing. It calculates the threshold required, and the orientation of the data. This is achieved as follows:

Reset is called first. Then the ECG data is read into ECGBuffer from the input stream and this data is saved to disk. Crude R-peak detection is then carried out on this data. Variables, iInvert, bQRSDetectOff, bRDetected, byCloseCount, Filt_Data, wFiltIndex, wLastRPeakPos, WRPos, wGrabStart, and wGrabEnd are set appropriately. An empty set of parameters are then saved to disk. Flag bSetupDone is set to TRUE.

TAnalysis — ANALYSIS.CPP

bIsStarted HeartRateWindow	WaveformWindow
TAnalysis	FECGProcessLoop
~TAnalysis	Paint
CanClose	SetupWindow
ContinueClicked	StartClicked
ExitClicked	TileClicked

TAnalysis is derived from TSubWindow. This window declares an instance of FECGProcessing class to analyse the FECG data. Buttons are defined to control this processing and two windows, instances of **TWaveformWindow** and **THeartRateWindow** are created to display the results.

Data members

bIsStarted

```
BOOL bIsStarted;
```

Flag to indicate if the FECG is being currently processed.

HeartRateWindow

```
THeartRateWindow* HeartRateWindow;
```

Pointer to the instance of THeartRateWindow.

WaveformWindow

```
TWaveformWindow* WaveformWindow;
```

Pointer to the instance of TWaveformWindow.

Member functions

Constructor

```
TAnalysisWindow(PTWindowsObject AParent );
```

Invokes TSubWindow constructor, passing *AParent*, and SW_ANALYSIS (indicating window type). Defines buttons, and constructs instance of TWaveformWindow, THeartRateWindow, and FECGProcessing class.

Destructor

```
~TAnalysisWindow();
```

Sets pointers to sub windows to NULL.

CanClose

```
virtual BOOL CanClose();
```

Returns TRUE.

ContinueClicked

```
virtual void ContinueClicked(RTMessage Msg )
        = [ ID_FIRST + ID_CONTINUE];
```

Responds to an ID_CONTINUE message, by showing and hiding various buttons, before calling FECGProcessLoop.

ExitClicked

```
virtual void ExitClicked(RTMessage Msg )
        = [ ID_FIRST + ID_EXIT];
```

Responds to an ID_EXIT message, by closing window immediately if *Close as soon as possible* flag not set.

FECGProcessLoop

```
void FECGProcessLoop();
```

This is the main loop for processing the raw signal. FECG→Init_IO is called to initialise input and output operations. FECG→Setup is then called to initialise the threshold and orientation of the data. Next FECG→ProcessECG and WaveformWindow→AutoScale are called repeatedly. WaveformWindow→Plot and THeartRateWindow→Plot are also called to display the data. This continues until either the ID_STOP or ID_EXIT message is received, or the end of the input file is reached (if the data is being read from a file). FECG→End_IO called to end input and output operations.

Paint

```
virtual void Paint(HDC PaintDC,
                   PAINTSTRUCT _FAR & PaintInfo);
```

Calls TSubWindow::Paint, passing *PaintDC* and *PaintInfo* structure contains information about the paint operation. Uses *PaintDC* as the paint display context. The supplied reference to the *PaintInfo* structure contains information about the paint operation. Draws background around buttons and writes patient name and date at top. If the command line contains an argument, processing will begin on the specified filename, and then windows will be shut down.

SetupWindow

```
virtual void SetupWindow();
```

Change caption and show or hide various buttons when the window is set-up.

StartClicked

```
virtual void StartClicked(RTMessage Msg )
        = [ ID_FIRST + ID_STARTBUT];
```

Responds to an ID_STARTBUT message. Firstly if FECG→Is_OK returns TRUE, Initialize is called to reset all processing variables. Secondly various control buttons are hidden or shown. Thirdly FECGProcessLoop is called to begin processing.

TileClicked

```
virtual void TileClicked( RTMessage Msg )
       = [ ID_FIRST + ID_TILE];
```

Responds to an ID_TILE message, by retiling subwindows.

TAnaSubWindow — TANASUB.CPP

FECG
hAnaSubWndIcon
hBKBrush
XFactor
YFactor
TAnaSubWindow
CanClose
DrawFrame
ScaleRect
SetFrame
WMSetMaxSize

TAnaSubWindow is derived from TWindow. It defines an abstract class, from which the two display windows, **TWaveformWindow** and **THeartRateWindow**, are inherited.

Data members

FECG `FECGProcessing* FECG;` **(protected)**

Handle to FECGProcessing object, from which data will be obtained.

hAnaSubWndIcon `HICON hAnaSubWndIcon;` **(protected)**

Handle to window's icon.

hBKBrush `HBRUSH hBKBrush;` **(protected)**

Handle to background brush.

XFactor `float XFactor;` **(protected)**

Scale factor to scale x co-ordinates from a pixel (800) wide screen to screen being used.

YFactor `float YFactor;` **(protected)**

Scale factor to scale y co-ordinates from a pixel (800) high screen to screen being used.

Member functions

Constructor

```
TAnaSubWindow(PTWindowsObject AParent,
              LPSTR AName,
              FECGProcessing* FECGPro);
```

Invokes TWindow constructor, passing *AParent* and *AName*. Copies *FECGPro* pointer to FECG, and defines window attributes.

CanClose

```
virtual BOOL CanClose()
```

Returns FALSE.

DrawFrame

```
void DrawFrame(HDC PlotDC, RECT FAR* PTFrame,
               COLORREF FrameColor,
               int fillcolor);                    (protected)
```

Draws a frame of colour FrameColor, around a rectangle of colour fillcolor.

ScaleRect

```
void ScaleRect(RECT FAR* Rect, int left, int top,
               int right, int bottom);    (protected)
```

Takes *left*, *top*, *right*, and *bottom* co-ordinates of rectangle, scales them in the x-direction and y-direction by XFactor and YFactor, and stores the result in the structure *Rect*.

SetFrame

```
void SetFrame(RECT FAR* PTRect,                   (protected)
              RECT FAR* PTFrame, int size);
```

Takes the rectangle stored in *PTRect*, put a frame of width *size* around it, and stores result in *PTFrame*.

WMSetMaxSize

```
virtual void WMSetMaxSize(RTMessage Msg)
        = [ WM_FIRST + WM_GETMINMAXINFO ];
```

Sets maximum size of window (i.e. stops buttons being covered when window maximised).

TBabyApp — STARTUP.CPP



TBabyApp is derived from TApplicationWindow. This is the application class.

Member functions

Constructor

```
TBabyApp(LPSTR name, HINSTANCE hInstance,
         INSTANCE hPrevInstance, LPSTR lpCmd,
         int nCmdShow);
```

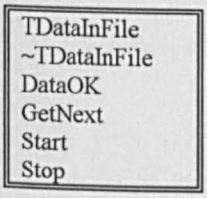
Invokes TApplication constructor, passing *name*, *hInstance* and *hPrevInstance*, *lpCmd* and *nCmdShow*.

InitMainWindow

```
virtual void InitMainWindow();
```

Creates instance of **TMainWindow**.

TDataInFile — TDATAFIL.CPP



This class controls data input from a file

Member functions

Constructor

```
TDataInFile(char* NameofFile);
```

Opens input file, *NameofFile,* and skips over header.

Destructor

```
~TDataInFile();
```

Close input file (if it exists).

DataOK

```
virtual BOOL DataOK();
```

Checks if there is enough data left in the file, to fill buffer with PROCESS_SAMPLES samples.

GetNext `virtual float GetNext(char *);`

Gets next sample from data file.

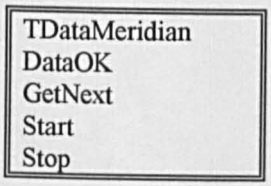
Start `virtual BOOL Start(char *);`

Returns TRUE.

Stop `virtual void Stop();`

Empty.

TDataMeridian TDATAMER.CPP



This class controls data input from a Meridian 800.

Member functions

Constructor `TDataMeridian();`

Empty.

DataOK `virtual BOOL DataOK();`

Returns TRUE.

GetNext `virtual float GetNext(char * Message);`

Returns next data sample from Meridian. Changes *Message* if a problem occurs.

Start `virtual BOOL Start(char * Message);`

Initialises communications with Meridian. Returns TRUE if successful. Any error is reported as text within *Message*.

Stop `virtual void Stop();`

Ends communications with Meridian.

TDisplayWindow DISPLAY.CPP

TDisplayWindow is derived from TWindow. It is the base class, from which **TSlideWindow** and **TScrollWindow** are inherited.

THeartRateWindow — HEART.CPP

THeartRateWindow
~THeartRateWindow
GetClassName
GetWindowClass
Paint
Plot

Class for displaying heart rate, conduction index, and T/QRS ratio.

Member functions

Constructor

```
THeartRateWindow(PTWindowsObject AParent,
                 LPSTR AName,FECGProcessing* FECG);
```

Invokes TAnaSubWindow constructor, passing *AParent*, *AName* and *FECG*. Sets attributes, loads icon, creates fonts, pens and rectangles.

Destructor

```
~THeartRateWindow();
```

Deletes rectangles, fonts, pens and brush and destroys icon.

GetClassName

```
virtual LPSTR GetClassName();
```

Returns icon name instead of *OWLWindow* to change icon and background used.

GetWindowClass

```
virtual void GetWindowClass
                        (WNDCLASS& AAnaSubWndClass);
```

Calls TWindow::GetClassName, then defines different background brush and icon of *AAnaSubWndClass*.

Paint

```
void Paint( HDC PaintDC, PAINTSTRUCT _FAR & );
```

Sets scale factors for window, scales rectangles, fonts and pens, then draws window. Uses *PaintDC* as the paint display context.

Plot

```
void Plot();
```

Updates the window with latest values for HR, CI & T/QRS. This should be called every 2 seconds.

TMainWindow — MAINMENU.CPP

TMainWindow
~TMainWindow
AnalysisClicked
CanClose
ExitClicked
GetWindowClass
HelpClicked
Paint
SetupClicked
ShowSubWindow

TMainWindow is derived from TWindow. TMainWindow first creates an instance of **TPoster** to display the bitmap title screen. Four buttons of type **TMenuButton** are created on the main menu to:

1. Create **TSetup** - to configure the system,
2. Create **TAnalysis** - to run the analysis section,
3. Provide help to the user,
4. Exit the program.

Member functions

Constructor

```
TMainWindow(LPSTR ATitle );
```

Invokes TWindow constructor, passing *ATitle* and settings for a main window. Defines window attributes, scale-factors, menu buttons; loads icon and university bitmap; and creates instance of **TPoster**.

Destructor

```
~TMainWindow();
```

Deletes icon and university bitmap.

AnalysisClicked

```
virtual void AnalysisClicked(TMessage& Msg )
        = [ ID_FIRST + ID_ANALBUTTON ];
```

Responds to a ID_SETUPBUTTON message, by displaying analysis window.

CanClose

```
virtual BOOL CanClose();
```

Returns TRUE if there are no subwindows, otherwise message appears to check if user really wants to end program.

ExitClicked

```
virtual void ExitClicked(TMessage& Msg )
        = [ ID_FIRST + ID_EXITBUTTON ];
```

Responds to a ID_SETUPBUTTON message, by closing window.

GetWindowClass

```
virtual void GetWindowClass(WNDCLASS &AWndClass);
```

Calls TWindow::GetClassName, then defines different background brush and icon of *AWndClass*.

HelpClicked

```
virtual void HelpClicked(TMessage& Msg )
        = [ ID_FIRST + ID_HELPBUTTON ];
```

Responds to a ID_SETUPBUTTON message, by displaying a help message.

Paint

```
virtual void Paint(HDC PaintDC,
                   PAINTSTRUCT _FAR & PaintInfo);
```

Draws four choice buttons with a description alongside. Uses *PaintDC* as the paint display context. The supplied reference to the *PaintInfo* structure contains information about the paint operation.

SetupClicked

```
virtual void SetupClicked(TMessage& Msg )
        = [ ID_FIRST + ID_SETUPBUTTON ];
```

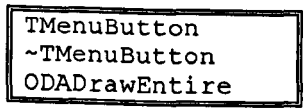
Responds to a ID_SETUPBUTTON message, by displaying Setup window.

ShowSubWindow

```
void ShowSubWindow(PTWindowsObject AParent,
                   TSubWinType ASubWinType);
```

If window *ASubWinType* doesn't already exist, it is created (i.e. instances **TSetup**, or **TAnalysis** created), otherwise, it (or its child) is made active. The parent window is supplied in *AParent*.

TMenuButton — MENUBTN.CPP



Derived from TButton, this class displays a button with a bitmap drawn on it. This has been done to make use of the software more obvious and user-friendly.

Member functions

Constructor

```
TMenuButton (PTWindowsObject AParent, int AnId,
             int X, int Y, int W, int H,
             int ButtonNo );
```

Invokes TButton constructor, passing it *AParent*, *AnId*, *X*, *Y*, *W*, *H*, but no text. Sets window style attributes and loads relevant bitmap according to *ButtonNo*.

Destructor

```
virtual ~TMenuButton ();
```

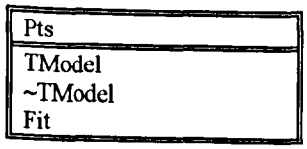
Deletes bitmap.

ODADrawEntire

```
virtual void ODADrawEntire
                    (DRAWITEMSTRUCT far &DrawInfo);
```

Draws button on notification that control needs drawing. The supplied reference to the *PaintInfo* structure contains information about the paint operation.

TModel — MODEL.CPP



This fits a linear model to the averaged ECG data (passed to the function Fit) and calculates the points of intersection.

Data members

Pts

```
FitPointType *Pts;
```

Pointer to an array, containing the points of intersection of the lines on the linear model. The array is created in the constructor.

Member functions

Constructor `TModel();`

Initialise the linear model lines, and their points of intersection, Pts.

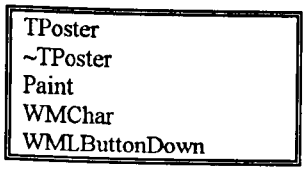
Destructor `~TModel();`

Free up memory allocated for Pts.

Fit `void Fit( float* data);`

This function takes the waveform stored in *data*, fits a series of 15 least-squares lines to the waveform, calculates their intersections, which is stored in the array, pointed to by Pts.

TPoster — POSTER.CPP



TPoster is derived from TWindow. This class displays the title screen, a bitmap, which is scaled for the screen size. Once a key, or the left mouse button, is pressed, the window is closed.

Member functions

Constructor
```
TPostWindow( PTWindowsObject AParent );
```

Invokes TWindow constructor, passing *AParent* and settings for a main window. Sets window style attributes and loads bitmap.

Destructor
```
virtual ~TPostWindow();
```

Deletes bitmap.

Paint
```
virtual void Paint(HDC PaintDC,
                   PAINTSTRUCT _FAR & PaintInfo);
```

Copies bitmap onto screen (with scaling), and sets focus to this window. Uses *PaintDC* as the paint display context. The supplied reference to the *PaintInfo* structure contains information about the paint operation.

WMChar
```
void WMChar(RTMessage Msg )
       = [ WM_FIRST + WM_CHAR];
```

Responds to WM_CHAR message, by shutting down window.

WMLButtonDown
```
virtual void WMLButtonDown(RTMessage Msg );
```

Responds to WM_LBUTTONDOWN message, by shutting down window.

TScrollWindow — DISPLAY.CPP

TScrollWindow scrolls through the raw ECG data.

TSetup — SETUP.CPP

This class contains the functions for entering the patient details, and setting up the system configuration, as explained previously. It allows the user to set up instances of **TSlideWindow** or **TScrollWindow** to view data.

TSlideWindow — DISPLAY.CPP

TSlideWindow displays the raw ECG data, in a similar manner to a conventional oscilloscope.

TSubWindow — SUB.CPP

Constructor
Destructor
GetWindowClass
GetClassName
Paint

TSubWindow is derived from TWindow. TSubWindow is an abstract class, from which the **TSetup** and **TAnalysis** subwindows are inherited. It defines the basic window attributes, icon, and draws the university logo in the corner of the client area.

Member functions

Constructor

```
TSubWindow(PTWindowsObject AParent,
           TSubWinType ASubWinType);
```

Invokes TWindow constructor, passing *AParent* and correct title, depending on the subwindow type passed by *ASubWinType*. Defines window attributes, and loads subwindow icon.

Destructor

```
virtual ~TSubWindow();
```

Sets relevant global variable SubWinPtr pointer to NULL to indicate window has been shut. Destroys icon.

GetWindowClass

```
virtual void GetWindowClass
                          (WNDCLASS & ASubWndClass);
```

Calls TWindow::GetClassName, then defines different background brush and icon of *AWndClass*.

GetClassName

```
virtual LPSTR GetClassName();
```

Returns icon name, to change icon and background used.

Paint

```
virtual void Paint(HDC PaintDC,
                   PAINTSTRUCT _FAR & PaintInfo);
```

Draws university logo bitmap in bottom right hand corner of client area. Uses *PaintDC* as the paint display context. The supplied reference to the *PaintInfo* structure contains information about the paint operation.

TWaveformWindow — WAVEFORM.CPP

TWaveformWindow
~TWaveformWindow
AutoScale
GetClassName
GetWindowClass
Paint
Plot

This class displays the original, average and linear waveforms.

Member functions

Constructor

```
TWaveformWindow(PTWindowsObject AParent,
                LPSTR AName,FECGProcessing* FECG);
```

Invokes TAnaSubWindow constructor, passing *AParent*, *AName* and *FECG*. Sets attributes, loads icon, creates pens, initialises scale factor and font, and defines rectangles.

Destructor

```
~TWaveformWindow();
```

Deletes icon, background brush, font and rectangles.

AutoScale

```
void AutoScale();
```

Calculates scale-factor for original waveform and linear model, and displays them.

GetClassName

```
virtual LPSTR GetClassName();
```

Returns icon name instead of *OWLWindow* to change icon and background used.

GetWindowClass

```
virtual void GetWindowClass
                    (WNDCLASS & AAnaSubWndClass );
```

Calls TWindow::GetClassName, then defines different background brush and icon of *AAnaSubWndClass*.

Paint

```
void Paint( HDC PaintDC, PAINTSTRUCT _FAR & );
```

Sets scale factors for window, scales rectangles, changes font size and calls Plot. Uses *PaintDC* as the paint display context.

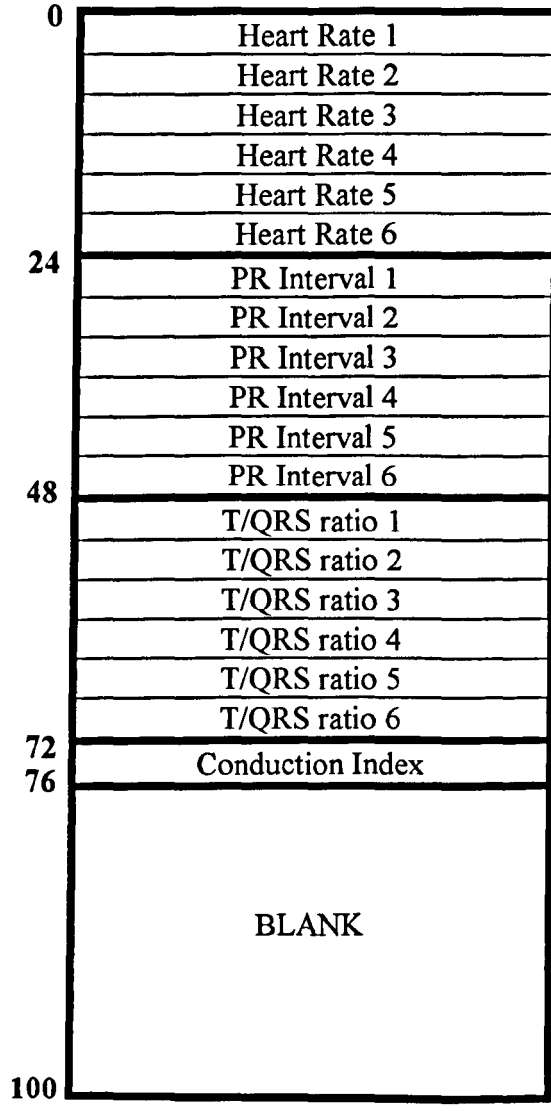
Plot

```
void Plot();
```

Plots the original, average and linear waveforms.

II.4 Parameter file

The file consists of 100-byte data blocks, each block relating to the parameters obtained from 2 seconds of FECG data. The parameters stored are up to 6 heart-rates (b.p.m.), up to 6 P-R intervals (ms), up to 6 T/QRS ratios and the Conduction Index value. These are stored as floats, which are each stored in 4 bytes, allowing 24 bytes free for any future parameters.



Appendix III : Simulator user guide

The windows user interface can be seen in Figure III.1. The main window contains 4 child windows, shown in Figure III.2, and these are detailed in this Appendix.

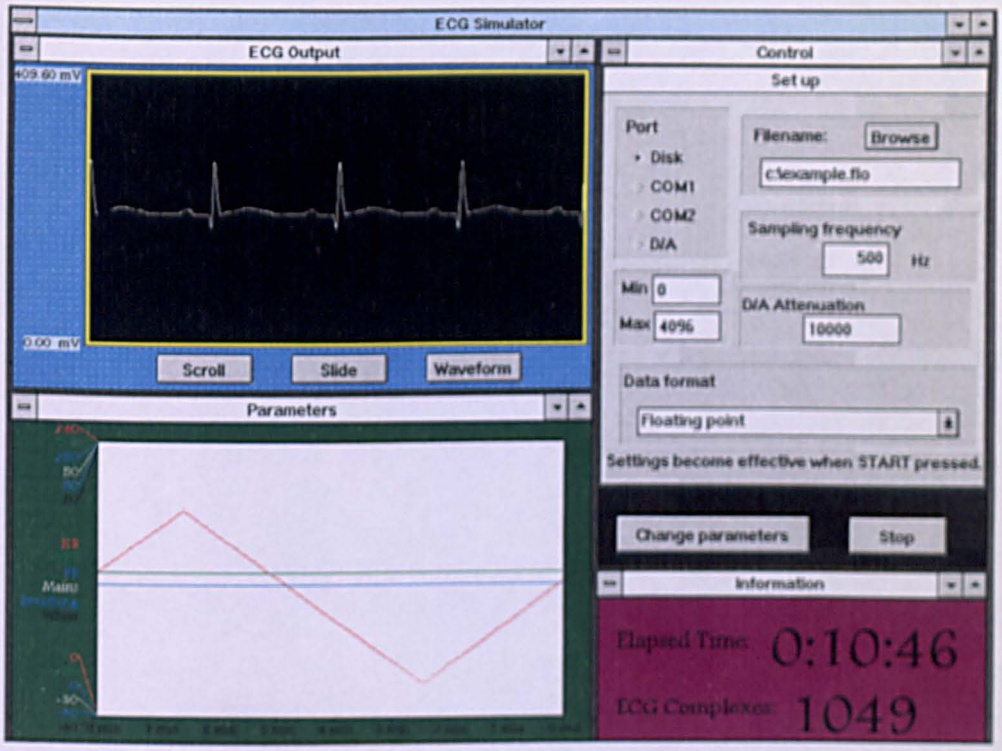
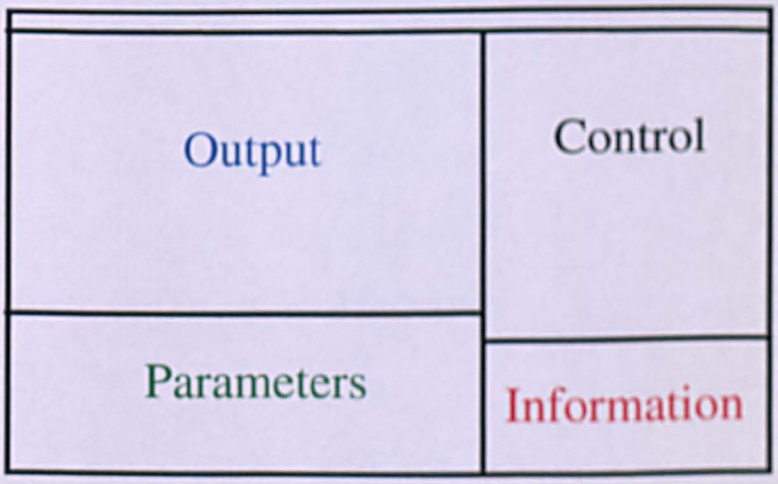


Figure III.1: ECG Simulator main display



III.1 Control window

In this window, the user determines the destination of the generated signal, the morphology of the ECG wave, and the level of added noise. Within the *Set-up* dialog box, the user has the following options:

Port	The output device for the data is to either disk, RS232 COM1, RS232 COM2, or the D/A card. If an RS232 port is selected, the data is output at 19200 baud consisting of 8 bit data, with no parity, and 1 stop bit.
Filename	If the disk is chosen within the *port* setting, the data is stored in the file specified under *filename*.
Sampling frequency	All data is output at 500 Hz, as the option within the dialog box to change the sampling frequency, is for future improvements.
Min & Max.	Limits generated data values to within these settings. These are also used to scale the output window.
D/A attenuation	This box, allows the user to specify the attenuation which will occur when the data is converted to an analogue signal. This is used within the simulation program only for displaying the voltage range within the output window.
Data format	The data format box, allows the user to specify the format of the data stored to **disk**. This may be either floating point data, or integer format (with or without a 20-byte dummy hypersignal header). This has no effect for other output devices.

Table III.1: Set-up options

By pressing the *change parameters* button, the *Change Parameters* dialog box is obtained, containing various scroll bars to adjust the ECG (*see* Figure III.3).

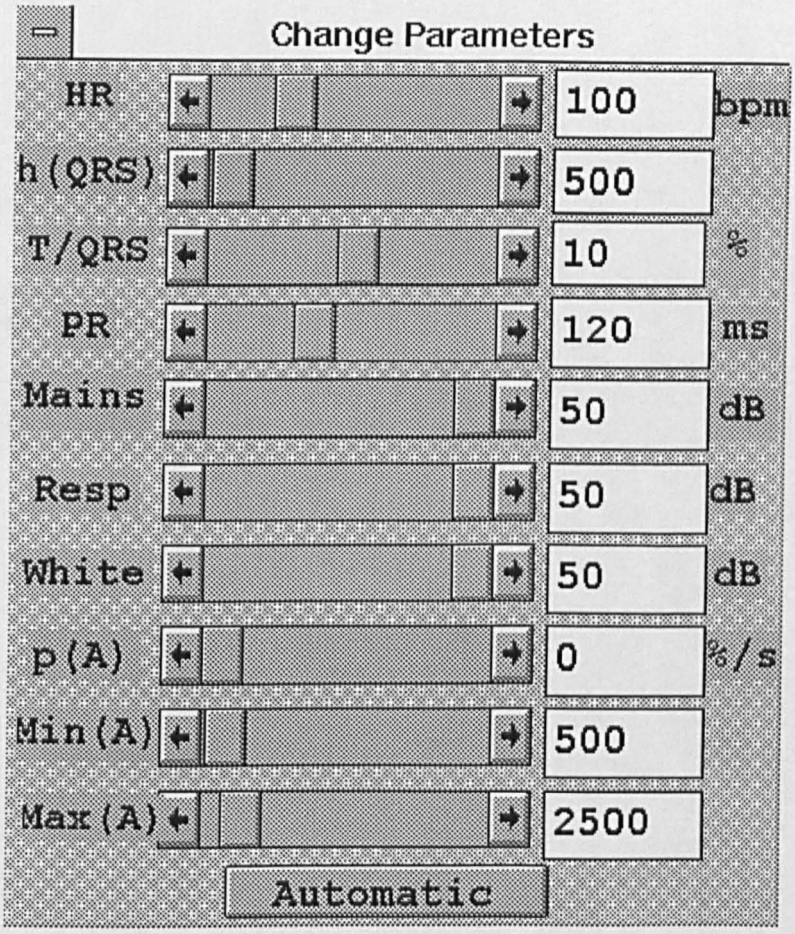


Figure III.3: *Change Parameters* dialog box

The effect of changing these scroll bars is detailed in Table III.2.

Scroll bar	Explanation
HR	Heart-Rate in beats per minute.
h(QRS)	Height of the QRS complex.
T/QRS	The T height is set to h(QRS)*T/QRS.
PR	PR Interval in milliseconds.
Mains	The mains (50 Hz) noise power, set as a signal-to-noise ratio in dB.
Resp	The respiration (0.3 Hz) power, set as a signal-to-noise ratio in dB.
White	The white noise power, set as a signal-to-noise ratio in dB.
p(A)	Probability (%) of a DC shift (artefact) occurring every second.
Min(A)	Minimum DC value
Max(A)	Maximum DC value. The initial DC value is set as the average of Min(A) and Max(A).

Table III.2: Explanation of scroll bars.

These can be changed automatically through use of the *Auto Update* dialog box (*see* Figure III.4), selected by pressing the *Automatic* button. The user may click any Radio button (or combination of Radio buttons), and the scroll bars in the *Change Parameters* window will be updated automatically. The user must enter the limits, and period of the variation, and the parameter will then vary linearly from the first limit to the second limit, and back, in the specified time period.

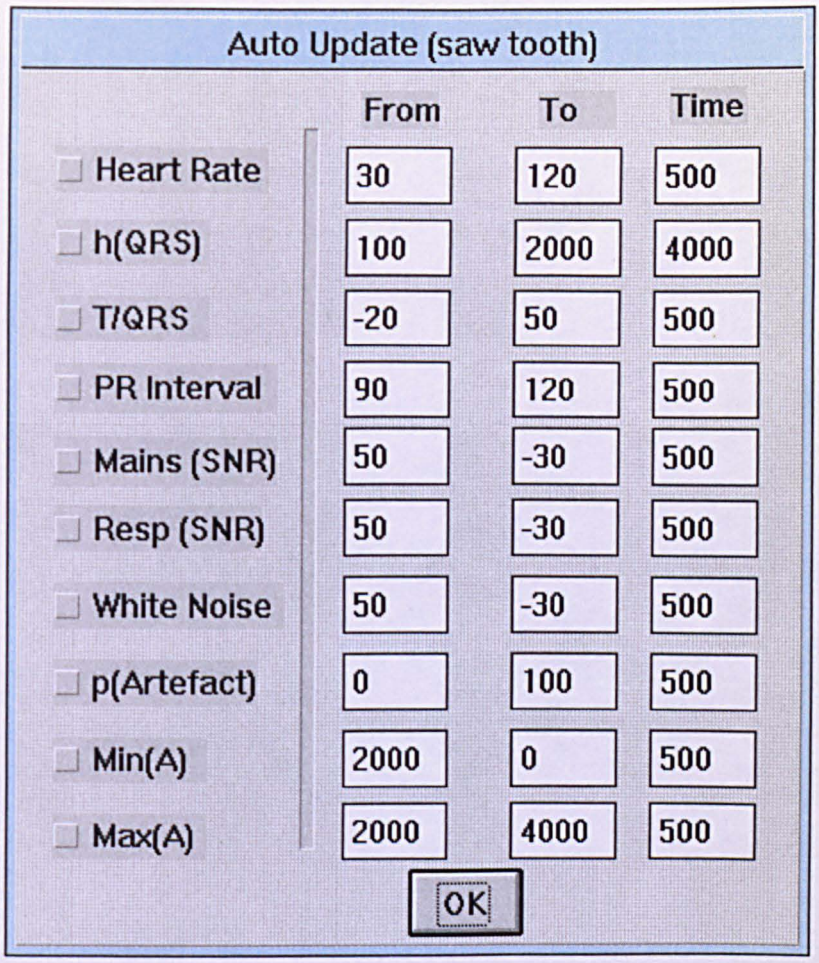


Figure III.4: *Automatic update* window

Example: The user requires the HR to vary from 100 b.p.m. to 140 b.p.m. and back to 100 b.p.m. over a 10 minute cycle. Over the same period, the PR interval is to vary from 110 ms to 90 ms, and back to 110 ms. The user must therefore select the *Heart Rate* and *PR Interval* Radio buttons and enter the following parameters:

Parameter	From	To	Time
Heart-Rate	100	140	600
PR Interval	110	90	600

Table III.3: Example for varying heart-rate and PR interval

III.2 Output window

This displays the waveform which is presently being output. The display can be in 3 forms:

Option	Display
Scroll	Scrolls the data from the right.
Slide	Displays the data in a sweeping fashion, similar to an oscilloscope.
Waveform	Displays each new waveform, triggered at the P wave onset.

Table III.4: Types of display for output window

III.3 Parameters window

This window displays a record of the ECG parameters, which have been set in the Control Window.

III.4 Information window

This window displays the total elapsed time, and the total number of ECG complexes, which have been generated since the start of simulation.

Appendix IV : Simulator software

IV.1 Development Files

Filename	Header filename	Description
SIMULATE.PRJ		Borland C++ IDE project file
SIMULATE.DSK		Borland C++ IDE desktop file
SIMULATE.CPP	SIMULATE.H	TSim class
SIMWIN.CPP	SIMWIN.H	TSimulateWindow class
	MESSAGE.H	Defines window messages
OUTWIN.CPP	OUTWIN.H	TOutputWindow class
CONWIN.CPP	CONWIN.H	TControlWindow class
	STRUCT.H	Defines structure which holds details to where the data will be output.
SETUP.CPP	SETUP.H	TSetUpDialog class
CHANGE.CPP	CHANGE.H	TECGDialog class
AUTO.CPP	AUTO.H	TAutoDialog class
TCOMM.CPP	TCOMM.H	TComm class
TDTOA.CPP	TDTOA.H	TDtoA class
TFILE.CPP	TFILE.H	TFile class
TGENERAT.CPP	TGENERAT.H	TGenerate class
	ECG.H	Defines names of waves in ECG
TFREQ.CPP	TFREQ.H	TFreq class
GAUSSIAN.CPP	GAUSSIAN.H	TGaussian class
ARTEFACT.CPP	ARTEFACT.H	TArtefact class
SAVER.CPP	SAVER.H	TSaver class
CONTIME.CPP	CONTIME.H	TConTime class
WAVEFORM.CPP	WAVEFORM.H	Waveform class: Base class for NodesOnWave and AmpChangeWave
NODES.CPP	NODES.H	NodesOnWave class
WAVEAMP.CPP	WAVEAMP.H	AmpChangeWave class
DISTORT.CPP	DISTORT.H	DistortWave class: Base class for WaveFromFile and ContWave classes
WAVEFILE.CPP	WAVEFILE.H	WaveFromFile class
CONTWAVE.CPP	CONTWAVE.H	ContWave class
PARAMWIN.CPP	PARAMWIN.H	TParametersWindow class
APLOT.CPP	APLOT.H	APlot class
INFOWIN.CPP	INFOWIN.H	TInfoWin class
SIMULATE.RC	SIMULATE.RC	Resource file
MODULE.DEF		Module definition
BWCC.LIB		Borland C++ library
BLUECHIP.LIB	BLUECHIP.H	Bluechip library

IV.2 Program run-time files

Filename	Description
BWCC.DLL	Borland dynamic link library
BLUECHIP.DLL	Bluechip dynamic link library
DATAAVG.TXT	Text file of ECG waveform

IV.3 Objects

This section lists all the objects used. It is given in a format similar to the class reference in the *ObjectWindows User's Guide* [Borland International, 1991]. Many of the properties of a class are inherited from base classes. Rather than duplicate this information, only new data members are listed. Figure IV.1 shows the class hierarchy. Classes taken from the ObjectWindows library are shown in **bold**. The structure for the classes is shown in Figure IV.2.

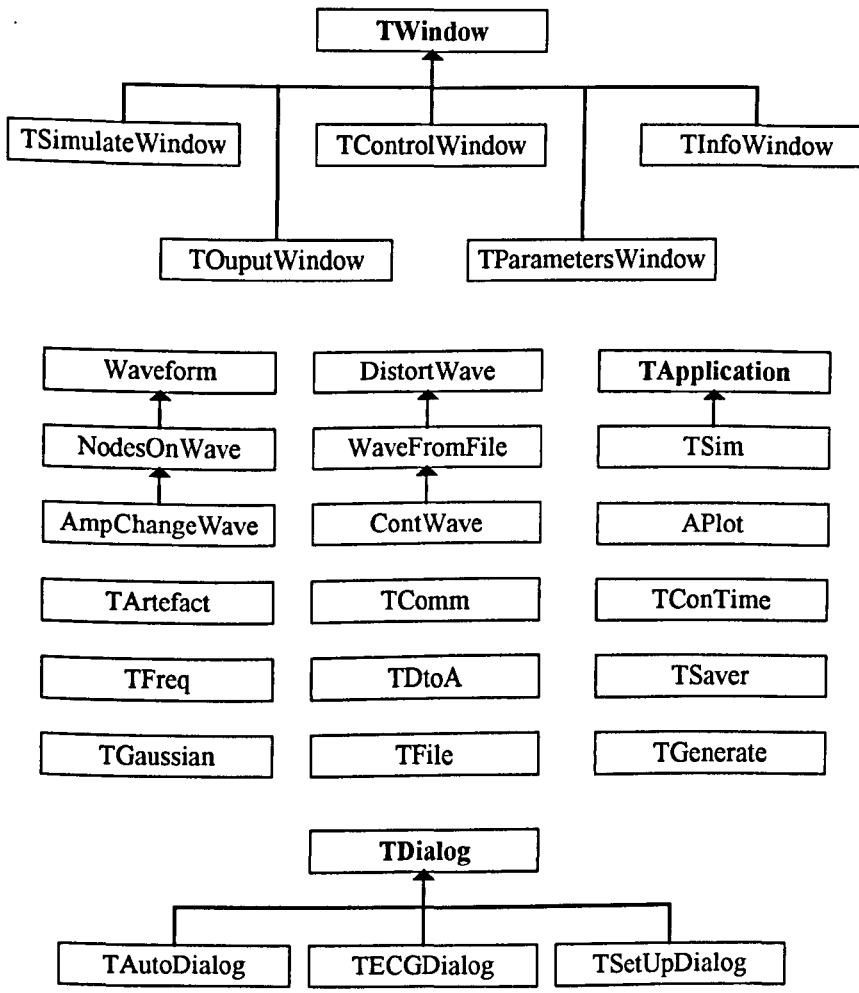


Figure IV.1: Class hierarchy

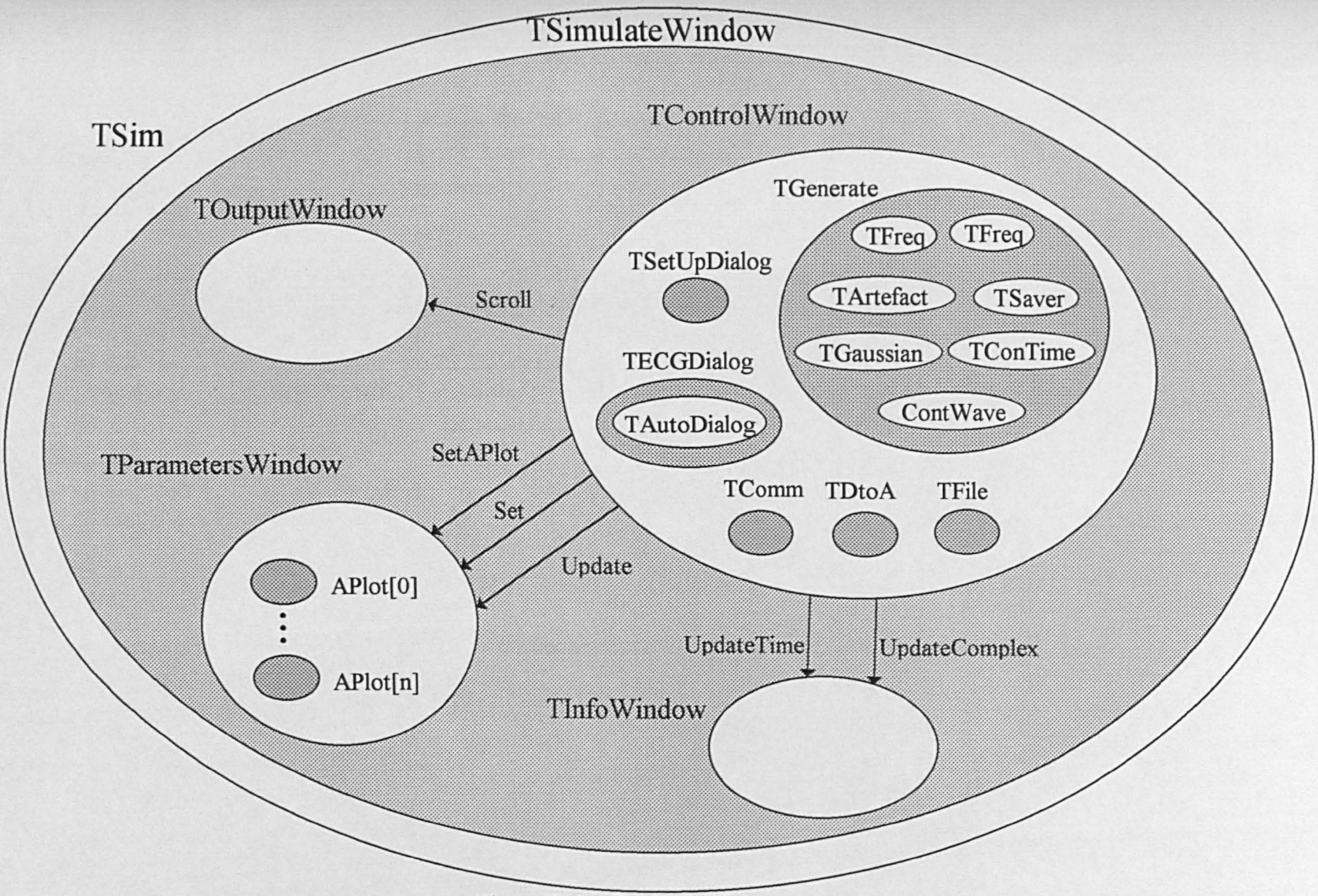


Figure IV.2: Structure of classes

AmpChangeWave — WAVEAMP.CPP

AmpChangeWave
~AmpChangeWave
SetAmp

AmpChangeWave is inherited form NodesOnWave, and allows the amplitude of a node to be changed.

Member functions

Constructor

```
AmpChangeWave(float* Samples, int MaxSamples,
                int NoOfNodes, int* Position);
```

Invokes NodesOnWave constructor, passing *Samples*, *MaxSamples*, *NoOfNodes*, and *Position*.

Destructor

```
~AmpChangeWave();
```

Empty.

SetAmp

```
SetAmp( int NTA, int NewHeight);
```

Sets the amplitude of node *NTA* to *NewData*. Also adjusts the data points from previous to next node.

APlot — APLOT.CPP

Caption
Colour
Min
Max
Value
APlot

Class for storing array of data to be displayed, including details on how it will be displayed.

Data members

Caption

```
char Caption[10];
```

Caption for data.

Colour

```
COLORREF Colour;
```

Colour of data, which will be displayed.

Max

```
float Max;
```

Maximum value, which will be displayed.

Min

```
float Min;
```

Minimum value, which will be displayed.

Value `float Value[MAX_VALUES];`

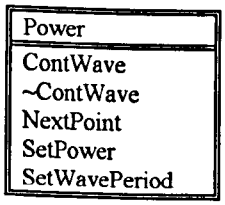
Array of values to be displayed.

Member functions

Constructor
```
APlot(float NewMin, float NewMax, char* NewCaption,
      COLORREF NewColour);
```

Sets Min to *NewMin*, Max to *NewMax*, Caption to *NewCaption*, Colour to *NewColour*, and initialises Value array.

ContWave — CONTWAVE.CPP



ContWave is inherited from WaveFromFile. It is used to give a repeated ECG waveform.

Data members

Power `float Power;`

Contains power of the ECG.

Member functions

Constructor
```
ContWave(char* filename, int MaxSamples, float Rate,
         int NoOfNodes, int* Position);
```

Invokes WaveFromFile constructor passing *filename*, *MaxSamples*, *Rate*, *NoOfNodes* and *Position*.

Destructor `~ContWave();`

Empty.

NextPoint `int NextPoint(float* value);`

Stores NewWave→Data in value. Once the data is exhausted, a linear model is calculated from the last Data point, to the first Data point, which will be output once the wave period is reached. Returns TRUE if next data point will be start of a new ECG waveform.

Reset `void Reset();`

Must be called before data is output, so that as soon as data is being generated, an ECG waveform is commenced.

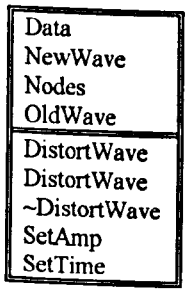
SetPower `void SetPower();`

SetPower calculates the power of the wave NewWave and stores the result in Power.

SetWavePeriod `void SetWavePeriod(long double newdelay);`

Sets wave period (time between start of ECG waves) to *newdelay*.

DistortWave — DISTORT.CPP



This is not a windows interface object class, but an abstract class, allowing a waveform to be stretched in both the amplitude and time directions.

Data members

Data `float* Data;` **(Protected)**

Pointer to data, which has been both amplitude and time-stretched. This is in fact a copy of NewWave➡Data.

NewWave `AmpChangeWave *NewWave;`

Pointer to instance of AmpChangeWave, the ECG waveform, with time-stretching.

Nodes `int Nodes;` **(Protected)**

Contains the number of nodes on the waveform.

OldWave `AmpChangeWave *OldWave;`

Pointer to instance of AmpChangeWave, the ECG waveform, without time-stretching.

Member functions

Constructor `DistortWave();`

Empty.

Constructor `DistortWave(float* Samples, int MaxSamples, float Rate, int NoOfNodes, int* Position);`

Calculates the total number of data points using array *Position*. Creates OldWave, an instance of AmpChangeWave, passing *Samples*, number of data points, *NoOfNodes*, and *Position*. Creates NewWave, an instance of SNWave passing *Samples*, *MaxSamples*, *NoOfNodes*, and *Position*. Copies *NoOfNodes* to Nodes. Sets Data to NewWave→Data.

Destructor `~DistortWave();`

Deletes object OldWave and NewWave.

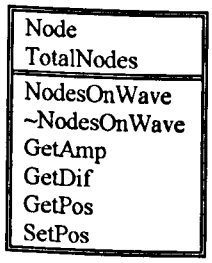
SetAmp `void SetAmp(int NTA, float Amplitude);`

Sets the amplitude of node *NTA* of both wave objects, by calling OldWave→SetAmp and NewWave→SetAmp, passing *NTA* and *Amplitude*.

SetTime `void SetTime(int N1, int N2, float TrueTime);`

Sets the time between node *N1* and *N2* of NewWave to *TrueTime*. This is done by interpolation from OldWave.

NodesOnWave — NODES.CPP



NodesOnWave is inherited from waveform. It controls a waveform with nodes positioned upon it.

Data members

Node `PTNODE Node;` **(Protected)**

Pointer to an array of NODEs (a structure containing the position and amplitude of each node).

TotalNodes `int TotalNodes;` **(Protected)**

Contains number of nodes on waveform.

Member functions

Constructor `NodesOnWave(float* Samples, int MaxSamples, int NoOfNodes, int* Position);`

Invokes Waveform constructor, passing *Samples*, number of data points (*NoOfNodes* th member in array *Position*), *MaxSamples*. Allocates array for nodes, and calls SetPos to set node positions to those given in array *Position*.

Destructor `~NodesOnWave();`

Frees up memory for Node array.

GetAmp `float GetAmp(int No);`

Returns the amplitude of a node *No.*

GetDif `float GetDif(int No1, int No2);`

Returns the difference in position of nodes, *No1* and *No2.*

GetPos `float GetPos(int No);`

Returns the position of node *No.*

SetPos `void SetPos(int No, float Position);`

Sets the position of node *No* (and thus amplitude) to *Position.*

TArtefact — ARTEFACT.CPP



TArtefact, was written to add baseline shifts to data. The user could set the probability of such an artefact occuring every second, and the limits to the baseline to be added.

Member functions

Constructor `TArtefact(int freq, int DC);`

The constructor should be passed to the output data frequency, *freq,* and the initial baseline level, *DC.*

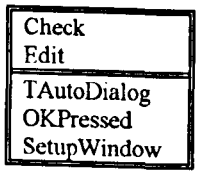
SetProb `void SetProb(int NewProb, int NewMin, int NewMax);`

This function should be called to set the probability that a shift will occur, *NewProb* %, and the minumum and maximum DC level, *NewMin* and *NewMax.*

Add `void Add(float* data);`

Adds DC level to the float pointed to by *data.*

TAutoDialog — AUTO.CPP



TAutoDialog is inherited from TDailog. It is a dialog box to control the automatic update of an instance of the TECGDialog box.

Data members

Check `PTCheckBox Check[NO_OF_BARS];`

Pointer to instance of TInfoWindow.

Edit `PTStatic Edit[NO_OF_BARS][3];`

Pointer to instance of TParamWindow.

Member functions

Constructor `TAutoDialog(PTWindowsObject AParent, LPSTR AName);`

Invokes TDialog constructor, passing *AParent* and *AName*. Creates instances of TCheckBox. and TStatic.

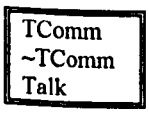
OKPressed `void OKPressed(RTMessage) = [ID_FIRST+ID_OKPRESS];`

Hides window.

SetupWindow `virtual void SetupWindow();`

Calls TDialog::SetupWindow. and initalises text in edit boxes.

TComm — TCOMM.CPP



TComm handles the communications across the serial ports for output of the simulated data.

Member functions

Constructor `TComm(char* port);`

Sets up communications on the RS232 communication port, specified in *port*. A baud rate of 19200 is used, with no parity, 8 data bits, and 1 stop bit.

Destructor `~TComm();`

Closes communications on the relevant communications port.

Talk `void Talk(int magnitude);`

Sends 2 bytes of data, specified in *magnitude*.

TConTime — CONTIME.CPP

TConTime
Increase

This class acts an alarm clock, which rings every second. The timer is increased by calling the Increase function with the time interval. With every second that elapses, a message is sent to the instance of TInfoWindow to update the time, and to TSaver to save various parameters.

Member functions

Constructor `TConTime(HWND HInf, TSaver* NewSaver);`

The constructor should be passed the Windows handle to the instance of TInfoWindow, and a pointer to TSaver. It then initialses the timer.

Increase `BOOL Increase(float interval);`

Updates the timer, by interval seconds. If another second has elapsed the following occurs: a Windows message UPDATET is sent to hInfWin to update the time; the Save function to the instance of TSaver is called to save various parameters; and the function returns TRUE.

TControlWindow — CONWIN.CPP

Data	SetUpStruct
TControlWindow	Paint
~TControlWindow	Setup
CanClose	Start
Change	Stop
Out	

TControlWindow controls where the simulated data is output.

Data members

Data

```
float Data[MAX_OUTPUT_BLOCK_SIZE];
```

Array to hold the ECG data points.

SetUpStruct

```
TSetUpStruct SetUpStruct;
```

Pointer to instance of TSetUpStruct, containing information to where the data is to be output.

Member functions

Constructor

```
TControlWindow(PTWindowsObject AParent, LPSTR ATitle);
```

Invokes TWindow constructor, passing *AParent* and *ATitle*. Sets windows attributes and creates instances of **TSetUpDialog**, **TECGDialog** and **TGenerate**. Also creates buttons to start and stop data output, and to choose child windows.

Destructor

```
~TControlWindow();
```

Stops output, and frees up memory used by instance of TGenerate.

Start

```
void Start() = [ID_FIRST + START];
```

Responds to a START message, by firstly creating the output object (**TFile**, **TComm** or **TDtoA**), depending on selection. Brings stop button to front and sets up timed interrupt.

CanClose

```
virtual BOOL CanClose();
```

Returns TRUE.

Out

```
void Out() = [WM_FIRST + WM_TIMER];
```

Responds to a WM_TIMER message, by outputting data. It outputs the data points, and sends message, SCROLL, to OutputWindow to redraw output. If new complex has occurred, an Update complex message, UPDATEC, is sent to InfoWindow.

Setup

```
void Setup() = [ID_FIRST + SETUP];
```

Responds to a SETUP message, by showing SetUpWindow and ChangeButton.

Stop

```
void Stop() = [ID_FIRST + STOP];
```

Responds to a STOP message, by stopping timed interrupt, deleting output object and bringing start button to front.

Paint

```
void Paint( HDC PaintDC, PAINTSTRUCT _FAR & );
```

Draws window. Uses *PaintDC* as the paint display context.

Change

```
void Change() = [ID_FIRST + PARAM];
```

Responds to a SCROLL message, by showing ChangeWindow and SetUpButton.

TDtoA — TDTOA.CPP

TDtoA
~ TDtoA
Talk

TDtoA handles the data output on the D/A card.

Member functions

Constructor `TDtoA(int Frequency);`

Initialises the BlueChip technology ADC-42 D/A board at an output rate, specified by *Frequency*.

Destructor `~TDtoA();`

Releases the D/A board.

Talk `int Talk(int Mag);`

Outputs the value specified in *Mag* to the D/A board and returns the status of the card.

TECGDialog — CHANGE.CPP

Bar	
TECGDialog	HREd
Advanced	HRPressed
Auto	HZEd
Reset	HZPressed
SetupWindow	PREd
BDEd	PRPressed
BDPressed	RHEd
BMnEd	RHPressed
BMnPressed	THEd
BMxEd	THPressed
BMxPressed	WNEd
BSEd	WNPressed
BSPressed	

This class is a dialog object, allowing the user to change various ECG parameters, either in real-time, or automatically.

Data members

Bar `PTScrollBar Bar[NO_OF_BARS];`

Array of pointers to scroll bars.

Member functions

Constructor

```
TECGDialog(PTWindowsObject AParent, LPSTR AName);
```

Invokes TDialog constructor, passing *AParent* and *AName*. Creates several instances of TScrollBar and TStatic, and instance of **TAutoDialog**.

HREd
RHEd
THEd
PREd
HZEd
BSEd
WNEd
BDEd
BMnEd
BMxEd

```
void HREd(RTMessage) = [ID_FIRST+ID_CHANGE+10];
void RHEd(RTMessage) = [ID_FIRST+ID_CHANGE+11];
void THEd(RTMessage) = [ID_FIRST+ID_CHANGE+12];
void PREd(RTMessage) = [ID_FIRST+ID_CHANGE+13];
void HZEd(RTMessage) = [ID_FIRST+ID_CHANGE+14];
void BSEd(RTMessage) = [ID_FIRST+ID_CHANGE+15];
void WNEd(RTMessage) = [ID_FIRST+ID_CHANGE+16];
void BDEd(RTMessage) = [ID_FIRST+ID_CHANGE+17];
void BMnEd(RTMessage)= [ID_FIRST+ID_CHANGE+18];
void BMxEd(RTMessage)= [ID_FIRST+ID_CHANGE+19];
```

Responds to message, by moving the relevant scrollbar, when the corresponding static window is changed.

HRPressed
RHPressed
THPressed
PRPressed
HZPressed
BSPressed
WNPressed
BDPressed
BMnPressed
BMxPressed

```
void HRPressed(RTMessage) = [ID_FIRST+ID_CHANGE  ];
void RHPressed(RTMessage) = [ID_FIRST+ID_CHANGE+1];
void THPressed(RTMessage) = [ID_FIRST+ID_CHANGE+2];
void PRPressed(RTMessage) = [ID_FIRST+ID_CHANGE+3];
void HZPressed(RTMessage) = [ID_FIRST+ID_CHANGE+4];
void BSPressed(RTMessage) = [ID_FIRST+ID_CHANGE+5];
void WNPressed(RTMessage) = [ID_FIRST+ID_CHANGE+6];
void BDPressed(RTMessage) = [ID_FIRST+ID_CHANGE+7];
void BMnPressed(RTMessage)= [ID_FIRST+ID_CHANGE+8];
void BMxPressed(RTMessage)= [ID_FIRST+ID_CHANGE+9];
```

Responds to message, by changing the relevant static window, when the corresponding scroll bar is moved.

Advanced

```
void Advanced(RTMessage) = [ID_FIRST+ID_CHANGE+20];
```

Responds to message, by displaying instance of AutoWindow.

Auto

```
void Auto();
```

Automatically updates bars and edit window depending on AutoWindow settings.

Reset

```
void Reset();
```

Must be called to reset internal variables, when data has started to be output.

SetupWindow

```
virtual void SetupWindow();
```

Calls TDialog::SetupWindow, then initialises bar positions.

TFile — TFILE.CPP



TFile handles the data output into a file.

Member functions

Constructor

```
TFile(char* filename, BOOL bHyperSignal,
      BOOL isinteger);
```

Opens a file specified in *filename*. If *bHyperSignal* is TRUE, a 20 byte dummy header is inserted. *isinteger* should be set to TRUE, if the file is going to contain integer values; otherwise floats will be stored.

Destructor

```
~TFile();
```

Closes the output file.

Talk

```
void Talk(float mag);
```

Outputs *mag* to the file, as either a float or an integer (this should have been set in the constructor).

TFreq — TFREQ.CPP



TFreq, was written to add fixed frequency noise (such as mains interference and baseline wander) to data. The user can set the signal to noise ratio for this noise.

Member functions

Constructor

```
TFreq(int Outfreq, float Noisefreq,
      PTScrollBar Adjust);
```

The constructor should be sent the sample frequency of the output data, *Outfreq*, the frequency of the sinusoidal noise to be generated, *Noisefreq*, and the pointer to the scroll bar, which will control the SNR of the noise, *Adjust*.

AddNoise

```
void AddNoise(float* value, float SigPower);
```

A pointer to the current data value should be passed to *value*. From the signal power, *SigPower*, and the position of the relevant scroll power, noise is added to this data value.

TGaussian — GAUSSIAN.CPP



This class may be used to add white noise to a signal. The user can set the signal to noise ratio for this noise.

Member functions

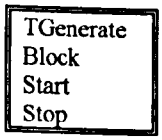
Constructor `TGaussian(PTScrollBar Adjust);`

The constructor should be passed the pointer to the scroll bar which will control the SNR of the noise, *Adjust*.

AddNoise `void AddNoise(float* value, float SigPower);`

A pointer to the current data value should be passed to *value*. From the signal power, *SigPower*, and the position of the relevant scroll power, noise is added to this data value.

TGenerate — TGENERAT.CPP



Class to generate ECG data plus noise.

Member functions

Constructor `TGenerate(TECGDialog* Win);`

The pointer to the instance of TECGDialog, the dialog box for controlling the morphology of the ECG and the noise, should be passed in *Win*. An instance of **ContWave** is created, with filename *DATAAVG.TXT*.

Destructor `~TGenerate();`

Frees up memory used by instances of ContWave.

Block `int Block(float* Data, int samples);`

Fills array, pointed to by *Data*, with the next *samples* number of samples. It returns position in the array of new complex (if one has started).

Start
```
void Start(int SampleFreq, TParametersWindow* param,
           HWND InfoH);
```

This should be passed the data output frequency, *SampleFreq*, the pointer to the instance of TParametersWindow, *param*, and the Windows handle for the instance of TInfoWindow, *InfoH*. Creates instance of **TSaver**, **TGaussian**, **TArtefact**, **TConTime** and two instances of **TFreq** (one for the mains interference at 50Hz and one for the baseline wander at 0.3 Hz).

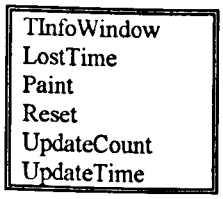
Stop `void Stop();`

Deletes instances of TSaver, TGaussian, TArtefact, TConTime and TFreq.

Update `void Update();`

Update the morphology of ECG wave.

TInfoWindow — INFOWIN.CPP



This windows object displays the elapsed time and number of complexes.

Member functions

Constructor `TInfoWindow(PTWindowsObject AParent, LPSTR ATitle);`

Invokes TWindow constructor, passing *AParent* and *ATitle*, and sets window attributes.

Reset `void Reset() = [WM_USER + RESET];`

Responds to a RESET message, by resetting time and counters for the hour, the minute, the second and the complex count.

LostTime `BOOL LostTime();`

Timer used in ControlWindow may not be accurate, but time() function is. This function checks for agreement, and returns TRUE if there is a discrepancy.

UpdateTime `void UpdateTime() = [WM_USER + UPDATET];`

Responds to a message, by updating time displayed by one second.

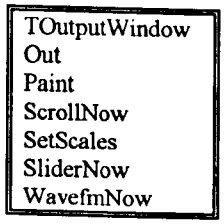
UpdateCount `void UpdateCount() = [WM_USER + UPDATEC];`

Responds to a UPDATEC message, by updating complex count displayed by one.

Paint `void Paint( HDC PaintDC, PAINTSTRUCT _FAR &);`

Draws text in window. Uses *PaintDC* as the paint display context.

TOutputWindow — OUTWIN.CPP



TOutputWindow is derived from TWindow. This windows object displays the current output data in either of the following forms:

1. Scroll mode - Scrolls data across window
2. Slider mode - Sweeps over window (like oscilloscope)
3. Waveform mode - Displays each waveform from P-wave onset

Member functions

Constructor `TOutputWindow(PTWindowsObject AParent, LPSTR ATitle);`

Invokes TWindow constructor, passing *AParent* and *ATitle*. Creates buttons to select display mode.

Out `void Out(RTMessage Msg) = [ WM_USER + SCROLL];`

Responds to a SCROLL message, by displaying data in window, in the correct mode of display. The data used is stored in the child window, ControlWindow, belonging to this object's parent window. It is scaled to the window, depending on the voltages passed in SetScales.

Paint `void Paint( HDC PaintDC, PAINTSTRUCT _FAR & );`

Draws window in which to display ECG data. Uses *PaintDC* as the paint display context.

ScrollNow `void ScrollNow() = [ ID_FIRST + B_SCROLL];`

Responds to a B_SCROLL message, by changing mode selection to scroll mode.

SetScales `void SetScales(float NewMin, float NewMax);`

This window should be passed both the minimum and maximum possible data voltages, *NewMin* and *NewMax*, so that the data can be scaled. These scales are drawn onto the window.

SliderNow `void SliderNow() = [ ID_FIRST + B_SLIDER];`

Responds to a B_SLIDER message, by changing mode selection to slider mode.

WavefmNow `void WavefmNow() = [ ID_FIRST + B_WAVEFM];`

Responds to a B_WAVEFM message, by changing mode selection to waveform mode.

TParametersWindow — PARAMWIN.CPP

TParametersWindow
~TParametersWindow
Paint
Set
SetAPlot
Update

TParametersWindow plots selected parameters in a window.

Member functions

Constructor
```
TParametersWindow(PTWindowsObject AParent,
                  LPSTR ATitle);
```

Invokes TWindow constructor, passing *AParent* and *ATitle* and sets windows attributes.

Destructor
```
~TParametersWindow();
```

Frees up memory used by instances of APlot.

Paint
```
void Paint( HDC PaintDC, PAINTSTRUCT _FAR & );
```

Draws window for displaying parameters, labels axes, and replots parameters. Uses *PaintDC* as the paint display context.

Set
```
void Set(int,float);
```

Returns TRUE.

SetAPlot
```
void SetAPlot(float Min, float Max, char* ThisCaption,
              COLORREF Colour);
```

Creates an instance of **APlot**, passing *Min*, *Max*, *ThisCaption*, and *Colour*.

Update
```
void Update();
```

Scrolls window and adds plots latest parameter values.

TSaver — SAVER.CPP

TSaver
~TSaver
Save

This class saves ECG parameters passed to it. The format for the saved data is 100 byte blocks, containing 25 floats, and has the same format as the analysis program, except no *Conduction Index* value is calculated.

Member functions

Constructor `TSaver();`

Opens parameter output file, with filename *C:\SIMULATE.PAR*.

Destructor `~TSaver();`

Closes output file.

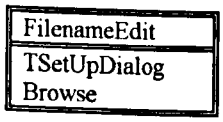
NextWave `void NextWave(float newHR, float newPR, float newTQRS);`

Should be called with each new waveform, detailing the HR, *newHR*, the PR interval, *newPR*, and the T/QRS ratio, *newTQRS* %.

Save `void Save();`

Needs to be called twice before it saves heart rate, PR intervals and T/QRS intervals to disk. Thus, it should be called, every second, and will save data every 2 seconds.

TSetUpDialog — SETUP.CPP



TSetUpDialog is derived from TDialog. It is dialog box which allows the user to set up where the data is to be output.

Data members

FilenameEdit `PTEdit FilenameEdit;`

Pointer to instance of TEdit, containing the output filename.

Member functions

Constructor `TSetUpDialog (PTWindowsObject AParent, LPSTR AName);`

Invokes TDialog constructor, passing *AParent* and *AName*. Creates variuos radio buttons, edit boxes, a combination box and a browse button.

Browse `void Browse() = [ID_FIRST + ID_BROWSE];`

Responds to a ID_BROWSE, by executing an instance of TFileDialog to allow user to determine output filename.

TSim — SIMULATE.CPP



TSim is derived from TApplicationWindow. This is the application class.

Member functions

Constructor

```
TSim(LPSTR name, HINSTANCE hInstance,
     INSTANCE hPrevInstance, LPSTR lpCmd,
     int nCmdShow);
```

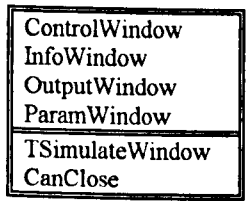
Invokes TApplication constructor, passing *name*, *hInstance* and *hPrevInstance*, *lpCmd* and *nCmdShow*.

InitMainWindow

```
virtual void InitMainWindow();
```

Creates instance of **TSimulateWindow**.

TSimulateWindow — SIMWIN.CPP



TSimulateWindow is derived from TWindow. It is the main display window, containg the four child windows, instances of TInfoWindow, TParamWindow, TControlWindow, and TOutputWindow.

Data members

ControlWindow

```
PTWindow ControlWindow;
```

Pointer to instance of TControlWindow.

InfoWindow

```
PTWindow InfoWindow;
```

Pointer to instance of TInfoWindow.

OutputWindow

```
PTWindow OutputWindow;
```

Pointer to instance of TOutputWindow.

ParamWindow

```
PTWindow ParamWindow;
```

Pointer to instance of TParamWindow.

Member functions

Constructor

```
TSimulateWindow(PTWindowsObject AParent,
                LPSTR ATitle);
```

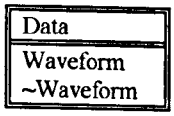
Invokes TWindow constructor passing *AParent* and *ATitle*. Sets window attributes and creates instances of four child windows, namely instances of **TInfoWindow**, **TParamWindow**, **TControlWindow** & **TOutputWindow**.

CanClose

```
virtual BOOL CanClose();
```

Returns TRUE.

Waveform — WAVEFORM.CPP



This is not a windows interface object class, but an abstract class, for storing any Waveform.

Data members

Data `float* Data;`

Pointer to ECG data.

Member functions

Constructor
```
Waveform(float *Samples, int NoOfSamples,
         int MaxSamples);
```

Allocates Data array of size *MaxSamples*, and copies *NoOfSamples* floats from array *Samples* into Data.

Destructor `~Waveform();`

Frees up memory for Data array.

WaveFromFile — WAVEFILE.CPP



WaveFromFile is inherited from DistortWave. It allows a filename to be passed to the constructor, from which the samples are read in.

Member functions

Constructor
```
WaveFromFile(char* filename, int MaxSamples,
             float Rate, int NoOfNodes, int* Position);
```

Reads in ECG data from file, specified in *filename*. Invokes DistortWave constructor passing this data, *MaxSamples*, *Rate*, *NoOfNodes* and *Position*.

Destructor `~WaveFromFile();`

Empty.

Appendix V : Simulator algorithms

The following details the algorithm to adjust the amplitude of component ECG waves: Consider the curve (see Figure V.1) between 3 nodes P_1, P_2 and P_3 on the averaged waveform. P_1, P_2 and P_3 have co-ordinates (x_1,y_1), (x_2,y_2) and (x_3,y_3) respectively. To stretch the curve such that P_2 moves to $P_2^{|}$, gives a scale factor,α.

$$\alpha = \frac{h^{|}}{h} \text{ where } \begin{cases} h^{|} = y_2^{|} - y_c \\ h = y_2 - y_c \end{cases}$$

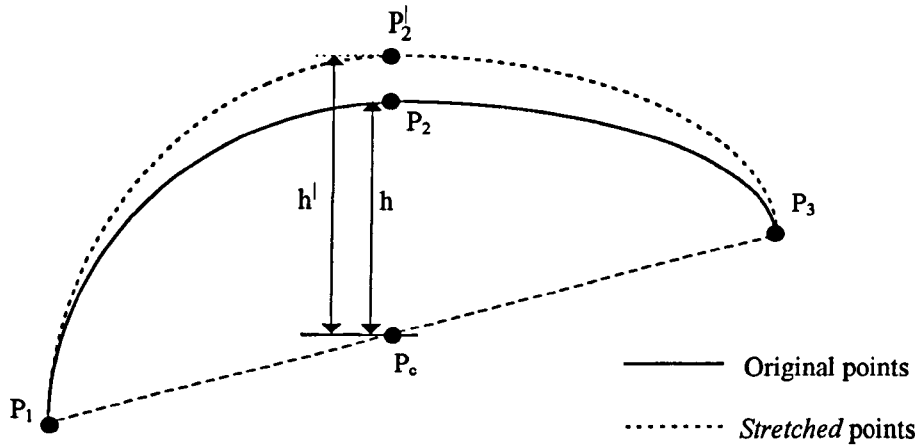


Figure V.1: Setting node amplitude

All points on the curve need to be *stretched* by the same scale factor from the baseline P_1P_3. Consider the general point on the curve $P_a(x_a,y_a)$, with base $P_b(x_b,y_b)$. This is shown in Figure V.2.

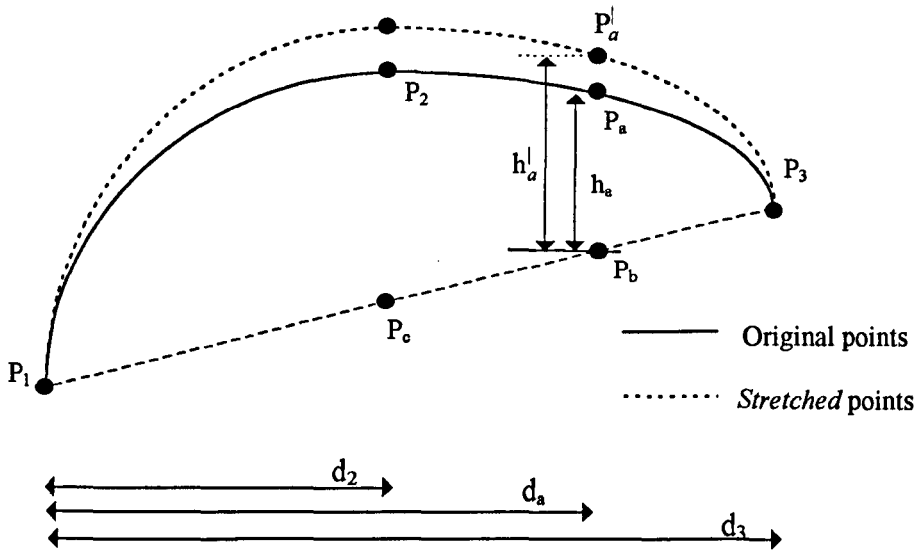


Figure V.2: Adjusting wave amplitude

Clearly $x_b = x_a$

Let $\begin{cases} d_2 = x_2 - x_1 \\ d_3 = x_3 - x_1 \\ d_a = x_a - x_1 \end{cases}$

Now, $\begin{cases} y_b = y_1 + \dfrac{d_a}{d_3}(y_3 - y_1) \\ y_c = y_1 + \dfrac{d_2}{d_3}(y_3 - y_1) \end{cases}$

To stretch the general point P_A, we have:

$$\alpha = \frac{h'_a}{h_a} = \frac{y'_a - y_b}{y_a - y_b}$$

$$\Rightarrow y'_a = \alpha(y_a - y_b) + y_b$$

Thus we have the following algorithm:

Scale factor calculation Given (x_1,y_1), (x_2,y_2), (x_3,y_3) moving to (x_1,y_1), (x_2,y_2'), (x_3,y_3)	$d_2 = x_2 - x_1$ $d_3 = x_3 - x_1$ $y_c = y_1 + \frac{d_2}{d_3}(y_3 - y_1)$ $h' = y_2' - y_c$ $h = y_2 - y_c$ $\alpha = \frac{h'}{h}$
New point calculation Any point (x_a,y_a) on the curve will move to (x_a, y_a').	$d_a = x_a - x_1$ $y_b = y_1 + \frac{d_a}{d_3}(y_3 - y_1)$ $\Rightarrow y_a' = \alpha(y_a - y_b) + y_b$