Delft University of Technology

BAP

EE3L11

Thesis Pre-Processing ECG- & Respiratory Signals for Detection of Stress

Authors:
Enes Kinaci (4370759)
Talha Kuruoglu (4718569)

June 19, 2020

Abstract

The main purpose of this thesis is the removal of different kinds of artifacts from incoming signals and the identification of relevant information which can be utilized for further analysis. This thesis proposes two designs which are used for the pre-processing of the electrocardiogram (ECG) signal and the respiratory signal. The ECG signal system design consists of an artifact removal system, a three-step quality check at the initial stage and after pre-processing the raw signal. The respiratory signal system consists of a two-step quality check, a artifact removal part and a part which calculates the respiratory rate from the respiratory signal.

Preface

This report was written in the context of the Bachelor Graduation Project to obtain the Electrical Engineering Bachelor at Delft University of Technology. We would like to thank dr. Carolina Varon Perez for her continuous help and support throughout the project. We also want to express our sincere gratitude to both dr. Ioan Lager and dr. Carolina Varon Perez for giving us the opportunity to continue the project amid the Covid-19 situation. We would also like to thank dr. Francesco Fioranelli for taking the time to be on the jury for our final assessment.

We would also like to thank our other group members, Yavuzhan, Geert Jan, Isar and Bob, whom have worked very hard together with us. Without their contributions, this would not have been possible. We had daily meetings with the group, which was divided into three subgroups, and biweekly meetings with Carolina. Their insight has extremely contributed to our progress throughout this project.

- *Enes Kinaci*
- *Talha Kuruoglu*

Contents

Chapter 1

Introduction

As the demand increases to monitor stress throughout the day, more research is conducted to find a way to continuously detect stress. To monitor stress throughout the day of ambulatory patients, a telehealth system which makes use of wearables is designed. This non-obstructive device will be able to record, process and detect stress from the ECG and respiratory signals. The detection of stress goes automatically based on machine learning. After processing all the data, the information should be accessible remotely for the patient in an environment where the privacy of each patient is safeguarded.

1.1 Problem Definition

The project is divided into three parts, where two students worked on each part. The first part is the pre-processing part of incoming raw data from the wearable. On this data, a quality assessment should be done and, if necessary, the signal must be filtered from noises and artifacts. The second part is the stress detection part with machine learning, where stress will be detected by extracting certain features from the processed ECG and respiratory signal, which is done in the first part [1].

The third part is the overall system design, where the signal processing system and stress detection system are integrated into a graphical user interface (GUI), where relevant information for the user can be displayed, such as the heart rate and, more significantly, whether the user is stressed [2].

This thesis focuses on the first part of the project. The problem definition of the pre-processing is mainly divided into two subjects: filtering of the raw data and a quality assessment of the data. This thesis presents how the signal will be processed while entering, how the quality assessment of the signal is done, and how this signal is being filtered. The processed signal shall be used for further calculations and determinations by the stress detection group. The system design group will use the information from the pre-processing for visual display to the user.

1.2 State of the Art Analysis

Many research is done on processing ECG signals, analyzing the Heart Rate Variability (HRV) and the effect of stress on it [3][4][5]. By looking at sympathetic and parasympathetic activities of the body, stress can be determined. To quantify these activities, spectral analysis is conducted on HRV. Therefore, it is of the utmost importance that the ECG signal that is analysed for the determination of HRV. Techniques are developed to detect and remove artifacts from the ECG signal to obtain a reliable signal [6],[7]. A quality assessment is done on the filtered signal, to maintain the accuracy of the overall system to detect stress. Throughout the years, many methods are developed to assess and indicate the signal quality. Some methods are described in [8],[9],[10] and [11]. While there is much knowledge about processing ECG signals, there is still no consensus on how to interpret the respiratory signals

1.3 Document Structure

This works is divided into two parts. First the system design for the ECG signal is briefly described; explaining the Finite State Machine (FSM) developed for this system (section 4.1), followed by a brief explanation about the Signal Quality Indicator (SQI) system (section 4.2) and the filtering system of the ECG signals in section 4.3. After this, the system developed for the Respiration signals is described, including the FSM for Respiratory system (section 5.1), the filtering system (section 5.2), the system for the calculation of the respiratory rate (section 5.3) and the SQI system (section 5.4). After this, the obtained results are discussed in chapter 6.

Chapter 2

Program of Requirements

This section discusses the requirements which need to be met for the pre-processing part. The main purpose of pre-processing is the removal of different kinds of artifacts from the entering signals and the identification of relevant information which can be used for further analysis. The data at the output, after pre-processing, has a Signal Quality Index (SQI) assigned to it after a quality check is performed on the incoming signal. The SQI indicates whether the incoming signal has a good or bad quality.

2.1 Functional Requirements

The following are the requirements that guided the execution of this work:

- Artifacts in the ECG and the respiratory signal must be removed.
- This system must be able to recognize the quality of the signal, either good or bad.
- The ECG and the respiratory signal both need to be labeled with a quality indicator.

2.2 Non-Functional Requirements

The following requirements elaborate the qualities or attributes which the pre-processing design must have.

- The overall system has to be suitable for real-time measurement.
- The respiratory rate needs to be obtained from the respiratory signal.
- The R-peaks of the ECG signal should be detected and sent as an output.
- The calculation of the heartbeat should be sent as an output.
- The system should work with all kinds of ECG and respiratory signal measurements which could have different sampling frequencies.
- The ECG system should accurately detect the R-peaks of the ECG signal.
- The Respiratory rate should cover the band of 0.1 Hz to 0.5 Hz.
- The delays induced in the respiratory and ECG due to filtering should be removed.

Chapter 3

Data Sets

Different datasets are used for the design of both the ECG system and the respiratory signal system. The sections below will briefly introduce the datasets used for the system design.

3.1 Stress Dataset

The first data set used in the design is the stress data set used in paper [3]. This dataset was collected at the University of Zaragoza and the Autonomous University of Barcelona. The ECG signal was sampled at 1 kHz and the respiration at 250 Hz. The volunteers underwent a stress session, in which emotional stress was induced by means of a modified Trier Social Stress Test. The test comprises the following phases:

- Baseline (BL): For about 10 minutes the subject listens to a relaxing audio
- Story Telling (ST): The subject listens to 3 stories and is asked to remember as many details as possible
- Memory Task (MT): The subject needs to tell all details that he/she remembers from the stories of ST in front of a camera.
- Stress Anticipation (SA): The subject needs to wait 10 minutes for the results of the evaluation of the MT phase.
- Video Exposition (VE): The video recorded during MT phase is shown to the subject and another video in which the stories are told entirely correct by an actor is shown as well.
- Arithmetic Task (AT): The subject is asked to count down aloud from 1022 in steps of 13. Whenever the subject is making a mistake, he/she needs to start again from 1022. To induce stress on the subject, a 5 minute constraint is induced.

Of the 46 volunteers, 11 are excluded from the study because some of the phases were corrupted by technical artifacts.

3.2 Drivers Dataset

The second data set is obtained from [12]. The dataset is called Stress Recognition in Automobile Drivers. It is recorded from 16 healthy volunteers while driving a car in Boston, Massachusetts USA. The duration of the measurements ranged between 53 minutes and 92 minutes.In the first and last 15 minutes of the measurement period, the subjects were asked to close their eyes and relax in the car in idle.These periods are regarded as non-stressful period. Afterwards they drove through quiet and busy streets for about 25 to 60 minutes, which is considered to be stressful. The only differing measurement is the ECG 16, which does not have the last 15 minutes of relaxation.

The sampling frequency of the ECG signal is equal to 496 Hz. While the respiratory is sampled at 31 Hz.

3.3 CinC2017 Dataset

The third and last data used in this dataset where taken from the PhysioNet/Computing in Cardiology Challenge of 2017 [13]. The ECG signals in this data-set are filtered and pre labeled with a signal quality indicator in which label 1 corresponds to a clean signal and 0 to contaminated signal.The ECG data consists only of the normal rhythm and noisy class data, which consists in total of 5334 recordings which are sampled at 300 Hz.This data will be mainly be used to evaluate the performance of the proposed quality indicator system.

Chapter 4

ECG System Design

This chapter will discuss the steps taken to design the ECG system. First the finite state machine of the system will be discussed. Afterwards the quality checks will be explained and finally the filtering of the ECG will be discussed.

4.1 Finite State Machine

The implementation of the overall ECG system design is shown in a finite state machine, which can be seen in Figure 4.2. In this system the following sub-parts are implemented: a function which detects the NaNs in the incoming data, the three-step quality check ,and the filters. The system changes states depending on the signal quality indicator (SQI). If the signal has a bad quality, which is determined by the NaN check or the three-step quality check, then the SQI will be 0 and otherwise it will be 1.

The NaN check is done on an incoming segmented ECG signal. If there is a NaN or if there are NaNs present in the segmented ECG signal, then the ECG system algorithm gives an error. To prevent this, it is checked whether there are NaNs or a NaN in the ECG signal before sending the signal to the quality check and the filtering part. If there are NaNs in the system, then the signal with NaNs are send with a SQI value of 0 labeled on it. The other outputs seen in Figure 4.2 get a value of zero.

The three-step quality indicator is implemented after the NaN check and the ECG signal is sent to the indicator when the SQI of the NaN check is equal to 1. The three-step quality indicator after NaN and filtering is identical and consists of the following steps:

- Quantification of the relative power of the ECG within the band of interest
- Weight computation based on the autocorrelation function (ACF)
- Heart rate evaluation

The filters are used if one of the three quality checks at the beginning of the system gives a SQI of zero. This indicates that the signal has too much artifacts to be identified as a good qualified signal, so the signal needs processing. Further details about the filters will be given in section 4.3. After the filtering, the ECG is checked again by the quality indicators and depending on the SQI value from these checks, the signal is put out with SQI 1 or 0.

The ECG will be filtered with a bandpass filter of 0.5Hz to 40 Hz after the quality checks or possibly after filtering. The ECG signal which is from 0.5 Hz to 150 Hz will have its components removed after 40 Hz. This means that the ECG will be sent with a band of 0.5 Hz and 40 Hz. The reason for this is that the band of 0.5 Hz to 40 Hz produces a more stable signal with less baseline noise and fewer high-frequency artifacts[14] ,also the system design subgroup is not performing R-peak detection which needs frequency information of at least 150 Hz.The ECG signal can thus be restricted to 40 Hz for the output.The ECG which spans from 0.5 Hz to 150 Hz is needed within the ECG system, to perform R-peak detection.The R-peak of an ECG can be seen in Figure 4.1. The frequencies of 40 Hz to 150 Hz contain high-frequency components which are needed for the R-peak detection.

Each ECG segment sent as output is characterized by:

- A filtered version
- The average heart rate
- SQI
- The position of the R-peaks in the signal
- Time differences between the R-peaks

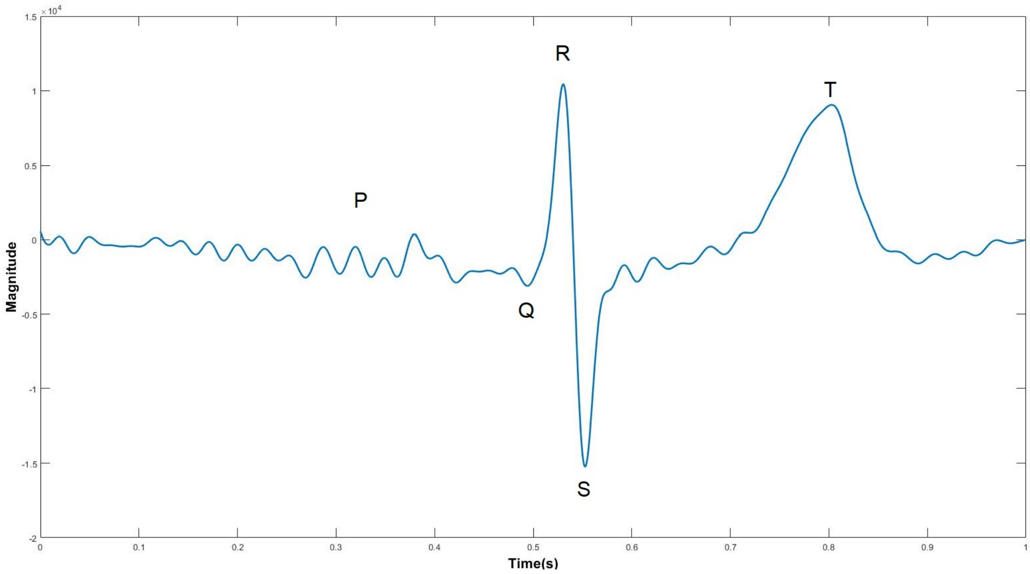

Figure 4.1: The ECG signal with its P wave, QRS complex and T-wave.

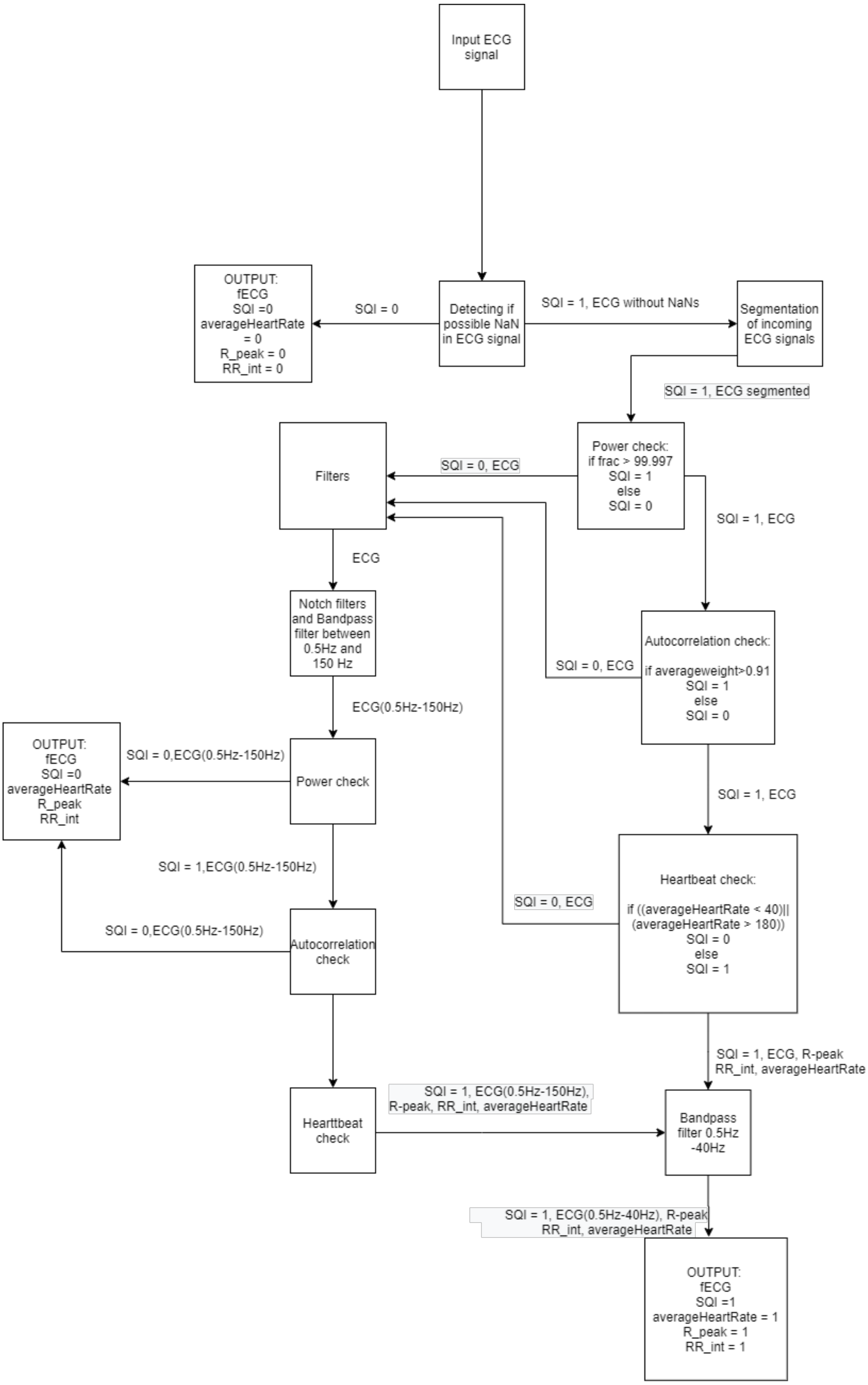

Figure 4.2: The finite state machine of the ECG system. Not every signal is shown between transitions to maintain simplicity of the diagram

4.2 Signal Quality Indicator of the ECG Signal

The Signal Quality indicator (SQI) section will discuss the quality assessment of the incoming signals. The output of the signal quality indicator is a single bit, which is used by the stress detection part when making a decision whether the a specific part of the signal will be used in further calculations or not.

The quality assessment is executed based on a three step decision making. At each step, the SQI is checked in a different way, using concepts such as power spectral density distribution, autocorrelation function and the heart rate.

4.2.1 Spectral Distribution Ratio of the ECG

The first step in the quality check procedure is looking at the power spectral distribution ratio. The ratio of the power between the spectrum 0.5 Hz and 150 Hz is compared to the whole power spectrum of segmented signal. A threshold is assigned to the ratio of the usable power. If the ratio is above this threshold, then the segment is labeled as a reliable segment in terms of power which can be used in further calculations. Otherwise, the segment is labeled as not reliable and will passed to the filter to remove excessive noise. The threshold is chosen by taking the mean of the average distribution ratio of the whole signal. Then, this value is fine tuned by comparing the results of the SQI's with the annotations from CinC2017 dataset. The threshold value is described in chapter 6. In appendix A, an example of two PSD's are given for more illustration.

Before, a cutoff of 100 Hz was considered adequate by the American Heart Association (AHA) to maintain the accuracy for diagnostics during visual inspection[15]. But higher-frequency components could also contain information from the QRS-complex, in particular the R-peak[14]. So following the recommendation of AHA [15], a range between the cutoff frequencies 0.5 Hz and 150 Hz is used for diagnostic purposes. For only monitoring purposes of the ECG, a bandwith between 0.5 Hz and 40 Hz can be taken, since it reduces a lot of noises but also removes high-frequency components. So, for visualizations of the ECG by the system design group [2], a processed signal between the bandwidths 0.5 Hz and 40 Hz will be sufficient.

Welch's method is used to calculate the power spectral density (PSD) of the segment. For the computation PSD of an entire waveform, the Fast Fourier Transform could be used (FFT). To enhance the statistical properties of the result, Welch's method is used instead of FFT [16][17]. The advantage of Welch's method is that smoother spectral components can be obtained and more accurate estimation of the PSD can be done [18].

The waveform is first divided into L number of sections.

$$x_i(n) = x(n + iD) \tag{4.1}$$

where $n = 0, 1, ..., M - 1$ and iD is the starting point for the ith sequence between $i = 0, 1, ..., L - 1$. D is the length of each segment

Then the periodogram of each is segment is calculated by first windowing each segment and then using the FFT. This is given in the following equation:

$$\tilde{P}_{xx}^{(i)}(f) = \frac{1}{MU} \left| \sum_{n=0}^{M-1} x_i(n) w(n) e^{-j2\pi fn} \right|^2 \tag{4.2}$$

where U is the normalization factor for the power in the window function $w(n)$.

$$U = \frac{1}{M} \sum_{n=0}^{M-1} w^2(n) \tag{4.3}$$

The result from eq.[4.2]is called the modified periodogram[16]. The Welch's method is finalized by computing the average of these modified periodograms:

$$P_{xx}^{W}(f) = \frac{1}{L} \sum_{i=0}^{L-1} \tilde{P}_{xx}^{(i)}(f) \tag{4.4}$$

where L is the number of segments.

This function is implemented in Matlab by executing the code $pwelch(x)$ in Matlab. By default, the function uses Hamming window, and the overlapping between segments is 50 %, which is used in this work.

4.2.2 Weight based on the Autocorrelation function

The second step in the quality check procedure is based on finding the repeating patterns using the autocorrelation function (ACF). In a method of Varon et al. (2012) [19] an algorithm is proposed to identify the artifacts in the ECG signal[19]. In general, this algorithm first divides the input ECG signal into segments of length L. Next, the ACF of each segment is calculated. Then the graph theory is used to identify the contaminated segments. By implementing the graph theory, a degree is characterized to each segment. These degrees are used as weights which indicates how clean the ECG section is [19]. These weights are used in our project as a parameter for reliability of the signal.

The matlab code for this algorithm was provided by one of the authors of [19]. In this work, relevant parts of this algorithm were taken out and modified for this work. The main modification to the algorithm is described follows:
The ECG signal used for the pre-process is already segmented into parts of N segments (suppose for now that this segment is called A_i where $i = 1, \dots, N$ is the number of the segment). When the ACF algorithm is conducted on segment A_i, segment A_i is further divided into J number of segments with length P (for explanation purposes, call this segment B_j), where $P < N$. The result is that j amount of weight are assigned inside A_i, while only one weight is used for further calculations of segment A_i. To resolve this, the average of j weights are taken and assigned to the segment A_i. So the following holds:

$$d_{A_i} = \frac{\sum_{a=1}^{j} d_{B_a}}{j} \tag{4.5}$$

where d is the weight assigned to the segment. If the weight is above a certain threshold, the segment is labeled as acceptable signal. Otherwise, the segment is sent to the filtering part in the FSM (see fig. 4.2)

For determining the threshold of the weights, same method is used that was used for the determination of the threshold for ratio of power distribution.

4.2.3 The Heart Rate

The second step in the quality check procedure is based on the heart rate (HR). According to several studies, as well as the consensus of experts, the normal resting heart rates for adults lie between 60 and 90 beats per minute (bpm) [20][21], while the AHA defines the normal HR as between 60 and 100 bpm [22]. Due to the presence of noise, some noises with a large amplitude can be mistakenly seen as a heart beat when detecting the R-peak. This will lead to a higher HR. Based on the HR, an assessment is conducted on the reliability of the segment of the signal. If the heartrate is outside the accepted range of HR, the segment is labeled as unacceptable.

The HR can be computed from the RR-interval. This is the distance in time between two consecutive R-peaks from the QRS-complex [23]. Since the HR is measured in beat per minute (bpm), the formula for calculating the average becomes

$$HR(bpm) = \frac{60}{RR_{avg}} \tag{4.6}$$

where RR_{avg} is the average distance between a set of RR-intervals in seconds. Note that other features from the QRS complex could be used for the calculation of HR. The R-peak is chosen for further determinations because of the widely use in studies since it is well defined and easy to locate [9]. For the detection and correction of the R-peak, an open-source Matlab based algorithm is used, called **R-Deco** [24].

In this method, the adaptive thresholding procedure of the Pan-Tompkins algorithm is implemented, which is used for the automated detection of the R-peaks that is part of the QRS-complex [25]. Before using the algorithm, the ECG envelopes are computed to get enhanced QRS-complexes and flatten the rest of the ECG [24][26]. The flattened ECG is defined as $F_{ecg} = U_{ecg} - L_{ecg}$, where U_{ecg} is the upper- and L_{ecg} is the lower ECG envelopes. By subtracting the lower envelopes from the upper envelopes, baseline is eliminated and only a positive signal F_{ecg} remains for the detection of the R-peak. This procedure is shown in figure 4.3.

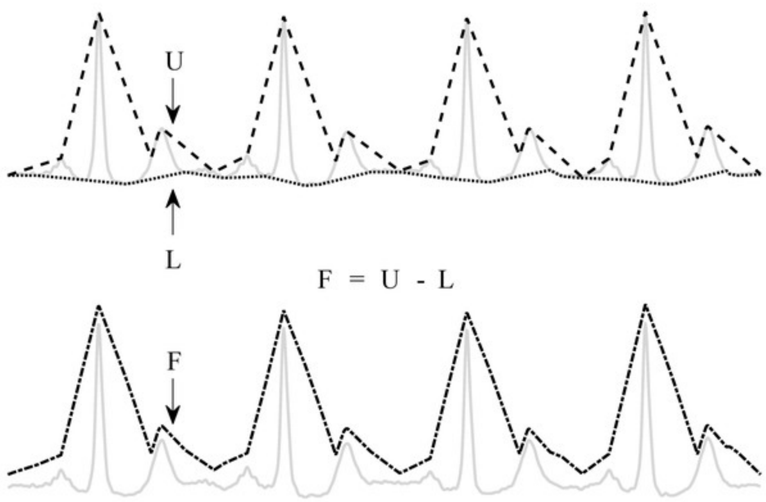

Figure 4.3: The flattened ECG (F_{ecg}) is given in the graph below. F_{ecg} is obtained by subtracting the lower envelope from the upper envelope of the signal. This figure was obtained from [24]
DOI:https://peerj.com/articles/cs-226/#fig-2

After determining the QRS-complex positions from the flattened ECG, the exact R-peak location is determined by using the original ECG signal. This is needed because the R-peak might be shifted in the flattened ECG signal towards the notch of the S-wave[24]. In figure 4.1, the notch of the S-wave is shown.

4.3 Artifact Removal

This section will discuss the removal of artifacts in the ECG signal.In subsection 4.3.1, the different kinds of artifacts present in the ECG are discussed together with the detection method of those artifacts by use of the power spectral density of the ECG signal and the plot of the ECG signal. After detection of the artifacts, removal methods are determined in section 4.3.2 and design choices are discussed.

4.3.1 ECG Artifacts

This subsection will discuss the different kinds of artifacts in the incoming ECG signal. Figure 4.4 shows a raw ECG signal ,at the upper figure, which is not processed and the power spectral density (PSD) ,at the lower figure, of the same ECG signal. Both images are used to detect the artifacts in the ECG signal.

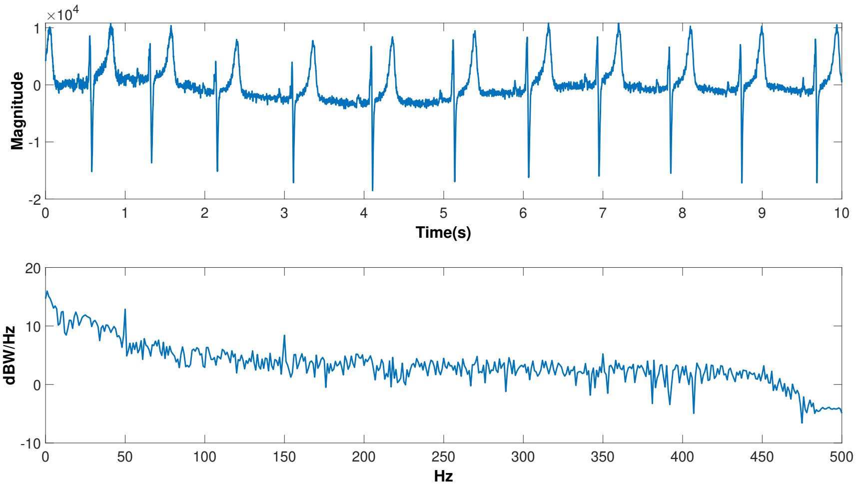

Figure 4.4: An unfiltered 10 second ECG segment and its corresponding PSD.

The most common ECG artifacts are: The powerline interference which, depending on the measurement location, is 50 Hz or 60 Hz and its harmonics. Secondly, the baseline wander is a low frequency noise component in the ECG signal which is mainly caused by respiration, body movement and electrode-skin impedance [27]. This phenomenon happens below 0.5 Hz [28]. Thirdly Electromyographic (EMG) noise is generated from electrical activity of the muscle [6] and tends to be non-stationary and has a frequency that overlaps the original ECG signal from 1 Hz up to 120 Hz . Finally the electrode motion artifacts are mentioned, which are mainly caused by skin stretching which alters the impedance of someones skin around the electrode. This artifact mainly occurs in the range from 1 to 10 Hz [27].

The PSD shown in Figure 4.4 has peaks at 50 Hz, 150 Hz and 350 Hz. These are the powerline interference frequencies. The multiples of 50 Hz, 100 Hz and 150 Hz must be removed which is done by means of notch filters, because those three frequencies are in the band of interest of the ECG signal. Details about the notch filters used for the removal of the powerline interference are given in subsection 4.3.2.

From Figure 4.4 it can be seen that the base of the ECG signal of the signal moves upwards and downwards [27]. This is a indication of baseline-wander. It could also be interpreted as electrode motion artifact because a similar effect on the signal happens when these artifacts are present however, the spectral content does not overlap that of the PQRST wave [27], so the artifact cause for this signal is baseline wander.

The band of interest of the ECG, as mentioned earlier in Chapter 4.1,is from 0.5 Hz to 40 Hz or 150 Hz. Figure 4.4 shows however frequency components higher than the upper limit of the frequency band of interest. In the design of the ECG system of the report, these frequencies are filtered out and treated as frequency components which probably contain mostly noise than usable information.

A different kind of noise, present in some of the signals, can be seen in Figure 4.5.This noise component causes oscillation in the ECG signal which can be seen in Figure 4.5. By investigating the PSD of the signal, it is concluded that this noise is a single component in 120 Hz. The noise is not however a powerline noise, since this data has a powerline frequency of 50 Hz and its harmonics. It could also not be baseline, because the frequency of the noise is too high. EMG noise is non-stationary and has a range of frequency components, so that is not the noise which is present at 120 Hz. This noise however can be removed in the same manner as the powerline interference noise. Further details will be given in subsection 4.3.2.

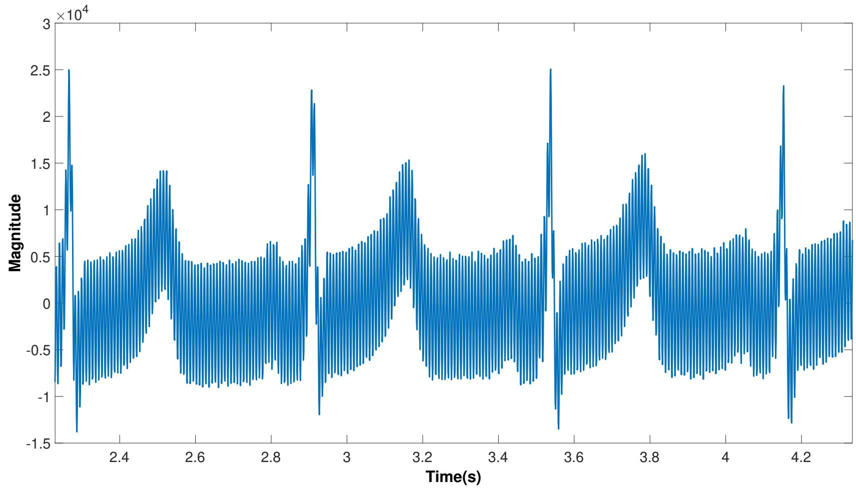

Figure 4.5: The effect of the 120 Hz noise component on the ECG signal of channel X measurements from patient 20 to patient 33.

4.3.2 Filtering

This section will discuss the filters used to remove the artifacts described in subsection 4.3.1. The design choices of the different kind of filters together will be discussed in detail in this subsection.

IIR vs FIR filters.

The filters which are used for artifact removal in the ECG and respiratory signal design are digital filters. There are two classes of digital filters, the Finite Impulse Response (FIR) filters and the Infinite Impulse Response (IIR) filters.

The mathematical difference between the IIR filter and the FIR filter is that the IIR filter is a recursive function which has its filter output as input. The mathematical representation of FIR filter is $y[n] = \sum_{k=0}^{N} a(k)x(n-k)$ IIR filter representation is $y[n] = \sum_{k=0}^{N} a(k)x(n-k) + \sum_{j=0}^{p} b(j)y(n-j)$ [29].

The IIR filter has an advantage that at a similar roll off as the FIR filter, a lower IIR filter order is sufficient enough to have the same effect as a FIR filter which has a much higher filter order [29]. A lower order filter means that the complexity of that filter is lower because less calculations are needed to be done , so the filter will be faster than a filter which has a higher order. However the IIR filter has a nonlinear phase ,which causes phase distortions , and stability issues[29]. The FIR however is always stable, but needs higher order filters to have the same performance as a IIR filter in frequency response. The group delay at FIR filters is equal at every frequency due to linear phase of these filters[29] .

Delays in filters is due to the fact that the number of time data at the input must be proportional to the number of terms (so for example N). This is needed for the filter to work. So increasing the filter order, will cause that the delays is increased [29].

The IIR filter is used for real-time applications[29]. This is why it is used for both the ECG and respiratory signals. The IIR is more useful because a faster filter with a lower order causes less delays in the signal after filtering, which is an advantage for a real time application.

Removal of Powerline Interference

The powerline interference consists of one frequency component and its harmonics. In the band of 0.5 Hz to 150 Hz there are three frequency components which need to be removed to eliminate the powerline interference, which are 50 Hz, 100 Hz and 150 Hz. To this end, a filter is needed to filter out a specific frequency while maintaining the other frequency component.This can be done by means of notch filter.

The notch filters implemented in the design are made by using the matlab function **iirnotch**. This function is a second-order IIR notch filter which needs as input: the normalized notch frequency that needs to be removed and the -3dB bandwidth. The output from this function is the denominator and the numerator of the transfer function of the designed notch filter, which is shown by the function:
H(z) $= \frac{K \cdot (z^2 - 2z \cdot cos(\theta) + 1)}{(z^2 - 2r \cdot z \cdot cos(\theta) + r^2)}$

The r in this function is the required magnitude of the poles, which is calculated from the following function: r $\approx 1 - (\frac{BW_{\text{-3dB}}}{fs}) \cdot \pi$.In this function the -3dB bandwidth is divided by the sampling frequency. θ is the angle of the pole location which is given by: $\theta = \frac{f_0}{fs} \cdot 360°$ in which f_0 stands for the notch frequency.K is the unit-gain scale factor which is given by: $K = (1 - 2r \cdot cos(\theta) + r^2) \times \frac{1}{2 - 2cos(\theta)}$.

According to [30], good values for the required magnitude for the poles are between 0.9 and 1. So the bandwidth needs to be chosen accordingly. Also the consideration has to be made that this bandwidth needs to be as narrow as possible to avoid distortion of other frequencies which are not intended to be altered. In the design of the notch filter, a -3dB bandwidth of 1 Hz is chosen. This is chosen because it satisfies the requirement of a good magnitude of poles and also for smaller bandwidths, the notch filter reduces the power at the powerline interference frequencies less effectively. This is because a second order filter has a slope of -40dB per decade. The reduction of the unwanted components is reduced by narrowing the bandwidth. Choosing a bigger value reduces the frequency which is intended to be removed however the unintended distortion on other frequency components is larger. Keeping this in mind and by using the matlab function **freqz**, which visualizes the bode plot of the notch filter, the -3dB bandwidth is chosen. Figure 4.6 shows the bode plot for a notch filter with -3dB bandwidth of 1Hz and a notch frequency of 50 Hz.

The noise present in 120 Hz is also removed with a notch filter. The design of the notch filter is identical with the notch filters which are used for the powerline interference. However the -3dB bandwidth of this notch filter is chosen to be 2 Hz, this can be done because 4 Hz bandwidth is also in line with the required magnitude for the poles. The different bandwidth is chosen because the noise component at 120 Hz is not removed entirely with a -3dB bandwidth of 1 Hz.

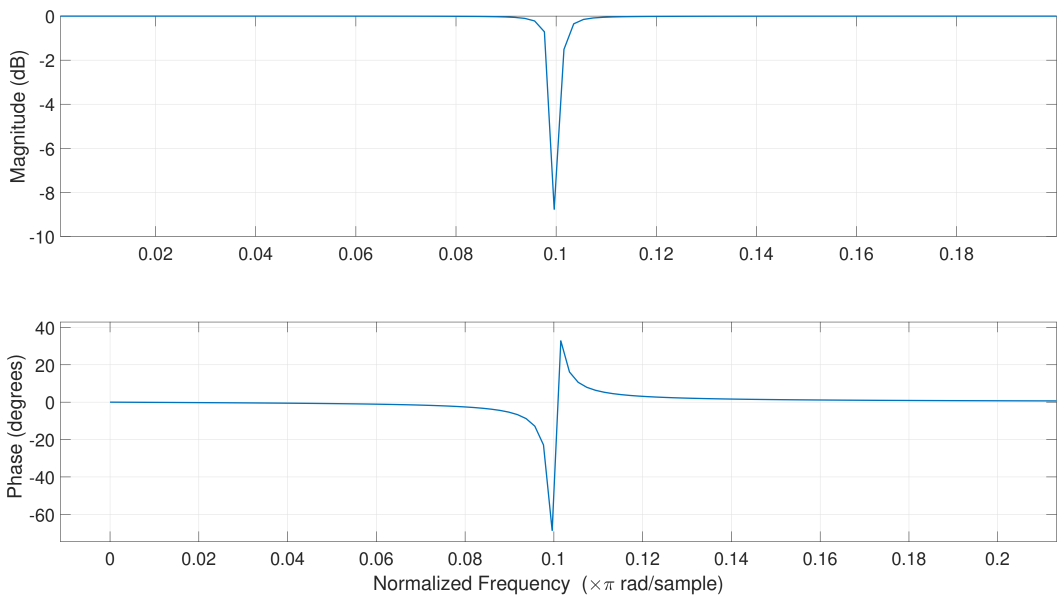

Figure 4.6: The bode plot of a second order IIR notch filter, with a notch frequency of 50 Hz(50/(Fs/2) = 0.1 π rad/samples where Fs = 1000 Hz) and a -3dB bandwidth of 1 Hz

The denominator and numerator values obtained from the iirnotch function are put in the matlab function **filtfilt**. This function uses the information of the denominator and numerator together with the ECG signal, which is used as input, to make the filter which is needed. The filtfilt function is a zero-phase digital filter which filters the input signal in both the forward and reverse directions. This causes that there is zero phase distortion, removal of delay effect[29] and a filter function with the squared magnitude of the original filter and double the order filter which is specified by the denominator and numerator values of the function iirnotch. However zero phase filtering causes that the data at end of the time trace is eliminated [29].

Removal of Baseline-wander

Baseline-wander is an artifact which is linked to respiration and affects the ECG signal.As described in section 4.3.1, this effect is due to frequency components below 0.5 Hz. Keeping this in mind, the information of the ECG signal lower than 0.5 Hz must be removed to eliminate baseline wander and the frequencies above 0.5 Hz must be preserved to prevent unwanted loss of information. This can be done with a high-pass filter.High-pass filters remove frequencies lower than the cutoff frequency, which is in this design 0.5Hz, while maintaining the higher frequency components. As mentioned in the filter design section , the filters in this design are IIR filters. IIR filters have different implementation methods like the Butterworth, Chebyshev, inverse Chebyshev, Cauer and Bessel[29]. The chosen method in the ECG signal design of this paper is the Butterworth filter. The Butterworth filter is preferred in literature for the analysis of ECG,see for instance [28],[31], [19] and [32].

The choice of the Butterworth filter can be justified by the fact that the Butterworth filter does not have a ripple in the passband and stopband. This is desired because, no alterations on the signal is wanted in the passband region of the filter. The same can be said about the stopband region, where all frequencies are desired to be totally removed without any ripple behaviour[28]. In [32] removes baseline wander by using a forward/backward fourth order Butterworth high-pass filter with a cutoff frequency of 0.5 Hz. The reason to use this method is described in [32],where that this method is one of the most accurate and easiest method to implement.

For this design a high-pass Butterworth filter of order 4 is used with a cutoff frequency of 0.5 Hz, which is the same as in [32]. The computational cost is lowered by choosing a lower order than 5 and also according to [32], as stated earlier, the filter used with filter order 4 is one of the most accurate and easiest methods to implement. To implement this, the matlab function **butter** is used with as input: the filter order, the normalized cutoff frequency and the type of filter. The outputs of this function are the transfer function coefficients b and a returned as a row vector of length n+1 where n is the order of the filter. The transfer function expressed in terms of a and b is: $H(z) = \frac{b(1)+b(2)z^{-1}+....b(n-1)z^{-n}}{a(1)+a(2)z^{-1}+...+a(n+1)z^{-n}}$

The implementation of the high-pass Butterworth filter ,however ,distorts the ST-segment of the ECG signal. The ST-segment can be seen in Figure 4.1. The distortion of the ST-segment is shown in Figure 4.7.This phenomenon is due to the fact that high-pass filters suffer from phase shift, which causes that the first 5 to 10 harmonics of the signal are affected. So when a high pass filter is implemented with a cut-off frequency of 0.5 Hz , up till 5 Hz can be affected[33]. However when the signal gets passed trough a zero-phase filter, by using the **filtfilt** matlab function, the phase distortion is nullified and the ST-segment distortion is restored. A second issue solved is the delay induced due to the filtering, by implementing a zero-phase filter.

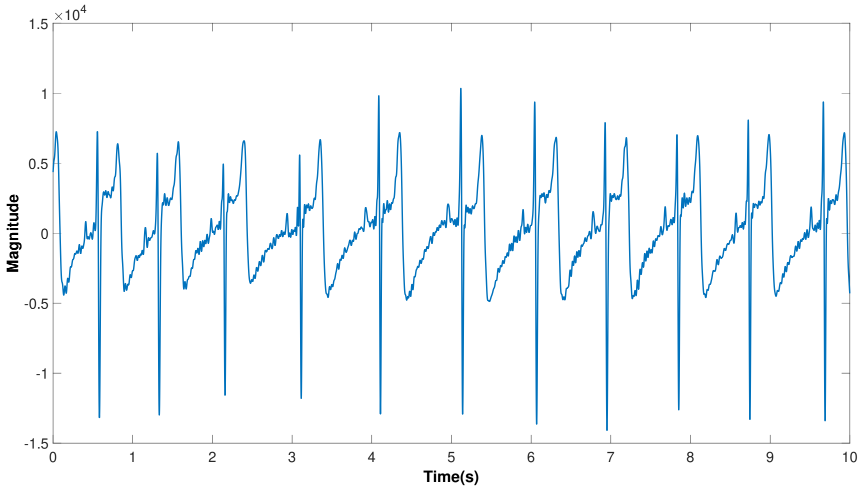

Figure 4.7: A ten second segment of a ECG signal with distorted ST-segment after high-pass filtering with a cut-off frequency of 0.5 Hz.

Removal of High Frequency Components

This section will discuss the removal of the high frequency components. The meaning of high frequency components in this design, are frequencies higher than the maximum frequency of the band of interest for the ECG signal. In the case of this design, the maximum frequency taken for the analysis of the ECG signal is 150 Hz. This is the minimum frequency needed for the analysis of high frequency components of the ECG signal according to [14], [15]. An upper cutoff frequency of 150 Hz is at least needed to measure routine durations and amplitudes accurately and this cutoff is also needed for diagnostic purposes. The ANSI/AAMI also recommends a high-frequency cutoff of at least 150Hz for ECG signal [15]. Due to these statements, the high frequency cutoff for the low-pass filter is chosen to be 150 Hz while the ECG signal is not sent as output.This signal is rather used to make computations, for example the R-peak calculation, in the ECG system,which is also explained in section 4.1.

The frequencies higher than 150 Hz need to be removed, while frequencies lower than 150 Hz need to be preserved. This is done with a Butterworth low-pass filter. This filter is designed in the same manner as the high-pass filter which is used to remove the baseline wander from the signal. However, a new order needs to be defined for the low-pass filter. This is done with the Matlab function **freqz** , where the phase behaviour of the ECG signal is analyzed, and the PSD . The order is determined by looking at different filter orders and their phase response. The filter order is increased until the phase response has a distorted segment in the band of interest to find the highest order filter which could be used.This is investigated because, such distortions also affect the amplitude response thus also the ECG signal if the order is chosen to be too high. Also the highest possible order means also the steepest slope after the cut-off frequency, which is wanted because the frequencies above the cut-off frequencies are unwanted. However the computational cost of the total design needs to be considered in the filter order aswell, because a higher order filter means that the filtering will take longer. For a filter order,50 ,the phase response gets distorted, which also distorts the amplitude response. So 49 could be the order chosen for the filter as the maximum filter order without distorting the ECG signal.However due to the possible high computational cost of such a filter, it is investigated by looking at the PSD if a lower filter order can be chosen which gives a sufficient result. After filter order 14, the change in the PSD at higher frequencies than 150 Hz is not significantly better so a filter order of 14 is chosen for the low-pass filter.This order removes the unwanted frequencies well enough while also working faster than a filter order of 49. The bode plot of the 14 order low-pass Butterworth filter can be seen in Figure 4.8.

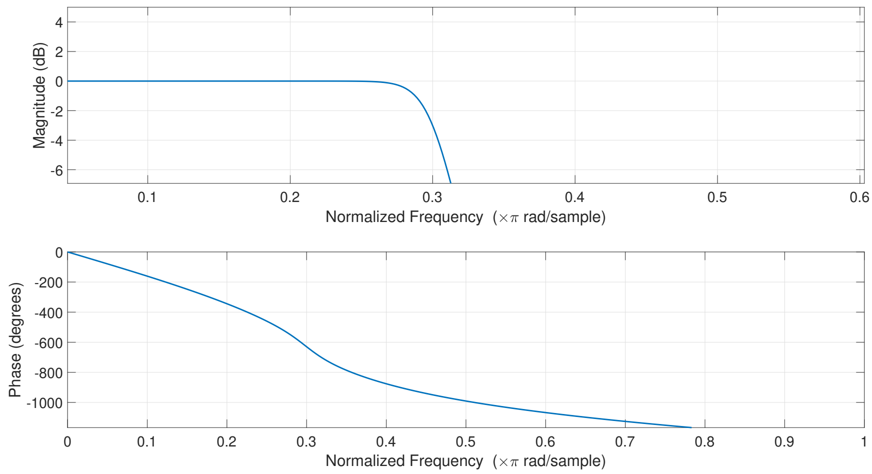

Figure 4.8: The bode plot of the Butterworth lowpass filter with a cut-off of 150 Hz (150/(Fs/2) = 0.3 π rad/samples where Fs = 1000 Hz.)

As discussed in chapter 2, a second frequency band for ECG signal is used for the ECG when it is sent as output. This band is from 0.5 Hz to 40 Hz. This band is used to enhance visualization of the ECG signal at the GUI described in [2]. According to [14], most of the information of the ECG signal is contained within 40 Hz, except for the QRS complex which has many frequency components above 100 Hz. This is especially the case for R-peak frequency components. The source also states that information between 0.5 Hz and 40 Hz is used for monitoring purposes. The reason for this is that the band of 0.5 Hz to 40 Hz produces a more stable signal with less baseline noise and fewer high-frequency artifacts[14]. According to [15], the band until 40 Hz will invalidate any amplitude measurements used for diagnostic classification, however the subgroup in paper [2] does not perform any diagnostic classification directly from the ECG signal. Thus the output of the prep-rocessing step will be the ECG signal with a band until 40 Hz. This will be done, by keeping in mind ,that the ECG is not altered badly visually and that the ECG is less noisier than a ECG signal with a band of 0.5 to 150 Hz.

The order of the low-pass Butterworth filter for a cut-off frequency of 40 Hz is chosen by taking the same considerations and steps as the low-pass filter which has 150 Hz. The order of this filter is chosen to be 14. The bode plot of this filter can be seen in Figure 4.9. For both low-pass filters, the filters are made with the Matlab functions butter and the filtfilt, which where explained in the Removal of baseline-wander and Removal of Powerline interference subsection respectively.

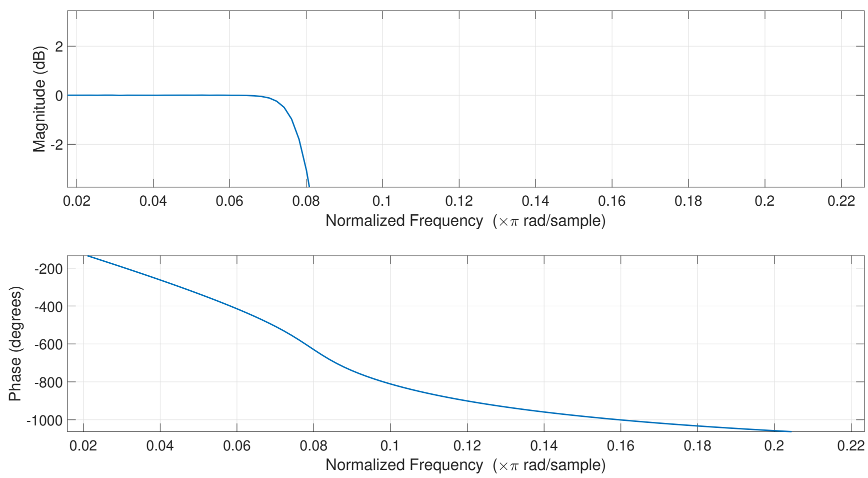

Figure 4.9: The bode plot of the Butterworth low-pass filter with a cut-off of 150 Hz (40/(Fs/2) = 0.08 π rad/samples where Fs = 1000 Hz.)

Chapter 5

Respiratory signal system design

This chapter will discuss the respiratory signal system design.The finite state machine will be discussed at the first section. Afterwards a detailed description of the components in the respiratory signal system will be given. The components that are discussed are the artifact removal part, the respiratory rate calculation, and the signal quality checks performed in the overall system.

There is in literature, however, no consensus of a clean respiratory signal. The respiratory signal can vary from very clean and periodic signal to a very complex signal. This results in a challenging task when the respiratory signal is analysed. The aim of this project is limited to a power quality check and the time interval check in which a breath is taken.

5.1 Finite State Machine of the Respiratory System

The implementation of the overall Respiratory system design is shown in a finite state machine, which can be seen in Figure 5.1. The system consists of the following components: A function which detects whether the respiratory signal has NaNs or not, the power quality check which is identical to the ECG system counterpart, a function which calculates the respiratory rate, and finally ,a quality check which controls if there is a breath taken within 10 seconds. The system labels the respiratory signal with a SQI by looking at:

- depending on the power quality check
- the breath check.

The NaN check at the beginning of the respiratory signal system has the same reasoning as the NaN check in the ECG system which is described in 4.1. The check is done to prevent errors in the code and also data with NaNs is unusable. So the signal gets separated and it gets labeled with SQI = 0 if it has NaNs in it.

Afterwards the signal gets passed to a bandpass filter with a band of 0.1 Hz to 0.5 Hz. This differs from the ECG system design, because respiratory has only artifacts outside the band of interest. More detail about the filters used and the choice of the band of interest will be given in section 5.2.1.

The signals gets downsampled to 2 Hz. This is because the relevant frequencies are below 1 Hz, so all the other frequency components are irrelevant for further analysis.

After downsampling, the signal is passed to the power check. This is identical to the ECG signal power check. Again as described at section 4.1,the power percentage in the band of interest of the respiratory signal is checked and if it is below the threshold in the quality check, then the signal is labeled with a SQI = 0 and it will be sent as output. If this condition is not satisfied, the signal will be passed to the system with an SQI = 1 label and there the respiratory rate will be calculated.

Afterwards, the second test will be conducted. The duration of the breaths in the signal will be checked. If this duration is longer than 10 seconds then the SQI will be zero, otherwise it will be 1. After the quality check, the signal will be sent out together with its respiratory rate and SQI to the system design subgroup [2].

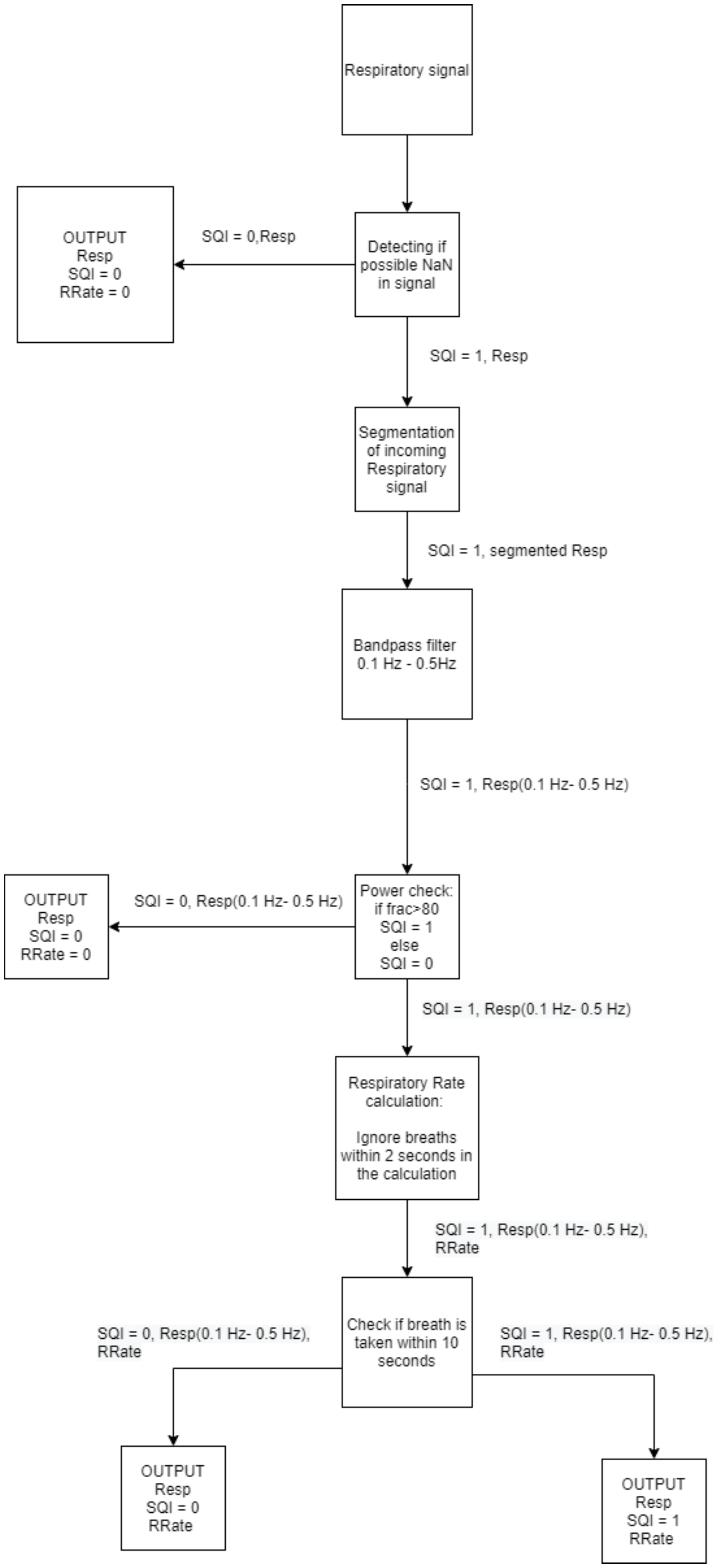

Figure 5.1: The Finite State Machine of the Respiratory system

5.2 Artifact Removal

The Artifact Removal section will discuss the removal of artifacts in the respiratory signal. Section 5.2.1 will emphasize on the artifacts in the respiratory signal, while section 5.2.2 will discuss what kind of filters are used to remove the artifacts discussed in section 5.2.1.

5.2.1 Artifacts Affecting The Respiratory Signal

The detection of artifacts in the respiratory signal is done with the idea that frequencies above and below the frequency band of interest of the respiratory signal are too low or too high for breathing. The band of interest according to [34] is between 0.1 Hz and 0.5 Hz which is 6 breaths in a minute and 30 breaths in a minute. Also the knowledge that the baseline wander noise of ECG signal is 0.5 Hz or lower and that this noise is linked with respiration also confirms the upper band limit of 0.5 Hz.

5.2.2 Filtering

The filters used in the design of the respiratory signal are the IIR Butterworth low-pass and IIR Butterworth high-pass filter. The reason is simply because the band of interest needs to be preserved while other frequencies of the respiratory signal need to be removed and these frequencies outside the band are considered as the only noise sources in the respiratory signal, which is also stated at subsection 5.2.1. The design choices of both filters underwent the exact same steps which were taken at the high-pass and low-pass filters in the ECG design, which are used to remove the high frequency components and the baseline wander. A more detailed description can be seen in section 4.3.2.The only difference between the filters of both designs are that the respiratory signal filter orders differ from the ECG signal filters.

The orders of the low-pass filter and high-pass filter are determined with the matlab function freqz, which is also described in section 4.3.2 to determine the filter orders used in the ECG design. By looking at the phase of the bode-plot of the filters and by investigating with different orders of filters to determine the filter order which could be taken before distorting the respiratory signal , the order of 6 is chosen for the low-pass filter and order 4 for the high-pass filter in the respiratory signal design.

5.3 Respiratory Rate Calculation

The respiratory signal consists of the inhale peaks, exhale troughs, inhale onsets, exhale onsets, inhale pauses and exhale pauses [35]. Only the inhale peaks and exhale throughs are used or the calculation of the respiratory rate. The inhale peaks and exhale throughs can be seen in Figure 5.2.

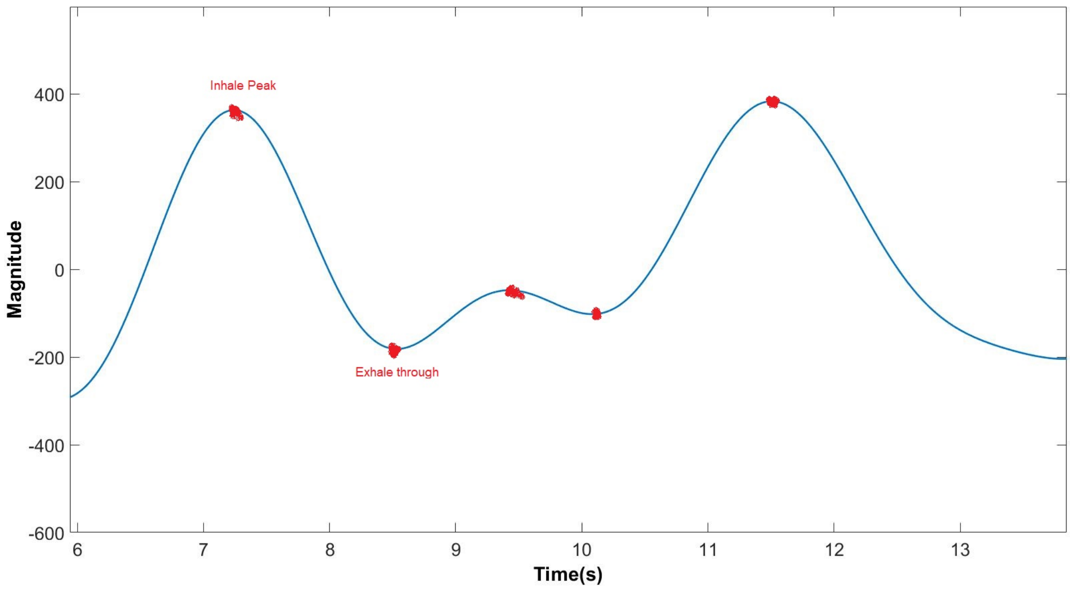

Figure 5.2: Respiratory signal with its components, figure is obtained from [35]

The consideration that a breath is taken after an inhale peak and exhale throughs is used in the calculation of the respiratory rate. The reason to not assume that a breath is taken when a new inhale peak is detected without a exhale throughs, is because at the end of the time interval it could be that a new inhale peak gets detected without the exhale throughs, which is not a breath taken but only indicates that the subject has inhaled.

The matlab function **findpeaks** is used to find the inhale peaks in a given time-interval. The inputs of this function are : the respiratory signal, the sampling frequency, **'MinpeakDistance'** and threshold which eliminates 'peaks' within curtain time interval. The last input is used to eliminate possible breaths taken within 2 seconds. The reason for this is that the maximum respiratory rate is 0.5 Hz, which corresponds with a breath duration of 2 seconds per breath. So faster breathing results in a respiratory rate which is higher than 0.5 Hz and are thus excluded from the inhale peaks detected. Findpeaks is also used for the detection of the exhale throughs. This is possible by reversing the respiratory signal and thus creating the illusion that the exhale throughs are peaks. Both the inhale peaks and the exhale throughs are stored in vectors and the amount of components in both vectors are compared. From both the vectors, the vector which contains the least components is taken as the amount of breaths taken in the time interval. The reason is, as explained earlier, that a breath needs both the inhale peak and exhale throughs to be seen as a breath, so they are needed in pairs, which means that both vectors should have the same amount of components. After this the amount of breaths in the time-interval is obtained,which corresponds to the respiratory rate.

5.4 Signal Quality Indicator of The Respiratory Signal

5.4.1 Downsampling

After the filtering of the respiratory signal, which is discusses in subsection 5.2.2, the signal is downsampled to 2 Hz. This is because the frequencies of interest are between 0.1 and 0.5 Hz. For example the data from section 3.1 has a respiratory signal which is sampled at 250 Hz and having information until 125 Hz is just too much and unnecessary frequency components.

The matlab function **resample** is used. The following inputs must be given as input to the function to use it properly: Respiratory signal, p and q. P and q are used in the ratio of $\frac{p}{q}$ which is multiplied with the original sample rate. The matlab function resample applies a FIR Anti-aliasing Low-pass Filter and also compensates the delay introduced by the filter. An anti-aliasing filter is used to prevent the effect of aliasing. This phenomenon is present when the sampling frequency does not satisfy the sampling theorem. The sampling theory states that the sampling frequency should be at least twice the signal frequency. This sampling frequency is called the Nyquist frequency and it can be seen in equation 5.1. If the sampling theorem is not satisfied then new frequency components will be created [36]. This is why it is recommended to have a low-pass filter before sampling which removes higher frequency components then the Nyquist frequency.

$$F_{\text{signal}} < \frac{F_{\text{Nyquist}}}{2} \tag{5.1}$$

5.4.2 Spectral Distribution Ratio of Respiration Signal

The idea of looking at spectral distribution ratio of the respiratory signal is identical to the idea of the spectral distribution ratio check of the ECG. The same code is implemented, only the frequency range of interest is different. More information about the method is described in section 4.2.1. Like given in 5.2.1, the range of interest is between 0.1 Hz and 0.5 Hz.

5.4.3 Breath Check

The breath checking is a signal quality assessment which is based on the breath rate, where the way of thinking is again similar like the HR checker. The SQI for the breathing is given as acceptable if the duration of a breath is within the determined range of duration. This range of duration is taken between 2 seconds and 10 seconds, which is the same as a range between a frequency range of 0.1 Hz and 0.5 Hz.

The breathing duration is determined by looking at the peaks of the signal. If the peak-to-peak distance is within 2 seconds, the algorithm decides that this is not possible and annotate to the signal a bad quality. This conditioning is set because of physical reasons. If the peak-to-peak distance is within 2 seconds, the conclusion can be made that the signal is contaminated with excessive noise. This can be during speaking or other physical activities. With the same reasoning, if the duration of peak-to-peak distance is longer than 10 second, the signal will again be labeled with a bad quality.

Chapter 6

Results & Discussion

6.1 Results

The incoming ECG signal, gets filtered depending on the result of the three quality checks at the start of the ECG system. If the ECG signal gets filtered, the signal has a bandwidth which spans from 0.5 Hz to 150 Hz which gets passed through the quality checks again, and depending on the result, the ECG sent to the output gets labeled with an SQI which is equal to 1 or 0. The result of filtering can be seen in Figure 6.1.

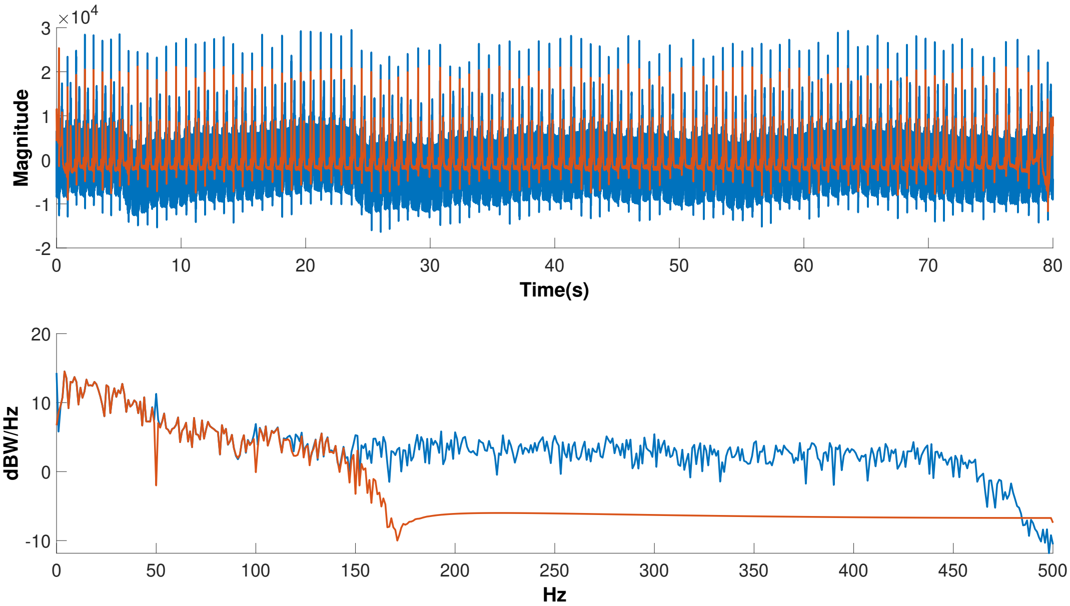

Figure 6.1: The blue images are the raw incoming ECG signal with its PSD. The orange images are the filtered ECG signal with the band of 0.5Hz and 150 Hz.

The ECG signal gets resricted to the band which spans from 0.5 Hz to 40 Hz, this is done with a bandpass filter. This happens after the passage of the ECG signal through the filtering and quality check. The reasoning of this narrowing of the band of interest can be reviewed in section 4.3. The result of this filter can be seen in Figure 6.2 .

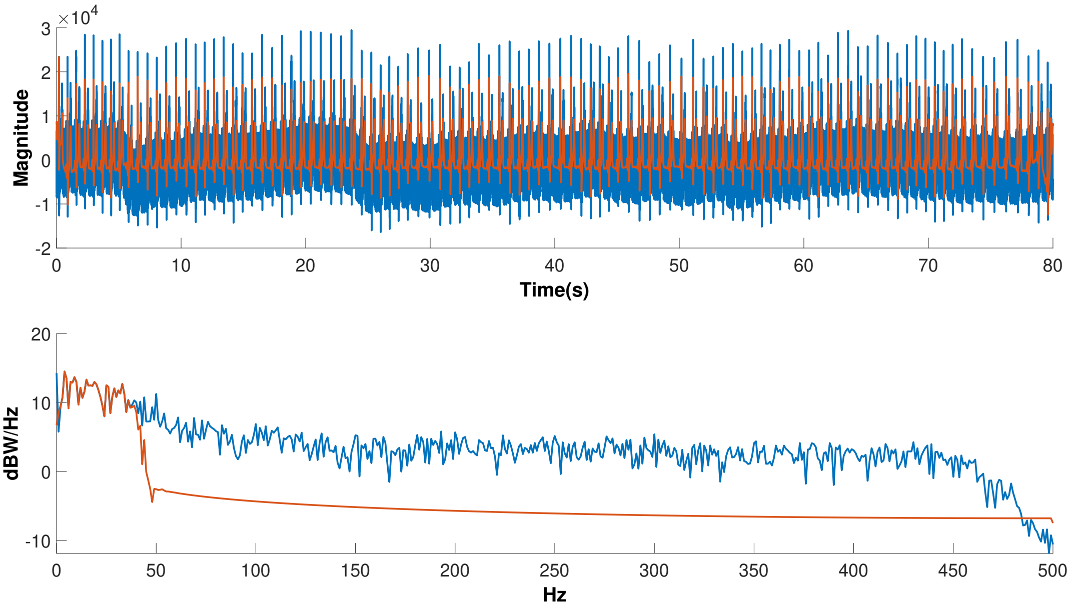

Figure 6.2: The blue images are the raw incoming ECG signal with its PSD. The orange images are the filtered ECG signal with the band of 0.5Hz and 40 Hz.

The accuracy of the signal quality indicators used in the ECG system design are mainly inspected with the CinC2017 dataset described in section 3.3 , where the ECG signals are already labeled with a quality indicator. These quality indicators are used as reference to determine how accurate the labels are which are obtained from the ECG system.This system uses signal quality checks that use optimal thresholds of 99.997 for the power check and 0.91 for the auto-correlation quality check together with a 40 bpm to 180 bpm range for the heartrate check. 70.0975% of the ECG signals from the CinC2017 dataset are labeled in the same manner as the labels obtained by the ECG system. So both the given labels and the ECG system agree with a percentage of 70.0975% on the labeling of all the ECG signals given by the dataset. However there is 29.9025% disagreement between the given labels from the third dataset and the ECG system. The disagreements could be in two different manners. Firstly, the labels of the third dataset label the ECG data as bad but the ECG system sees it as good. The percentage of this is 17.5666%. Secondly, the third dataset labels the ECG data as good but the ECG system sees it as bad. The percentage of this situation is 12.3359%.

The driver dataset is analysed by checking at different outputs and then compared with each other. The first and last 15 minutes of the recordings,of this dataset are in relaxing mode. In between, the patient is driving a car. A duration of 15 minutes corresponds to (rounded-off) 12 segments of each 80 seconds. So the first and last 12 segments can be seen as relaxation mode.

To analyze the segments, the following parameters are being implemented as the output of the process:

- initial overall SQI of the segment (combining all the initial sub-SQI's. The sub-SQI's are the SQI based on NaN detection, on power, on ACF and on the heart rate)
- final overall SQI of the segment after processing the segment
- the average heart rate and the initial spectral distribution ratio before and after processing the segment.

Segment number	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**
initial fraction [%]	99.9935	99.9884	99.9883	99.9832	99.7603	99.9079	99.955	99.9577	99.9666	99.8943	99.9755
processed fraction [%]	99.9998	99.9993	99.9994	99.9994	99.9817	99.9971	99.9976	99.9990	99.9988	99.9978	99.9991
heart rate [bpm]	72.1674	70.5935	67.9074	68.7425	78.5557	86.0864	79.0363	78.4353	79.6530	76.3562	77.2658
initial SQI	0	0	0	0	0	0	0	0	0	0	0
final SQI	1	1	1	1	0	1	1	1	1	1	1

Table 6.1: 'ecg7' of the driver dataset is runned, and the results from segments 9 till 17 are shown in the table

Patient *ecg7* from the driver dataset is processed and examined with the threshold values for the three-step quality check which were obtained by using the CinC2017 dataset. In table 6.1, a part of the results are given. When looking at the first 12 segments, it can be seen that there are no bad labeled segments after processing the signal, which means all the sub SQI's are labeled as 1. Also, it can be observed that the average HR is almost stable with a mean of 73.5924 ± 3.6422 bpm. On the other hand when looking at results of the segments while driving (after the first 15 minutes), there is an increase in the average heart rate and is less stable, resulting in a mean of 76.700 ± 4.3479 bpm. Increased activity of the patient can be the reason of an increased HR, and experiencing stress and pressure during the driving could be the reason for more instability and more fluctuating heart rate. With this said, the amount of unaccepted segments due to bad labeling is increased while driving. The percentage of bad labeled segments while driving is 28.57% compared to whole range of segments while driving. Multiple reasons can lead to this bad quality of segments, such as electrode contact noise, motion artifacts and muscle contractions[8]. Looking at segment number 13 in table 6.1, it can be observed that the final SQI is still labeled with zero, even after filtering it. The plot of segment 13 is given in figure 6.3.

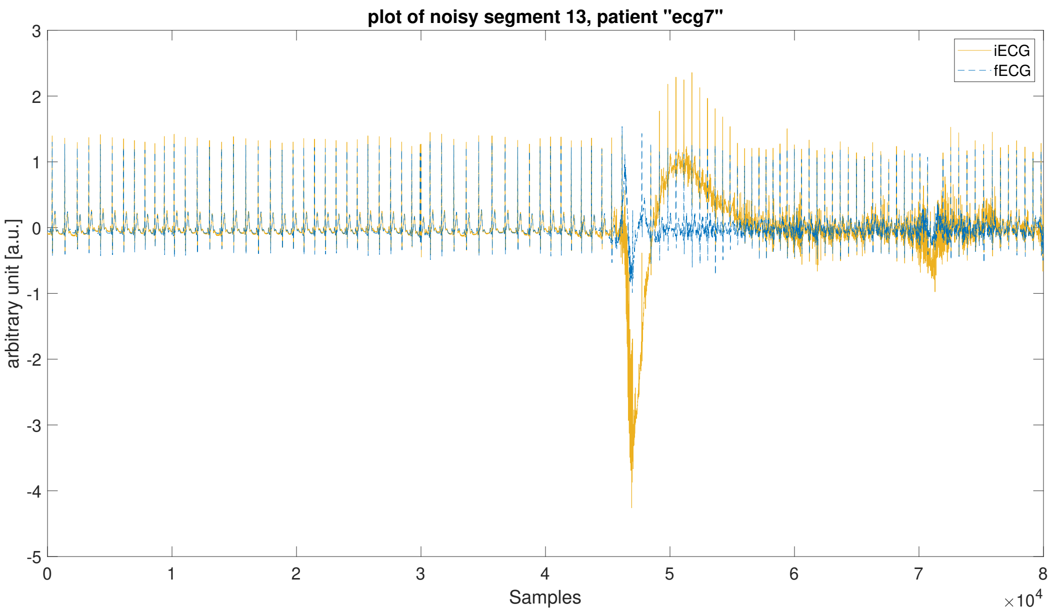

Figure 6.3: The noisy segment 13 of patient 'ecg7' the driver dataset. Even after filtering, the right part of the segment seems noisy

Segment 13 is the moment where the patient starts with driving. Based on this, it can be said that artifacts due to body motion affects the quality of this segment.

The SQI labeling of the ECG signals of all patients of the Stress dataset is also analysed. The percentage of bad labeled ECG signals of every patient is 13.1370 % and the good quality signals according to the ECG system are equal to 86.863%.

The respiratory signal is filtered by using a band-pass filter which spans from 0.1 Hz to 0.5 Hz. Figure 6.4 shows one of the respiratory signal segments from the Stress dataset before filtering with its corresponding PSD and Figure 6.5 shows the result after the band-pass filtering of the same respiratory signal with its corresponding PSD.

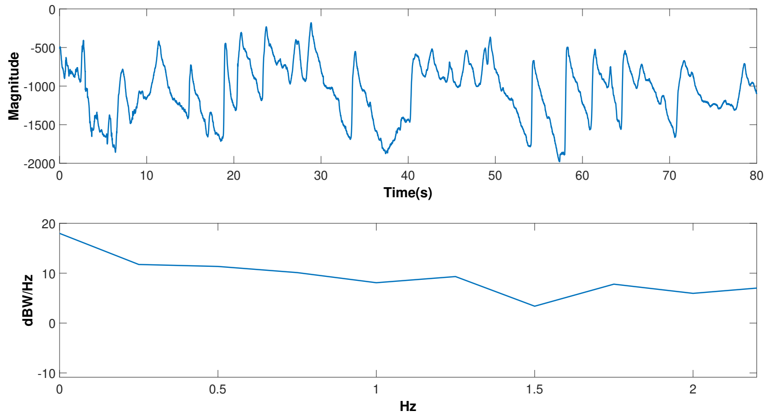

Figure 6.4: An unfiltered respiratory signal segment from the Stress dataset with its corresponding PSD.

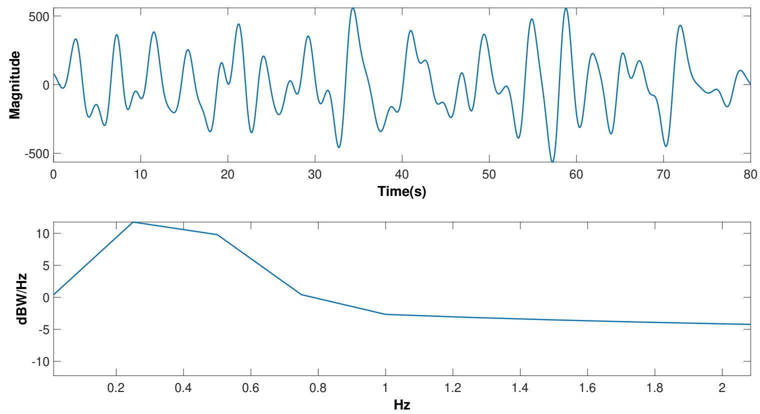

Figure 6.5: A filtered respiratory signal segment from the Stress dataset with its corresponding PSD.

The respiratory signal SQI is analysed with the Stress dataset. In this it is looked at the amount of bad signals per phase.The respiratory signal is segmented in lengths of 80 seconds when this is done. For the baseline phase ,80 from the 277 respiratory segments give a bad signal which corresponds with 28.88%. The Story telling phase gives 56% bad signals. The memory task phase has 7 bad segments out of 34, which corresponds with 20.59%. The stress anticipation phase has 68 bad segments out of 267 which gives a percentage of 24.72%. The video exposition phase has 29 bad segments out of the 66 which corresponds to a percentage of 43.94%. The Arithmetic task phase had 23 bad segments out of 121 which corresponds to a percentage of 19%.

6.2 Discussion

Figure 6.1 shows the result of the filtering which is performed on the ECG signal segment. Here it can be seen from the PSD that the frequencies of 50 Hz,100 Hz and 150 Hz are removed from the ECG signal. This indicates that the notch filters, which are used to remove the powerline interference,are performing as intended. Secondly the good performance of the notch filter, which removes the 120 Hz, can be seen from Figure 6.1, here it can be observed that the oscillations, which are created by the 120 Hz noise component, are clearly removed from the ECG signal. The highpass filter, which is made for the removal of the baseline wander, is also working as intended. Figure 6.1, shows that the baseline wander is removed from the original ECG signal. The lowpass filter performance can be seen from Figures 6.1 and 6.2, by looking at the PSDs of both figures. In these PSDs it can be observed that frequency components higher than 150Hz/40Hz are suppressed in the PSD. This indicates the good performance of the lowpass filter. However noise components such as EMG noise which have frequency that overlap the original ECG could not be removed, because bad processing of these

	1	2	3	4	5	6	7	8	9	10	11	12
RR-interval of segment 71	695	233	872	595	599	606	608	613	611	610	611	612

Table 6.2: The results of segment number 71 from the CinC2017 dataset with their respective differences between R-peak values at the first part. Each column corresponds to the segmentation of the signal segment 71

noises could result in distortion of the ECG signal, and thus are not tackled within the ECG design. The filters designed however work as intended.

When looking at the ECG dataset from CinC2017, the results of 'Labeled as Good while Bad' (LGwB) segments show that 17.5666% is accepted as a good signal while the segments are annotated as bad in the dataset from CinC2017. This is an unwanted result due to the fact that bad signals will be used while further analysis is done, which can result in an inaccurate indication of the patients vital signs. The analysis on the 'Labeled as Bad While Good' (LBwG) 12.3359 % percentage of the data is thrown away by labeling it as a bad signal. This situation is inefficient, because good quality data is thrown away, but can be acceptable because further analysis of the ECG is not disturbed by these ECG signals.

A possible reason that the percentage of LGwB is such high is the strict annotating of the experts. When analyzing the LBwG segments, it is observed that some of the segments looked very noisy and annotated bad by the experts. In figure 6.6(b) the graph of LGwB signal-segment 71 is shown. Here, the observation is that only the first part of the signal-segment 71 is contaminated and the R-peaks can not be detected, while the other part looks like a clean signal. From this, the conclusion is that the experts annotated the whole signal-segment with a bad label because the R-peaks in the first part can not be detected because of noise. The same conclusion can be taken when looking at figure 6.6.(a). The automatic algorithm proposed in this project, detects all possible R-peaks and takes the average of the RR-intervals to compute the average heart rate. Looking at column 2 of the signal-segment 71, a very short RR-interval of 233 can be observed. This is physically not possible, which means that the R peak detection goes wrong at that moment.

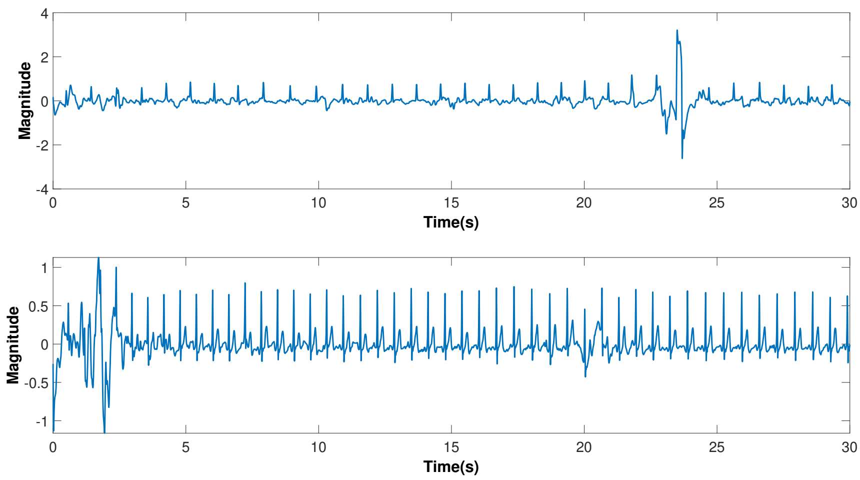

Figure 6.6: The segmented ECG signals from the CinC2017 dataset. The upper figure (a) belongs to segment 23, while the lower figure (b) is from segment 71

The accuracy of the R-peak detection can be improved by conditioning the distance between R-peaks. If the RR-interval is below a value that is physically not possible, the related R-peak that is detected will not be accepted. And if the computed RR-interval is below the set value, the algorithm should expect that this value will hold on for a couple of periods. In this way, the algorithm will be able to distinguish an alarming situation and a wrong R-peak detection.

Many algorithms obtained excellent accuracy when comparing the results of the automated algorithms with export annotations, but none reached perfection. So it is highly likely that some annotations are incorrect labeled[24].

The results of the respiratory signal filtering by use of a bandpass filter can be seen by comparing the Figures 6.4 and 6.5. The only thing that can be concluded from those figure is that the bandpass filter is working as intended. This can be seen by comparing the PSDs from both figures. However, it can not be concluded that respiratory signal at figure 6.5 is of good quality. This is because there is not a general consensus on what a good quality respiratory signal is. This makes it harder to analyse the signal quality indicator which is labeled on the respiratory signal. However some small analyses can still be made by analysing the Driver dataset. It is known that the respiratory has possibly a lower quality when the subject is speaking. Knowing this, it is expected that the respiratory signal is labeled with an SQI zero more often at speaking phases, because talking disturbs the respiration of a subject. However this is not the case. The phases which correspond with the subject talking are : ST, AT. AT phase has the lowest percentage, however story telling has the highest. So the accuracy of the quality indicators of the respiratory system is up to debate. The only quality indicator which could be determined more scientifically is the power check. However as stated earlier, a consensus of a good quality respiratory signal is not yet achieved in literature, so improving this power check is hard without knowing what a good respiratory signal is.

The computation time for one patient is also recorded. The computation of patient 'ecg7' from driver dataset takes in total 18.9021 seconds for a recording of 88.667 minutes long. The computation time for segment has a mean of 0.2844 seconds with a deviation of 0.0615 seconds. Since the system design group computes only the Stress dataset, the computation time for the stress dataset is also observed. For a recording of 33.2 minutes, there was a total delay of 8.2618 seconds, with a mean of 0.0379 seconds and a standard deviation of 0.0128 for the segment computation. Note that is algorithm is runned on a 'Intel core i5-8300H CPU, 2.30GHz' processor.

Chapter 7

Conclusion

This thesis proposes two systems which can be used for the pre-processing of ECG signals and respiratory signals which can be used for further computations to detect stress. These signals are obtained from three different recordings in which the subjects were asked to perform different tasks. The ECG system consists of two main part, which are the filtering and the signal quality part.

The signal quality part consists of a three-step quality check, which is performed at the beginning and after pre-processing of the signal. The signal quality part is based on the following parts: the quantification of the relative power of the ECG within the band of interest, weight computation based on the ACF and a heart rate evaluation. The respiratory signal system out of three main parts. These parts are the filter part and the signal quality part.The signal quality part consists out of two checks. The first one is the quantification of the relative power of the respiratory signal within the band of interest, this is similar to the ECG power check. The second part is the breath check, which checks whether a breath is taken faster than 2 seconds or slower than 10 seconds.The respiratory rate calculation part is the third part of the respiratory signal part and calculates the respiratory rate per respiratory signal segment.

When comparing the results of the algorithm with the the labels from dataset CinC2017, there was a coherence of 70.0975% between the quality labels from the dataset and the ECG system. However 29.9025 % of the of ECG segments did not have a coherence. From this percentage, 17.5666% are data which are labeled as a bad quality ECG segment by the dataset while the ECG system passes it as a good quality signal. The remaining 12.3359% is labeled as good by the dataset, while these segments are labeled as a bad quality segment by the ECG system. 29.9025% is a high amount of segments which are incorrectly labeled by the ECG system. The data with the 17.5666% truly apposes as a problem, because further analysis is done with this data. Thus it can be concluded that the ECG system is working but not as accurate as intended.

The respiratory signal results were not as expected. This is concluded by analysing the Stress dataset, which is one of the three datasets that are used in this thesis. This dataset consist of different phases, in which two phases consist of subjects whom talking. The expectation is that these phases contain more bad quality signals than the phases in which the subject is quiet because talking effects the respiration of a subject. The expectation is true for ST phase which has the highest percentage of bad signals while the AT has the lowest. However a overall consensus of a good quality respiratory signal is not yet achieved in literature, so analysing a signal with a bad quality indicator is a hard task, because it is not well known whether this signal is truly good or bad. This makes much needed improvements on the respiratory signal system hard.

An important contribution of this work is the giving the ability to locate the segments which contaminated by using ACF and graph theory. Furthermore, the algorithm proposed in this project is easy to implement and suitable for on-line environment since it is able to compute in real-time. Moreover, the average computation time is 0.0379 ± 0.0128 seconds, which is fast enough for a real-time recording every 10 seconds.
The system can be improved by computing only ECG and deriving the Respiratory rate from it. This will be an advantage because the limitation of the number of sensors on a wearable system.

Bibliography

[1] Bob Morssink Isar Meijer. “Stress Detection System Using ECG and Respiratory Signals.” In: (2020).

[2] Yavuzhan Mercimek Geert Jan Meppelink. “Thesis System Design of a Telehealth System”. In: (2020).

[3] Carolina Varon et al. “Unconstrained Estimation of HRV Indices After Removing Respiratory Influences From Heart Rate”. In: *IEEE Journal of Biomedical and Health Informatics* 23.6 (2019), pp. 2386–2397.

[4] Alberto Hernando et al. “Inclusion of Respiratory Frequency Information in Heart Rate Variability Analysis for Stress Assessment”. In: *IEEE Journal of Biomedical and Health Informatics* 20.4 (2016), pp. 1016–1025.

[5] Daniel Mcduff, Sarah Gontarek, and Rosalind Picard. “Remote measurement of cognitive stress via heart rate variability”. In: *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (2014). DOI: 10.1109/embc.2014.6944243.

[6] Aswathy Velayudhan and Soniya Peter. “Noise analysis and different denoising techniques of ECG signal-a survey”. In: *IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)* (2016), eISSN–2278.

[7] Mohamed Abdelazez, Sreeraman Rajan, and Adrian D. C. Chan. “Detection of Noise Type in Electrocardiogram”. In: *2018 IEEE International Symposium on Medical Measurements and Applications (MeMeA)* (2018). DOI: 10.1109/memea.2018.8438664.

[8] Gary M. Friesen et al. “A comparison of the noise sensitivity of nine QRS detection algorithms”. In: *IEEE Transactions on Biomedical Engineering* 37.1 (1990), pp. 85–98. DOI: 10.1109/10.43620.

[9] Gari D. Clifford, Francisco Azuaje, and Patrick McSharry. *Advanced methods and tools for ECG data analysis.* Artech House, 2006.

[10] Udit Satija, Barathram Ramkumar, and M. Sabarimalai Manikandan. “Automated ECG Noise Detection and Classification System for Unsupervised Healthcare Monitoring”. In: *IEEE Journal of Biomedical and Health Informatics* 22.3 (2018), pp. 722–732. DOI: 10.1109/jbhi.2017.2686436.

[11] G D Clifford et al. “Signal quality indices and data fusion for determining clinical acceptability of electrocardiograms”. In: *Physiological Measurement* 33.9 (2012), pp. 1419–1433. DOI: 10.1088/0967-3334/33/9/1419.

[12] J. A. Healey and R. W. Picard. “Detecting stress during real-world driving tasks using physiological sensors”. In: *IEEE Transactions on Intelligent Transportation Systems* 6.2 (2005), pp. 156–166.

[13] G. D. Clifford et al. “AF classification from a short single lead ECG recording: The PhysioNet/computing in cardiology challenge 2017”. In: *2017 Computing in Cardiology (CinC)*. 2017, pp. 1–4.

[14] David L Reich. *Monitoring in anesthesia and perioperative care.* Cambridge University Press, 2011.

[15] Paul Kligfield et al. “Recommendations for the standardization and interpretation of the electrocardiogram: part I: the electrocardiogram and its technology a scientific statement from the American Heart Association Electrocardiography and Arrhythmias Committee, Council on Clinical Cardiology; the American College of Cardiology Foundation; and the Heart Rhythm Society endorsed by the International Society for Computerized Electrocardiology”. In: *Journal of the American College of Cardiology* 49.10 (2007), pp. 1109–1127.

[16] John G. Proakis and Dimitris G. Manolakis. *Digital signal processing: Principles, Algorithms and Applications.* Prentice-Hall, 2007.

[17] Abdulhamit Subasi. *Practical guide for biomedical signals analysis using machine learning techniques: a MATLAB based approach.* Academic Press, 2019.

[18] M. Malik et al. “Heart rate variability: Standards of measurement, physiological interpretation, and clinical use”. In: *European Heart Journal* 17.3 (Jan. 1996), pp. 354–381. DOI: 10.1093/oxfordjournals.eurheartj.a014868.

[19] Carolina Varon et al. “Robust artefact detection in long-term ECG recordings based on autocorrelation function similarity and percentile analysis”. In: *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2012, pp. 3151–3154.

[20] Robert Avram et al. “Real-world heart rate norms in the Health eHeart study”. In: *npj Digital Medicine* 2.1 (2019). DOI: 10.1038/s41746-019-0134-9.

[21] Jay W. Mason et al. “Electrocardiographic reference ranges derived from 79,743 ambulatory subjects”. In: *Journal of Electrocardiology* 40.3 (2007). DOI: 10.1016/j.jelectrocard.2006.09.003.

[22] J J Bailey et al. “Recommendations for standardization and specifications in automated electrocardiography: bandwidth and digital signal processing. A report for health professionals by an ad hoc writing group of the Committee on Electrocardiography and Cardiac Electrophysiology of the Council on Clinical Cardiology, American Heart Association.” In: *Circulation* 81.2 (1990), pp. 730–739. DOI: 10.1161/01.cir.81.2.730.

[23] Ary L. Goldberger, Zachary D. Goldberger, and Alexei Shvilkin. “How to Make Basic ECG Measurements”. In: *Goldbergers Clinical Electrocardiography* (2018), pp. 11–20. DOI: 10.1016/b978-0-323-40169-2.00003-2.

[24] Jonathan Moeyersons et al. “R-DECO: an open-source Matlab based graphical user interface for the detection and correction of R-peaks”. In: *PeerJ Computer Science* 5 (2019). DOI: 10.7717/peerj-cs.226.

[25] Jiapu Pan and Willis J. Tompkins. “A Real-Time QRS Detection Algorithm”. In: *IEEE Transactions on Biomedical Engineering* BME-32.3 (1985), pp. 230–236. DOI: 10.1109/tbme.1985.325532.

[26] C. Varon et al. “A Novel Algorithm for the Automatic Detection of Sleep Apnea From Single-Lead ECG”. In: *IEEE Transactions on Biomedical Engineering* 62.9 (2015), pp. 2269–2278.

[27] Rahul Kher. “Signal Processing Techniques for Removing Noise from ECG Signals”. In: *Biomedical Engineering and Research* 1.1 (2017).

[28] Manpreet Kaur and Birmohan Singh. “Comparison of different approaches for removal of baseline wander from ECG signal”. In: *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*. 2011, pp. 1290–1294.

[29] Mindie Mosiman Jessica Fertier and Tim Mila. *Introduction to Filters: FIR versus IIR*. 2020. URL: https://community.sw.siemens.com/s/article/introduction-to-filters-fir-versus-iir.

[30] Li Tan and Jean Jiang. *Digital Signal Processsing(Second Edition)*. 2013, pp. 354–355.

[31] Jonathan Moeyersons et al. “Artefact detection and quality assessment of ambulatory ECG signals”. In: *Computer methods and programs in biomedicine* 182 (2019), p. 105050.

[32] Carolina Varon et al. “A Comparative Study of ECG-derived Respiration in Ambulatory Monitoring using the Single-lead ECG”. In: *Scientific Reports* 10.1 (2020), pp. 1–14.

[33] Christopher Watford. *Understanding ECG Filtering*. 2014. URL: http://ems12lead.com/2014/03/10/understanding-ecg-filtering/#gref.

[34] Laura Mason. *Signal processing methods for non-invasive respiration monitoring*. University of Oxford Oxford, 2002.

[35] Torben Noto et al. “Automated analysis of breathing waveforms using BreathMetrics: a respiratory signal processing toolbox”. In: *Chemical senses* 43.8 (2018), pp. 583–597.

[36] M. Sami Fadali and Antonio Visioli. “Chapter 12 - Practical Issues”. In: *Digital Control Engineering (Second Edition)*. Ed. by M. Sami Fadali and Antonio Visioli. Second Edition. Boston: Academic Press, 2013, pp. 491–531. ISBN: 978-0-12-394391-0. DOI: https://doi.org/10.1016/B978-0-12-394391-0.00012-5. URL: http://www.sciencedirect.com/science/article/pii/B9780123943910000125.

Appendices

Appendix A

Power Spectral Density

Here are two graphs shown. These are the PSD's of raw data. figure A.1 shows incoming clean segment and figure A.2 shows a noisy segment. There is a visible difference in the fluctuations in the range of interest.

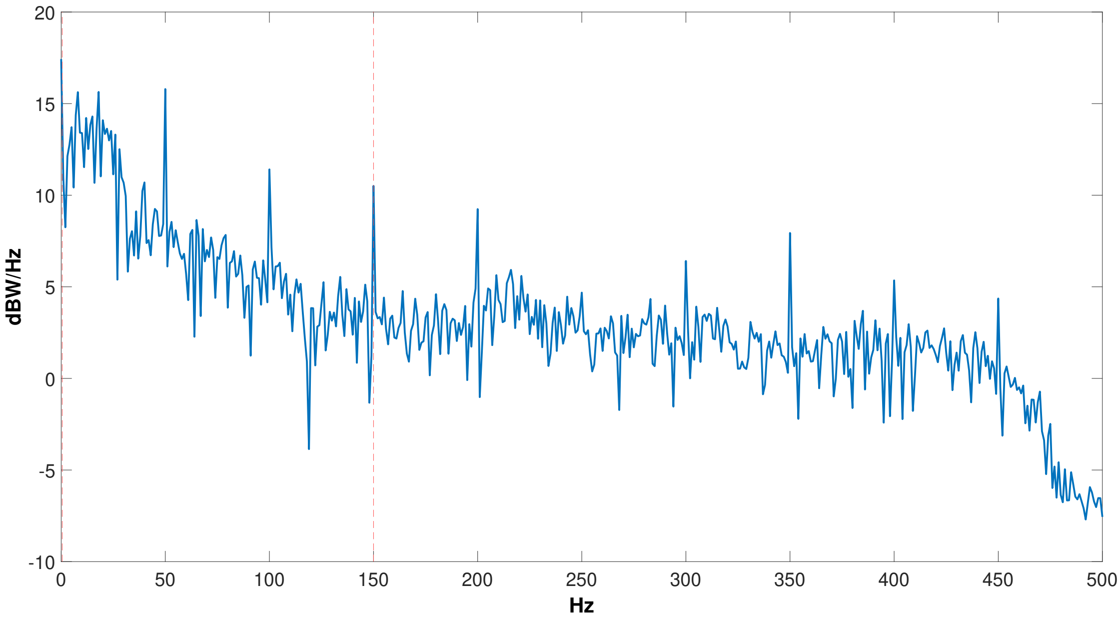

Figure A.1: The PSD of a clean raw segment. Range of interest is between the red dashed lines

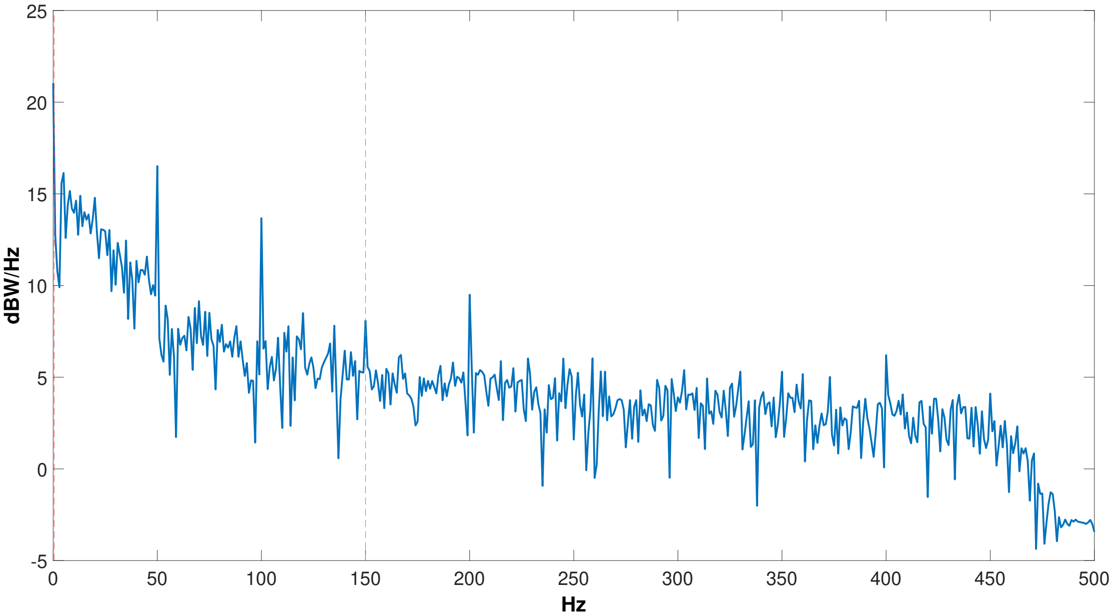

Figure A.2: The PSD of a noisy raw segment. Range of interest is between the red dashed lines

Appendix B

Matlab Code

B.1 ECG system design code

The codes listed in this section are the scrips which are used to make the ECG system. The MainECG.m is used to run the FINALECG.m code which uses the other files given in this section to perform the computations which are described in chapter 4.

B.1.1 MainECG.m

```
%%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% The main of the ECG system. This code executes FINALECG.m by supplying
% the code with its inputs. The ECG signals are doubles which are put in in
% the manner of nx1, in which n is the length of n.
%%

clear all;
load('Patient_signaal.mat')

%% Insert patient number, window of segment and measurement channel
total = tic;
Fs =              1000;           % Sample rate
t =               80;             % Time duration of the ECG segment
Nsegment =        t*Fs;           % Length/Window Segment
Measurement =     1;              % Channel 1 = X, 2 = Y , 3 = Z.
patientdata =     Patientdatabase;     % Initiates the dataset
patientnumber =   23;              % Patient number


parameters = {300, 100, 1, 1, 0}; %envelope size = 300ms, postprocessing = true, ectopic beats removal = true,inverted signal = false


%% Segmentation of the signal
lengthsignal = length(patientdata{1,patientnumber}); %Lengte van de totale signaal
number_segments = floor(lengthsignal/Nsegment); % i zal gebruikt worden in de for-loop

% Function of everything
% Storing/showing one segment
processed_signal = cell(6,1);
```

```
segment_delay = zeros(1, number_segments);

for i = 0:(number_segments - 1)
    [iECG,fECGapp,SQI,R_peak,averageHeartRate,RR_int] = FINALECG(patientdata, patientnumber,Nsegment,Fs,Measurement,i,parameters);
    processed_signal{1,(i+1)} = iECG;   %Assigns the initial signal values to segment
    processed_signal{2,(i+1)} = fECGapp;  % Final signal values to segment which filter higher freq than 40Hz
    processed_signal{3,(i+1)} = SQI;    %Assigns signal quality to segment
    processed_signal{4,(i+1)} = R_peak; %Assigns Rpeak results to segment
    processed_signal{5,(i+1)} = averageHeartRate; %Assigns Rpeak results to segment
    processed_signal{6,(i+1)} = RR_int; %Assigns Rpeak results to segment
end
```

B.1.2 FINALECG.m

```
% FINAL CODE FOR ECG, with filters and SQI.
%%

% Output:
% SQI = Signal quality indicator of the ECG.
% R_peak = Vector which has the R-peaks of the ECG signal which is put in
% the code
% averageHeartRate = The average heart rate which is determined by using
% the ECG signal
% RR_int = Vector which has the RR-intervals of the R_peak vector
% iECG = The segmented ECG signal before pre-processing
% fECGpp = The segmented ECG after pre-processing
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% The FINALECG script uses the scripts of Rpeak.m, SQI_ACF.m , Power.m
% NaNorNot.m and ECGFILTER.m .The NaNorNot function is used to detect possible NaNs in the ECG before making computations.
% Afterward the ECG is segmented and sent to the three-step quality check.
% The first three scrips are which are mentioned are used for the
% three-step quality check. Depending on the SQI, the segmented ECG is sent
% to the filters which are implemented in ECGFILTER.m. Afterwards the signal
% is again put into the three-step quality check and depending on the
% result , the SQI = 0 or SQI = 1. This is sent as output together with the
% other outputs.
%
%

%%

function [iECG,fECGapp,SQI,R_peak,averageHeartRate, RR_int, frac, averageweights] = FINALECG(patientdata, patientnumber,Nsegment,Fs,Measurement,i,parameters,t)
fullpatient = patientdata{1,patientnumber}; %selecting signal patient
segmentpatient = fullpatient((1+(i*Nsegment)):((1+i)*Nsegment), Measurement); % 318149:328148 - Dit zijn samples die NaN bevatten
iECG = detrend(segmentpatient); %Taking away the mean.

Fnotch = 50;            %Powerline Notch frequency
Fnotch2 = 100;          %Powerline Notch frequency
```

```
Fnotch3 = 150;          %Powerline Notch frequency
Order1 = 4;             %Order Butterworth for baseline filter
fcut = 0.5;
Order2 = 14;            %Order low-pass filter of frequencies above 150Hz
fcut2 = 150;
fcut3 = 40;
Order3 = 14;            %Order low-pass filter of frequencies above 40Hz

%------------------ begin ACF code ---------------
% The function ACF_Artefact can filter the signal before
% For ECG signals it is recommended to use a bandpass filter with cutoff
% frequencies at 1Hz and 40Hz
filtro.type = 'HL';
filtro.hf = 40;
filtro.lf = 0.5;

segm = 1; % length of the segments to be analyzed (in seconds)
lags = []; % Lags used in the ACF (default 250ms)
%manual = 1; % 1 in case you want to apply a threshold in the weights
% ------------------ eind ACF code ---------------
%% SQI and filtering.

Nanindicator = 0;
SQI1 = NaNorNot(iECG, Nsegment); %%%% NaN detector
if SQI1 == 1                % If NaN not detected.
    [SQI2] = Power(Fs,iECG, Nsegment);     % First Power check.
    if SQI2 == 0      %If power not good.
        filteredECG = ECGFILTER(Fs,iECG,Fnotch,Fnotch2,Fnotch3,Order1,fcut,Order2,fcut2); % FILTER NA EERSTE POWER SQI = 0;
        [SQI2_1] = Power(Fs,filteredECG, Nsegment);  %If power good for filtered signal.
        if SQI2_1 == 1 %If power after filtering good.
            [SQI2_2,averageweights] = SQI_ACF(Fs,filteredECG,segm,filtro,lags); %ACF with filtered data.
            if SQI2_2 == 1 %If ACF result is good.
                [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters);
                if SQI2_3 == 1 %If Heartbeat result is good.
                    SQI = 1;
                    [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                    fECG = filtfilt(b,a,filteredECG);
                    [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                    fECGapp = filtfilt(b2,a2,fECG);
                else            %If Heartbeat result is bad.
                    SQI = 0;
                    [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                    fECG = filtfilt(b,a,filteredECG);
                    [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                    fECGapp = filtfilt(b2,a2,fECG);
                end
            else
                SQI = 0;
                [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters);
                [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                fECG = filtfilt(b,a,filteredECG);
                [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                fECGapp = filtfilt(b2,a2,fECG);
```

```
            end
        else %If power after filtering not good.
            SQI = 0;
            [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters);
            [b,a] = butter(Order3,fcut3/(Fs/2),'low');
            fECG = filtfilt(b,a,filteredECG);
            [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
            fECGapp = filtfilt(b2,a2,fECG);
        end
    else                %If power original signal is good.
        [SQI2_5,averageweights] = SQI_ACF(Fs,iECG,segm,filtro,lags); %ACF with original data.
        if SQI2_5 == 1 %ACF with original data is good.
            [SQI2_6,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,iECG,parameters); %Rpeak with original signal
            if SQI2_6 == 1 %Rpeak with original signal is good
                fECG = iECG;
                SQI = 1;
                [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                fECG = filtfilt(b,a,iECG);
                [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                fECGapp = filtfilt(b2,a2,fECG);
            else %Rpeak with original signal is bad
                filteredECG = ECGFILTER(Fs,iECG,Fnotch,Fnotch2,Fnotch3,Order1,fcut,Order2,fcut2); % FILTER iECG na Rpeak detection;
                [SQI2_7] = Power(Fs,filteredECG,Nsegment);
                if SQI2_7 == 1
                    [SQI2_8,averageweights] = SQI_ACF(Fs,filteredECG,segm,filtro,lags);
                    if SQI2_8 == 1
                        [SQI2_9,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters);
                        if SQI2_9 == 1
                            SQI = 1;
                            [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                            fECG = filtfilt(b,a,filteredECG);
                            [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                            fECGapp = filtfilt(b2,a2,fECG);
                        else
                            SQI = 0;
                            [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                            fECG = filtfilt(b,a,filteredECG);
                            [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                            fECGapp = filtfilt(b2,a2,fECG);
                        end
                    else
                        [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters);
                        SQI = 0;
                        [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                        fECG = filtfilt(b,a,filteredECG);
                        [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                        fECGapp = filtfilt(b2,a2,fECG);
                    end
                else
                    SQI = 0;
                    [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,
```

```
                    filteredECG,parameters);
                    [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                    fECG = filtfilt(b,a,filteredECG);
                    [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                    fECGapp = filtfilt(b2,a2,fECG);
                end
            end
        else %ACF with original data is bad.
            filteredECG = ECGFILTER(Fs,iECG,Fnotch,Fnotch2,Fnotch3,Order1,fcut,Order2,fcut2); % FILTER iECG na ACF detection;
            [SQI2_10] = Power(Fs,filteredECG,Nsegment);
            if SQI2_10 == 1
                [SQI2_11,averageweights] = SQI_ACF(Fs,filteredECG,segm,filtro,lags);
                if SQI2_11 == 1
                    [SQI2_12,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters,t);
                    if SQI2_12 == 1
                        SQI = 1;
                        [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                        fECG = filtfilt(b,a,filteredECG);
                        [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                        fECGapp = filtfilt(b2,a2,fECG);
                    else
                        SQI = 0;
                        [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                        fECG = filtfilt(b,a,filteredECG);
                        [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                        fECGapp = filtfilt(b2,a2,fECG);
                    end
                else
                    SQI = 0;
                    [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                    [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters, t);
                    fECG = filtfilt(b,a,filteredECG);
                    [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                    fECGapp = filtfilt(b2,a2,fECG);
                end
            else
                SQI = 0;
                [b,a] = butter(Order3,fcut3/(Fs/2),'low');
                [SQI2_3,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,filteredECG,parameters, t);
                fECG = filtfilt(b,a,filteredECG);
                [b2,a2] = butter(Order3,fcut3/(Fs/2),'low');
                fECGapp = filtfilt(b2,a2,fECG);
            end
        end
    end

else  %%%% If NaN detected.
    Nanindicator = 1;
    SQI = 0;
    R_peak = 0;
    averageHeartRate =0;
    averageweights = 0;
    RR_int = 0;
```

```
    frac = frac;
    fECG = segmentpatient;
    fECGapp = fECG;
end

end
```

B.1.3 NaNorNot.m

```
%%

% Output:
% SQI_NaN - signal quality indicator which is an indication of the possible
    presence of a NaN/NaNs.


% Author(s): Enes Kinaci
%            Talha Kuruoglu

%This algoritm checks whether the incoming ECG signal has a NaN or NaNs inside
    of it.
%Depending on the presence of a NaN or NaNs, a SQI is sent as output.
%SQI = 1, means that a NaN is detected, while SQI = 0, means that no NaNs are
    detected.
%%

% NaN detector
function SQI_NaN = NaNorNot(iECG, Nsegment)
i =1;
SQI_NaN =1;
while (i <= Nsegment && SQI_NaN == 1)
    Index = isnan(iECG(i:1));
    if Index == 1;
        SQI_NaN = 0;
        i = i+1;
    else
        SQI_NaN =1;
        i = i+1;
    end

end
```

B.1.4 Power.m

```
% Powerindicator
%%

% Output:
% SQI_power = Signal quality indicator after the power quality check
%
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% This algorithm is the contains the power quality check in which the percentage
    of the power in the band of interest of the
% ECG is compared with its whole spectrum. Depending on this percentage, a
    SQI_power = 0 or SQI_power = 1 is send as output.

```

```

%%

function [SQI_power] = Power(Fs,iECG,Nsegment)

%% input
windowlength =  Nsegment;          %Length of the window
overlap =       Nsegment*0.5;      %number of overlap samples
ndft    =       Fs;                %number of DFT points
%% Plotting the PSD using Welch Method

[pxx1,f] = pwelch(iECG,windowlength,overlap,ndft,Fs); % the power components with their frequencies are calculated

%% Calculating the total power

P0 = trapz(pxx1); %Whole spectrum power
pxx1ECG = pxx1(1:150); % Calculating area of the ECG interest band
PECG = trapz(pxx1ECG);%Calculating area of the whole power spectrum
frac = ((PECG)/P0)*100; % determining the percentage of power in the ECG interest band compared to the whole spectrum.

if (frac > 99.997)
    SQI_power = 1;
else
    SQI_power = 0;
end

end
```

B.1.5 Rpeak.m

```
% Rpeak detection SQI

% Output:
% SQI = Signal quality indicator which is obtained from the heartbeat check.
% R_peak = Vector which has the R-peaks of the ECG signal which is put in
% the code
% averageHeartRate = The average heart rate which is determined by using
% the ECG signal
% RR_int = Vector which has the RR-intervals of the R_peak vector
%
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% This algorithm detects the R-peaks and the corresponding RR-intervals of the incoming ECG by using the function RRpeak.
% Afterwards a mean of the RR-intervals is taken to calculate the averageHeartRate. A check is performed afterwards to determine if the averageHeartrate
% is in the range of 40 or 180 bpm. If it is then the SQI = 0 afterwards
% its not the SQI = 1.
%%
function [SQI,R_peak,averageHeartRate, RR_int] = Rpeak(Fs,ECG,parameters)
 [R_peak,RR_int] = RRpeak(parameters,ECG,Fs); % Calculation of the R-peaks and RR-intervals.
 avgRR= mean(RR_int{1,1}); % average RR-interval

```

```
averageHeartRate = 60/(avgRR/1000); % averageHeartRate is calculated in beats per minute(bpm).


if ((averageHeartRate < 40)||(averageHeartRate > 180))
    SQI = 0;
else
    SQI = 1;

end
```

B.1.6 RRpeak.m

```
function [R_peak,RR_int] = RRpeak(parameters,signal,fs)
% Output:
% R_peak      - Location of the R-peaks in samples
% RR_int      - Intervals between the R-peaks in ms
% check       - Check if the method was run correctly
%
% Author(s):     Jonathan Moeyersons       (Jonathan.Moeyersons@esat.kuleuven.be)
%                Sabine Van Huffel         (Sabine.Vanhuffel@esat.kuleuven.be)
%                Carolina Varon            (Carolina.Varon@esat.kuleuven.be)
%
% Version History:
% - 06/05/2019   JM      Initial version
%
% Copyright (c) 2019,  Jonathan Moeyersons, KULeuven-ESAT-STADIUS
%
% This software is made available for non commercial research purposes only
% under the GNU General Public License. However, notwithstanding any
% provision of the GNU General Public License, this software may not be
% used for commercial purposes without explicit written permission after
% contacting jonathan.moeyersons@esat.kuleuven.be
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <https://www.gnu.org/licenses/>.


% Paramaters is a cell containing:
% {envelope size [ms], heart rate [bpm], postprocessing [bool], ectopic beat
% removal [bool], inverted signal [bool]}


% Get the size of the signal
[original_signal_length,nr_ch] = size(signal);

% Pre-allocate the R-peak and RR-interval variable
R_peak = cell(1,nr_ch);
```

```
RR_int = cell(1,nr_ch);


env = round(fs*parameters{1}/1000);
avgHR = parameters{2};
postproc = parameters{3};
ect = parameters{4};
inverted = parameters{5};

% Check if the signal is inverted, if so, take action
if inverted
    signal = -signal;
end

% Set the segment size to one minute and compute the amount of minutes in
% the signal
segm_length = 60*fs;
nr_segm_original = floor(original_signal_length/segm_length);

% Add zeros if the signal does not contain a round number of minutes
if nr_segm_original ~= 0
    signal = [signal; zeros(segm_length-(original_signal_length-nr_segm_original*segm_length),nr_ch)];

    % Get the new length and amount of segments
    [new_signal_length,~] = size(signal);
    nr_segm_new = floor(new_signal_length/segm_length);
else
    % Get the new length and amount of segments
    [new_signal_length,~] = size(signal);
    nr_segm_new = 1;
end

% Pre-allocate the R-peaks per channel
R_peak_ch = cell(nr_segm_new);

% Loop over the channels
for ii = 1:nr_ch
    %% Segmentize the signals
    avg = mean(signal(:,ii));
    s = std(signal(:,ii));
    if nr_segm_new > 1
        segmented_ecg = (double(reshape(double(signal(:,ii)),segm_length,nr_segm_new)));
%             segmented_ecg = segmented_ecg-(repmat(avg,size(segmented_ecg,1),size(segmented_ecg,2)));
%             segmented_ecg = segmented_ecg./(repmat(s,size(segmented_ecg,1),size(segmented_ecg,2)));
    else
        segmented_ecg = (signal(:,ii)-avg)/s;
    end

    % Go through each segment
    for iii = 1:nr_segm_new
        %% Make the segments five seconds longer on each side
        if nr_segm_new > 1
            if iii == 1
                % End the segment five seconds later
```

```
                segm = [segmented_ecg(:,1); segmented_ecg(1:fs*5,2)];
            elseif iii==nr_segm_new
                % Start the segment five seconds earlier
                segm = [segmented_ecg(end-(fs*5)+1:end,iii-1); segmented_ecg(1:end-(new_signal_length-original_signal_length),iii)];
            else
                % Start the segement five seconds earlier and end five seconds later
                segm = [segmented_ecg(end-(fs*5)+1:end,iii-1);segmented_ecg(:,iii);segmented_ecg(1:fs*5,iii+1)];
            end
        else
            % Take the whole segment
            segm = segmented_ecg;
        end

        %% Envelope the signal
        % Get the time
        time = (1:length(segm))/fs;

        % Upper envelope
        up_env = env_secant(time,segm,env,'top');

        % Lower envelope
        low_env = env_secant(time,segm,env,'bottom');

        % Enveloped segment
        env_segm = up_env-low_env;

        %% Detect peaks in the enveloped segment
        % Get the frequency ratio with 250 Hz
        freq_ratio = fs/250;

        % Get the envelope ratio
        env_ratio = env/(0.15*fs);

        % Peaks are detected by checking whether the derivative is positive and
        % if the peaks last long enough (step size). The smaller the step
        % size, the more sure that all R-peaks are detected, but also the
        % more non-R-peaks are detected. To avoid local minima, a step
        % (~=1) is used. The original step size is 20 samples = 80ms
        % for an envelope size of 150ms and a sampling frequency of
        % 250 Hz.

        all_peaks = calc_pos_opt(env_segm,floor(20*freq_ratio*env_ratio),1);

%               figure; plot(env_segm); hold on; scatter(all_peaks,env_segm(all_peaks));

        % Only continue if enough peaks have been detected
        if length(all_peaks) > time(end)/2
            % Adaptive thresholding
            [R_peak_segm, RR_int_segm, ~] = adaptive_thresholding(all_peaks, env_segm, fs, avgHR, time);

%                   figure; plot(env_segm); hold on; scatter(R_peak_segm,env_segm(R_peak_segm));

```

```
            if postproc && length(RR_int_segm) > 5
                % Post-process the RR-intervals
                [R_peak_segm] = post_processing(all_peaks, R_peak_segm, RR_int_segm, env_segm, fs);
            end
        else
            % Set the R and RR variables empty
            R_peak_segm = [];
        end

        % Remove excessive R-peaks on both sides
        if length(R_peak_segm) > time(end)/2
            if nr_segm_new > 1
                if iii == 1
                    % Remove the last five seconds
                    R_peak_segm = R_peak_segm(R_peak_segm<=segm_length);
                elseif iii == nr_segm_new
                    % Remove the first five seconds
                    R_peak_segm = R_peak_segm(R_peak_segm>fs*5)-(fs*5);
                else
                    % Remove the first and last five seconds
                    R_peak_segm = R_peak_segm(R_peak_segm>fs*5)-fs*5;
                    R_peak_segm = R_peak_segm(R_peak_segm<=segm_length);
                end
            end
            R_peak_segm = reshape(R_peak_segm,1,length(R_peak_segm));
        else
            R_peak_segm = [];
        end

         % Store the R-peaks of this channel in a cell
         R_peak_ch{iii} = unique(R_peak_segm);
    end

    %% Merge segments
    R_peak_temp = [];

    % Loop over the segments
    for iii = 1:nr_segm_new
        R_peak_temp = [R_peak_temp,R_peak_ch{iii}+segm_length*(iii-1)]; %#ok
    end

    %% R-peak correction in the ECG signal
    % Define window width (originally this was equal to the envelope size)
    window = round(0.065*fs);

    R_peak_temp = peaks_in_ecg(signal(:,ii),R_peak_temp,window);

    % Take only the unique R-peaks
    R_peak_temp = unique(R_peak_temp);

    % Get the RR-intervals in ms
    RR_int_temp = 1000*(diff(R_peak_temp)/fs);

    if ~isempty(R_peak_temp)
        %% Correct the R-peaks for ectopics
        if ect
            [R_peak_temp,~,~] = ectopic_detection_correction(fs,RR_int_temp,
```

```
                R_peak_temp);
        end
    end

    %% Store the R-peaks and RR-intervals
    if ~isempty(R_peak_temp)
        R_peak{ii} = round(unique(R_peak_temp));
        RR_int{ii} = 1000*(diff(R_peak{ii})/fs);
    else
        R_peak{ii} = [];
        RR_int{ii} = [];
    end
end

end




%%%%% Functions
function [R_peak_segm, RR_int_segm, RR_avg] = adaptive_thresholding(peaks,signal
    ,fs,avgHR,time)

    % Get the standard deviation of the enveloped signal
    STD = std(signal);

    % Set the average RR-interval
    RR_avg = zeros(1,length(peaks)-2);
    RR_avg(1) = 60000/avgHR;

%     % Set the average signal and noise peak
%     peak_avg = 0.7*trimmean(signal(peaks),20);
%     noise_avg = peak_avg/2;

    peak_avg = 0.7*(median(signal(peaks))+mean(signal(peaks)))/2;
    noise_avg = 0.3*(median(signal(peaks))+mean(signal(peaks)))/2;

    % Set thresholds
    peak_threshold = noise_avg + 0.25*(peak_avg-noise_avg);
    noise_threshold = peak_threshold/2;

    % Pre-allocate
    R_peak_segm = zeros(1,length(peaks));
    RR_int_segm = zeros(1,length(peaks)-1);

    % Select the first R-peak
    count = 1;
    while R_peak_segm(1) == 0
        if signal(peaks(count)) > noise_threshold
            R_peak_segm(1) = peaks(count);
        else
            count = count + 1;
        end
    end

    % Start the peak counter
    peak_counter = 1;

```

```
    % Loop over all peaks, starting from the second
    for idx = count + 1 : length(peaks)

        % Select the peak
        peak = signal(peaks(idx));

        % Define the lower limit of the RR-interval
        RR_lower_limit = 0.65*RR_avg(idx-1);
        if RR_lower_limit < 200
            RR_lower_limit = 200; % Lower limit for RRI is 200 ms --> Includes every condition
        end

        % Define the upper limit of the RR-interval
        RR_upper_limit = 1.35*RR_avg(idx-1);
        if RR_upper_limit > 1600
            RR_upper_limit = 1600; % Upper limit for RRI is 1600 ms
        end

        % Determine whether the peak is a signal or a noise peak
        if peak >= peak_threshold % Peak is an R-peak

            % Make the peak amplitude smaller if it is too high
            % NOTE: This was previously done before the peak
            % search, but this might cause the location of the
            % peak to deviate from the actual location.
            if peak > 4*STD
                peak = 4*STD;
            end

            % Adjust the peak average
            peak_avg = 0.125*peak + 0.875*peak_avg;

            % Add signal peak
            peak_counter = peak_counter + 1;
            R_peak_segm(peak_counter) = peaks(idx);

            % Adjust RR interval
            RR_int_segm(peak_counter-1) = 1000*(R_peak_segm(peak_counter)-R_peak_segm(peak_counter-1))/fs; % In ms

        elseif peak <= noise_threshold % Peak is a noise peak
            % Adjust the noise average
            noise_avg = 0.125*peak + 0.875*noise_avg;

        else % searchback procedure
            if peak_counter > 1

                % Check if the candidate RR-interval is within
                % the boundaries of the lower and upper
                % RR-limits with the previous R-peak
                if time(peaks(idx)) >= RR_lower_limit/1000+time(R_peak_segm(peak_counter))...
                        && time(peaks(idx)) <= RR_upper_limit/1000+time(R_peak_segm(peak_counter))

                    % Check if we are at the end, if the next peak is a
                    % signal peak, if it is a signal peak, check the time
```

```
                        % interval
                        if idx == length(peaks) || signal(peaks(idx+1)) <
                            peak_threshold || (signal(peaks(idx+1)) >= peak_threshold
                             && time(peaks(idx+1))-time(peaks(idx)) > 0.2)
                            % Adjust the peak average
                            peak_avg = 0.25*peak + 0.75*peak_avg;

                            % Add R-peak
                            peak_counter = peak_counter + 1;
                            R_peak_segm(peak_counter) = peaks(idx);

                            % Adjust RR interval
                            RR_int_segm(peak_counter-1) = 1000*(R_peak_segm(
                                peak_counter)-R_peak_segm(peak_counter-1))/fs; % In
                                ms

                        else % Peak is a noise peak
                            % Adjust the noise average
                            noise_avg = 0.125*peak+0.875*noise_avg;
                        end

                    else % Peak is a noise peak
                        % Adjust the noise average
                        noise_avg = 0.125*peak+0.875*noise_avg;
                    end
                else % Peak is a noise peak
                    % Adjust the noise average
                    noise_avg = 0.125*peak+0.875*noise_avg;
                end
            end

            % Adjust the RR-average
            if peak_counter > 9
                RR_avg(idx) = mean(RR_int_segm(peak_counter-8:peak_counter-1));
            elseif peak_counter > 1
                RR_avg(idx) = 0.5*RR_avg(idx-1)+0.5*median(RR_int_segm(1:
                    peak_counter-1));
            end

            % Adjust thresholds
            peak_threshold = noise_avg + 0.25*(peak_avg-noise_avg);
            noise_threshold = peak_threshold/2;
        end

        % Get the actual R-peaks
        R_peak_segm = R_peak_segm(1:peak_counter);

        % Get the actual RR-intervals, in ms
        RR_int_segm = RR_int_segm(1:peak_counter-1);

%       figure;
%       plot(signal)
%       hold on;
%       scatter(R_peak_segm,signal(R_peak_segm))
end

function R_peak_segm = post_processing(all_peaks, R_peak_segm, RR_int_segm,
    signal, fs)
```

```
    % To check whether RRI is correct. This will be done by comparing
    % the RR-interval with the mean of the 3 past RR-intervals. A past
    % RR-interval should not be a 'problem' interval.

    % Set basic variables
    a = 0.3; % 30% difference
    d = 0.5; % 50% difference
    max_k = 6; % Maximum ammount of previous RR-intervals

    % Get the amount of RR-intervals
    len = length(RR_int_segm);

    % Pre-allocate
    large_prob = zeros(1,len); % Index of large problems
    RRref_all = zeros(1,len); % Reference RR intervals

    % Set first RR-reference
    RR_ref = trimmean(RR_int_segm(1:5),50);
    RRref_all(1) = RR_ref;

    % Set loop start
    counter = 1;

    % Loop over the different RR-intervals
    while counter < len-1
        % Define the RR-ref if you are further than the first RR-interval
        if counter > 2

            % Set variables
            k = 1; % Number of previous RR-intervals
            sum_ref = 0; % Sum of the good previous RRI's
            num = 0; % Number of good previous RRI's
            wght = [0.5 0.3 0.2]; % Weights: the farther away, the lower the weight

            while counter-k > 0 && num < 3 && k < max_k

                % Check if the previous RRI is a big problem
                if large_prob(counter-k) == 1
                    prob_check = 1;
                elseif large_prob(counter-k) == 0
                    prob_check = 0;
                end

                % If this is not the case, use it for reference
                if ~prob_check
                    num = num + 1;
                    sum_ref = sum_ref + wght(num)*RR_int_segm(counter-k);
                end
                k = k+1;
            end

            % If no good previous RR-intervals are found, this probably means that
            % this is the new normal, so just take the three previous
            if num <= 1
                k = 1;
                while counter-k > 0 && num < 3
```

```
                    num = num + 1;
                    sum_ref = sum_ref + wght(num)*RR_int_segm(counter-k);
                    k = k+1;
                end
            end

            % Define the reference variables
            RR_ref = sum_ref/sum(wght(1:num));
            no_RR_ref = nnz(RRref_all);
            RRref_all(no_RR_ref+1) = RR_ref;
        end

        % Check for too small RR-intervals
        if RR_int_segm(counter) < (1-a)*RR_ref || RR_int_segm(counter) < 200
            % Set looping variables
            counter1 = 1;
            counter2 = 0;
            test = 1;

            % Get the sum with the previous RR-interval
            try
                RRnew1 = RR_int_segm(counter) + RR_int_segm(counter-1);
            catch
                RRnew1 = [];
            end

            % Get the sum with the next RR-interval
            sumRR = RR_int_segm(counter) + RR_int_segm(counter + counter1);
            RRnew2 = sumRR;

            % Loop until a good summation is found
            while test == 1

                % Only proceed if we are not at the end of the signal
                if counter + counter1 < len
                    % If the new RR-interval is still too small
                    if sumRR < (1-a)*RR_ref || sumRR < 200
                        % Adjust counter
                        counter1 = counter1 + 1;

                        % Add the next RR-interval
                        sumRR = sumRR + RR_int_segm(counter+counter1);

                    % Good summation
                    elseif abs(sumRR-RR_ref) < a*RR_ref
                        % Set new looping variable
                        test2 = 1;

                        % Adjust the second counter
                        counter2 = counter2 + 1;

                        % Loop until the best summation is found
                        while test2 == 1

                            % Only proceed if we are not at the end of the
                            % signal
                            if counter + counter1 + counter2 < len

```

```
                % Add the next RR-interval
                sumRR2 = sumRR + RR_int_segm(counter + counter1
                    + counter2);

                % Compute the difference of both sums with the
                % RR-reference
                diff1 = abs(RR_ref-sumRR) / RR_ref;
                diff2 = abs(RR_ref-sumRR2) / RR_ref;

                % If the initial sum is best
                if diff1 < diff2
                    % Stop both loops
                    test = 0;
                    test2 = 0;

                    % Store the initial sum as new RR-interval
                    RRnew2 = sumRR;

                    % Adjust the second counter (necessary for
                    % later)
                    counter2 = counter2 -1;

                % If the second sum is best
                else
                    % Redefine the initial sum and go through
                    % the loop again
                    sumRR = sumRR2;

                    % Adjust the counter
                    counter2 = counter2+1;

                    % Stop the loop if we are at the end of the
                    % signal
                    if counter + counter1 + counter2 > len
                        test = 0;
                        test2 = 0;
                        RRnew2 = sumRR2;
                    end
                end
            else
                % Stop the loop
                test = 0;
                test2 = 0;
                RRnew2 = sumRR;
            end
        end
    % Too big
    else
        % Stop the loop
        test = 0;

        % Select the previous sum as correct
        RRnew2 = sumRR - RR_int_segm(counter + counter1);

        % Adjust the first counter (necessary for later)
        counter1 = counter1 - 1;
    end
else
```

```
                % Stop loop
                test = 0;
                RRnew2 = sumRR;
            end
        end

        % If no better option has been found,
        % or if the new option is better than the
        % combination of the previous RR-intervals

        if ~isempty(RRnew1) && abs(RRnew1 - RR_ref) < abs(RRnew2 - RR_ref) && abs(RRnew1-RR_ref) < a*RR_ref
            % Set RRnew
            RRnew = RRnew1;

            % Replace RR-interval
            RR_int_segm(counter - 1) = RRnew1;

            % Change RR
            RR_int_segm(counter) = [];

            % Change large prob vector (has equal length as RR)
            large_prob(counter) = [];

            % Change RRref_all (is one smaller than RR)
            if counter + 1 < len-1
                RRref_all(counter) = [];
            end

            % Get new length
            len = length(RR_int_segm);

            % Make sure that the new interval is checked
            counter = counter-1;
        else
            % Set RRnew
            RRnew = RRnew2;

            % Replace RR-interval
            RR_int_segm(counter) = RRnew;

            % Change the variables depending on circumstances
            if counter1 >= 1
                % Change RR
                RR_int_segm(counter + 1 : counter + counter1 + counter2) = [];

                % Change large prob vector (has equal length as RR)
                large_prob(counter + 1 : counter + counter1 + counter2) = [];

                % Change RRref_all (is one smaller than RR)
                if counter+1 < len-1
                    RRref_all(counter+1:min([counter+counter1+counter2, len-1])) = [];
                end

                % Get new length
```

```
                len = length(RR_int_segm);
            end
        end


        % Check if the interval is a big problem
        if abs(RRnew-RR_ref)/RR_ref > d || RRnew < 200 % Check for big problems
            large_prob(counter) = 1;
        end

    % Check for too big RR-intervals
    elseif RR_int_segm(counter) > (1+a)*RR_ref || RR_int_segm(counter) > 1600

        % Convert RR-intervals to samples
        RR_samples = RR_int_segm*fs/1000;

        % Define R-peak positions
        R_peak_positions = cumsum(RR_samples(1:counter)) + R_peak_segm(1);

        % Define beginning and end of the interval
        if counter > 1
            begin_int = R_peak_positions(end-1);
        else
            begin_int = R_peak_segm(1);
        end
        end_int = R_peak_positions(end);

        % Get the first peak that is bigger than the beginning of the
        % interval
        test_peak = all_peaks(all_peaks > begin_int);
        test_peak = test_peak(test_peak < end_int);

        if length(test_peak) > 1
            % Get the test RR interval
            RR_test = 1000*(test_peak-begin_int)/fs;

            % Find the smallest difference with the
            % reference RR interval
            [~,I] = min(abs(RR_test-RR_ref));
            test_peak = test_peak(I);
        end

        % See if that peak is before the end of the
        % interval and not too small
        if ~isempty(test_peak) &&...
                signal(test_peak)                  > prctile(signal,50)  && ...
                (((test_peak-begin_int)/fs)*1000) > (1-a)*RR_ref &&...
                (((test_peak-begin_int)/fs)*1000) > 200 &&...
                (((end_int-test_peak)/fs)*1000)    > (1-a)*RR_ref  && ...
                (((end_int-test_peak)/fs)*1000)    > 200

            % Add an RR interval
```

```
                    if counter > 1
                        RR_int_segm = [RR_int_segm(1:counter-1) (((test_peak-
                            begin_int)/fs)*1000) (((end_int-test_peak)/fs)*1000)
                            RR_int_segm(counter+1:end)];
                    elseif counter+1 > len
                        RR_int_segm = [RR_int_segm(1:counter-1) (((test_peak-
                            begin_int)/fs)*1000) (((end_int-test_peak)/fs)*1000)];
                    else
                        RR_int_segm = [(((test_peak-begin_int)/fs)*1000) (((end_int-
                            test_peak)/fs)*1000) RR_int_segm(counter+1:end)];
                    end

                    % Get new length
                    len = length(RR_int_segm);

                    % Adjust the large problem variable
                    large_prob = [large_prob 0]; %#ok

                    % Adjust the RR-ref all variable
                    RRref_all = [RRref_all 0]; %#ok

                    % Make sure that the new interval is checked
                    counter = counter-1;

                else
                    % State that the interval is a large problem and should not
                    % be included for the reference intervals
                    large_prob(counter) = 1;
                end
            end

            % Go one peak further
            counter = counter + 1;
        end

        % Define the R-peak positions
        RR_int_segm = RR_int_segm*fs/1000; % Go back to samples
        R_peak_segm(2:length(RR_int_segm)+1) = round(cumsum(RR_int_segm) +
            R_peak_segm(1));
    end


    function R_peak_temp = peaks_in_ecg(signal,R_peak_temp,window)

    %     % Get the percentage of positive R-peaks
    %     perc_pos = sum(sign(signal(R_peak_temp)))/length(R_peak_temp);

        for idx = 1:length(R_peak_temp)
            % Get the samples from the R-peak minus the envelope length
            % to the R-peak

            p = signal(max(1, R_peak_temp(idx) - window) : R_peak_temp(idx));

            % Do the same but for the time locations
            t = max(1,R_peak_temp(idx)-window) : R_peak_temp(idx);

            % Search for the maximum in p
            [~,ip] = max(p);
```

```
%           % Search for the extremum in p
%           if perc_pos >= 0.5
%               [~,ip] = max(p);
%           else
%               [~,ip] = min(p);
%           end

          % Adjust the R-peak
          try
              if signal(t(ip(end))) >= signal(t(ip(end))-1) && signal(t(ip(end))) >= signal(t(ip(end))+1)
                  R_peak_temp(idx) = t(ip(end));
              end
          catch
              R_peak_temp(idx) = t(ip(end));
          end
      end
end
```

B.1.7 RRPeakCalc.m

```
function [R_peak,RR_int] = RRPeakCalc(DATA, fs_ecg)
%RRPEAKCALC Summary of this function goes here


    % envelope size = 300ms
    % heart rate [bpm] = 100
    % postprocessing = true
    % ectopic beats removal = true
    % inverted signal = false
    parameters = {300, 75, 1, 1, 0};

    [R_peakTEMP(:,1), RR_intTEMP(:,1)] = RRpeak(parameters, DATA{1}, fs_ecg);
    try
        [R_peakTEMP(:,2), RR_intTEMP(:,2)] = RRpeak(parameters, DATA{2}, fs_ecg);
    catch E
        [R_peakTEMP(:,2), RR_intTEMP(:,2)] = RRpeak(parameters, DATA{1}, fs_ecg);
    end

    parameters = {300, 75, 1, 1, 1};
    [R_peakTEMP(:,3), RR_intTEMP(:,3)] = RRpeak(parameters, DATA{3}, fs_ecg);


    if(length(R_peakTEMP{1}) == length(R_peakTEMP{2}) && length(R_peakTEMP{1}) == length(R_peakTEMP{3}))
        RR_int = floor(RR_intTEMP{1}./3 + RR_intTEMP{2}./3 + RR_intTEMP{3}./3);
        R_peak = floor(R_peakTEMP{1}./3 + R_peakTEMP{2}./3 + R_peakTEMP{3}./3);
    else
        if(length(R_peakTEMP{1}) == length(R_peakTEMP{2}))
            RR_int = floor(RR_intTEMP{1}./2 + RR_intTEMP{2}./2);
            R_peak = floor(R_peakTEMP{1}./2 + R_peakTEMP{2}./2);

        elseif(length(R_peakTEMP{2}) == length(R_peakTEMP{3}))
            RR_int = floor(RR_intTEMP{2}./2 + RR_intTEMP{3}./2);
            R_peak = floor(R_peakTEMP{2}./2 + R_peakTEMP{3}./2);

```

```
        elseif(length(R_peakTEMP{1}) == length(R_peakTEMP{3}))
            RR_int = floor(RR_intTEMP{1}./2 + RR_intTEMP{3}./2);
            R_peak = floor(R_peakTEMP{1}./2 + R_peakTEMP{3}./2);
        else


            % calculate average of the lengths
            avg = mean([length(RR_intTEMP{1}) length(RR_intTEMP{2}) length(RR_intTEMP{3})]);

            % calculate difference between average for each length
            dfc = [length(RR_intTEMP{1}) length(RR_intTEMP{2}) length(RR_intTEMP{3})] - avg;

            % get index of the one that is furthest away
            [~,index] = max(abs(dfc));

            % remove that one
            indices = [1 2 3];
            indices(index) = [];


            % get the smallest length of the two
            minLength = min([length(RR_intTEMP{indices(1)}) length(RR_intTEMP{indices(2)})]);

            % average the interval of the final two
            RR_int = floor(RR_intTEMP{indices(1)}(1:minLength)./2 + RR_intTEMP{indices(2)}(1:minLength)./2);

            % get the smallest length of the two
            minLength = min([length(R_peakTEMP{indices(1)}) length(R_peakTEMP{indices(2)})]);

            % average the interval of the final two
            R_peak = floor(R_peakTEMP{indices(1)}(1:minLength)./2 + R_peakTEMP{indices(2)}(1:minLength)./2);

        end


    end

%
%     plot(DATA{1})
%
%     input('press to continue', 's');

end
```

B.1.8 calc-pos-opt.m

```
function [pos_opt] = calc_pos_opt(signal,step,maxmin)
% Calculation of optima
%
% Input:    signal      signal
%           step        stepsize according to application
%           maxmin      for finding maxima: maxmin = 1
%                       for finding minima: maxmin = -1
```

```
% Output:    pos_opt      positions of the optima
%
% Author(s):     Jonathan Moeyersons        (Jonathan.Moeyersons@esat.kuleuven.be)
%                Sabine Van Huffel          (Sabine.Vanhuffel@esat.kuleuven.be)
%                Carolina Varon             (Carolina.Varon@esat.kuleuven.be)
%
% Version History:
% - 06/05/2019   JM      Initial version
%
% Copyright (c) 2019,  Jonathan Moeyersons, KULeuven-ESAT-STADIUS
%
% This software is made available for non commercial research purposes only
% under the GNU General Public License. However, notwithstanding any
% provision of the GNU General Public License, this software may not be
% used for commercial purposes without explicit written permission after
% contacting jonathan.moeyersons@esat.kuleuven.be
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <https://www.gnu.org/licenses/>.

% Get the length of the signal
sze = length(signal);

%% Select the upward slopes

% Pre-allocate
difference = zeros(sze-1,1);

for ii = 1:sze-step
    % If the amplitude of the selected sample is smaller than the amplitude
    % of the selected sample plus the step size
    if (signal(ii+step)-signal(ii))*maxmin > 0

        difference(ii,1) = 1;
    else
        difference(ii,1) = 0;
    end
end

% time = 1:length(signal);
% figure; plot(signal); hold on; scatter(time(difference==1),signal(difference==1))

%% Select the optima

% Pre-allocate
pos_opt = zeros(1,sze-step);
no_opt = 0;
```

```

for ii = 2:sze-step %step/2+1:sze-step
    real_opt = 1;

    % If you are at the end of the upwards slope
    if difference(ii-1,1) == 1 && difference(ii,1) == 0
        count = 1;

        % Check if the upward slope lasts long enough
        while real_opt == 1 && count < step && count < ii
            if difference(ii-count,1) == 1
                real_opt = 1;
            else
                real_opt = 0;
            end
            count = count + 1;
        end

        % If the upward slope is long enough
        if real_opt == 1 && count >= 0.75*step

            % Select the interval containing the peak
            interval = ii : ii + step;

            % Select the extremum in that interval
            if maxmin == 1
                [~,index] = max(signal(interval));
            else
                [~,index] = min(signal(interval));
            end

            % Select the position of the peak
            no_opt = no_opt + 1;
            pos_opt(no_opt) = interval(index);
        end
    end
end

pos_opt = pos_opt(1:no_opt);

% figure
% plot(signal);
% hold on
% plot(pos_opt,signal(pos_opt),'ro')
% title('Optima')
```

B.1.9 ectopic-detection.m

```
function [ect_loc] = ectopic_detection(Rpos)
% Input
% Rpos - Position of the R peaks
%
% Output
% ect_loc  - Position of the ectopic peaks. Used for respiration analysis
%
% Author(s):     Jonathan Moeyersons        (Jonathan.Moeyersons@esat.kuleuven.be)
%                Sabine Van Huffel          (Sabine.Vanhuffel@esat.kuleuven.be)
%                Carolina Varon             (Carolina.Varon@esat.kuleuven.be)
%
```

```
% Version History:
% - 17/02/2019   JM      Initial version
%
% Copyright (c) 2019,  Jonathan Moeyersons, KULeuven-ESAT-STADIUS
%
% This software is made available for non commercial research purposes only
% under the GNU General Public License. However, notwithstanding any
% provision of the GNU General Public License, this software may not be
% used for commercial purposes without explicit written permission after
% contacting jonathan.moeyersons@esat.kuleuven.be
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <https://www.gnu.org/licenses/>.

% Set basic variables
a = 0.1; % 10% difference
b = 0.05; % 5% difference
max_k = 5; % Maximum ammount of previous RR-intervals

% Get the RR-intervals
RR = diff(Rpos);

% Get the amount of RR-intervals
len = length(RR);

% Pre-allocate
RRref_all = zeros(1,len); % Reference RR intervals
ectopic = zeros(1,len); % Index of ectopics

% Set first RR-reference
RR_ref = median(RR(1:max_k));
RRref_all(1) = RR_ref;

% Set loop start
counter = 1;

% Loop over the different RR-intervals
while counter < len-1
    % Define the RR-ref if you are further than the first RR-interval
    if counter > 2

        % Set variables
        k = 1; % Number of previous RR-intervals
        sum_ref = 0; % Sum of the good previous RRI's
        num = 0; % Number of good previous RRI's
        wght = [0.5 0.3 0.2]; % Weights: the farther away, the lower the weight

        while counter-k > 0 && num < 3 && k < max_k
```

```
            % Check if the previous RRI is an ectopic
            if ectopic(counter-k) == 1
                prob_check = 1;
            elseif ectopic(counter-k) == 0
                prob_check = 0;
            end

            % If this is not the case, use it for reference
            if ~prob_check
                num = num + 1;
                sum_ref = sum_ref + wght(num)*RR(counter-k);
            end
            k = k+1;
        end

        % If no good previous RR-intervals are found, this probably means that
        % this is the new normal, so just take the three previous
        if num <= 1
            k = 1;
            while counter-k > 0 && num < 3
                num = num + 1;
                sum_ref = sum_ref + wght(num)*RR(counter-k);
                k = k+1;
            end
        end

        % Define the reference variables
        RR_ref = sum_ref/sum(wght(1:num));
        no_RR_ref = nnz(RRref_all);
        RRref_all(no_RR_ref+1) = RR_ref;
    end

    % Check for ectopics
    if ((RR(counter) < (1-a)*RR_ref) && (RR(counter+1) > (1+b)*RR_ref)) ||...
            ((RR(counter) > (1+b)*RR_ref) && (RR(counter+1) < (1-a)*RR_ref))

        % Indicate the presence of an ectopic
        ectopic(counter) = 1;
    end

    % Add to the counter
    counter = counter + 1;
end

% Define the ectopic locations
ect_loc = find(ectopic == 1)+1;
```

B.1.10 ectopic-detection-correction.m

```
function [Rpos, RR, Pod] = ectopic_detection_correction(fs,RR,Rpos)
% Input
% fs    - sampling frequency
% RR    - RR signal
% Rpos - Position of the R peaks
%
% Output
% Rpos - New position of the peaks
% RR    - Corrected RR interval
```

```
% Pod  - Position of the ectopic peaks. Used for respiration analysis
%        The two beats are replaced, even if the last one is ok.
%
% Author(s):    Jonathan Moeyersons       (Jonathan.Moeyersons@esat.kuleuven.be)
%               Sabine Van Huffel         (Sabine.Vanhuffel@esat.kuleuven.be)
%               Carolina Varon            (Carolina.Varon@esat.kuleuven.be)
%
% Version History:
% - 06/05/2019   JM      Initial version
%
% Copyright (c) 2019,  Jonathan Moeyersons, KULeuven-ESAT-STADIUS
%
% This software is made available for non commercial research purposes only
% under the GNU General Public License. However, notwithstanding any
% provision of the GNU General Public License, this software may not be
% used for commercial purposes without explicit written permission after
% contacting jonathan.moeyersons@esat.kuleuven.be
%
% This program is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.  If not, see <https://www.gnu.org/licenses/>.

% Pre-allocate
Pod = [];

% Remove ectopics first
RRmed = medfilt1(RR,5);    % Fifth order median filter
for ii = 6:length(RR)-1
    if((RR(ii) < RRmed(ii)*0.9) && (RR(ii+1) > RRmed(ii)*1.05)) ...
            || ((RR(ii) > RRmed(ii)*1.05) && (RR(ii+1) < RRmed(ii)*0.9))% one way down, next one way up and viceversa

        % Adjust the investigated RR-interval
        RR(ii) = (RR(ii)+RR(ii+1))/2;
        Rpos(ii+1) = Rpos(ii)+round(RR(ii)*fs/1000);

        % Adjust the next RR-interval
        RR(ii+1) = RR(ii);
        Rpos(ii+2) = Rpos(ii+1)+round(RR(ii+1)*fs/1000);

        % Store the index of the ectopic beats
        Pod = [Pod; Rpos(ii+1); Rpos(ii+2)];   %#ok
    end
end

% Correction of ectopic beats. 20% filter is used but for cases when the
% actual interval is small and the next one is large and viceversa
RRmed = medfilt1(RR,5);    % Fifth order median filter
for ii = 6:length(RR)-1
```

```
    if((RR(ii) < RRmed(ii)*0.85) && (RR(ii+1) > RRmed(ii)*1.15)) ...
        || ((RR(ii) > RRmed(ii)*1.15) && (RR(ii+1) < RRmed(ii)*0.85))% one way down, next one way up and viceversa

        % Adjust the investigated RR-interval
        RR(ii) = (RR(ii)+RR(ii+1))/2;
        Rpos(ii+1) = Rpos(ii)+round(RR(ii)*fs/1000);

        % Adjust the next RR-interval
        RR(ii+1) = RR(ii);
        Rpos(ii+2) = Rpos(ii+1)+round(RR(ii+1)*fs/1000);

        % Store the index of the ectopic beats
        Pod = [Pod; Rpos(ii+1); Rpos(ii+2)];  %#ok
    end
end

% Clean and sort Pod
Pod = sort(unique(Pod));
```

B.1.11 env-secant.m

```
function [env] = env_secant(x_data, y_data, view, side)
% Function call: env_secant(x_data, y_data, view, side)
% Calculates the top envelope of data <y_data> over <x_data>.
% Method used: 'secant-method'
% env_secant() observates the max. slope of about <view> points,
% and joints them to the resulting envelope.
% An interpolation over original x-values is done finally.
% <side> ('top' or 'bottom') defines which side to evolve.
% Author: Andreas Martin, Volkswagen AG, Germany


side = strcmpi ( {'top','bottom'}, side ) * [ 1 ; -1 ];

assert (view > 1, ...
    'Parameter <view> too small!');
assert (ndims (x_data) == 2, ...
    'Parameter <x_data> has to be vector type!' );
assert (size (x_data, 1) == 1 || size (x_data, 2) == 1, ...
    'Parameter <x_data> has to be vector type (Nx1)!' );
assert (ndims (y_data) == 2, ...
    'Parameter <y_data> has to be vector type (Nx1)!' );
assert (size (y_data, 1) == 1 || size (y_data, 2) == 1, ...
    'Parameter <y_data> has to be vector type (Nx1)!' );
assert (length (x_data) == length (y_data), ...
    'Parameters <x_data> and <y_data> must have same length!' );
assert (side ~= 0, ...
    'Parameter <side> must be ''top'' or ''bottom''' );

data_len = length (y_data);
x_new = x_data(1);
y_new = y_data(1);

i = 2;
while i < data_len
    m_max = -Inf; % stores maximum slope in forward viewed neighbourhood
    for ii = i+1 : min(i+view, data_len)
        m = ( y_data(ii) - y_data(i) ) / (ii-i) * side;
```

```
        % Equidistant x_data assumed! Use next row instead, if not:
        % m = ( y_data(ii) - y_data(i) ) / ( x_data(ii) - x_data(i) );
        if m >= m_max
            % New max. slope found: store new "observation point"
            % always traced when ii==1
            m_max = m;
            i_op = ii;
        end
    end
    x_new = [ x_new x_data(i_op) ];
    y_new = [ y_new y_data(i_op) ];
    i = i_op;
end

env = interp1 (x_new, y_new, x_data,'linear', 'extrap');
```

B.1.12 SQI-ACF.m

```
function [SQI4,averageweights] = SQI_ACF(Fs,iECG,segm,filtro,lags)
weights= ACF_Artefact(iECG,Fs,segm,filtro,lags);
%SQI_ACF Summary of this function goes here
% This function determines if the weight of the calculated autocorrelation
% function is acceptable or not. The output of this function is a bit and
% is defined with SQI:
% If SQI = 1, signal is acceptable
% If SQI = 0, signal is not acceptable
%   Detailed explanation goes here

% The function ACF_Artefact can filter the signal before
% For ECG signals it is recommended to use a bandpass filter with cutoff
% frequencies at 1Hz and 40Hz

filtro.type = 'HL';
filtro.hf = 40;
filtro.lf = 1;

fs = Fs;
ecg = iECG;

% segm = 1; % length of the segments to be analyzed (in seconds)
% lags = []; % Lags used in the ACF (default 250ms)
%manual = 1; % 1 in case you want to apply a threshold in the weights

averageweights = mean(weights);
if averageweights > 0.91
    SQI4 = 1;
else
    SQI4 = 0;
end

end
```

B.1.13 ACF-Artefact.m

```
% Artefact detection based on the Autocorrelation Function of the ECG
% signal
function weights =ACF_Artefact(iECG,Fs,segm,filtro,lags)
fs = Fs;
ecg = iECG;
```

```
% INPUTS:
%       ecg  - Signal to be segmented
%       fs   - Sampling frequency (Hz)
%       segm - Duration of the segments, in seconds
%
%       filtro.type - Filter type 'low' (low pass), 'high' (high pass),
%                     'stop50', 'stop60''All'.
%                     'LS' low and stop
%                     'HS' high and stop
%                     'HL' high and low
%       filtro.hf   - cutoff frequency for lowpass filter
%       filtro.lf   - cutoff frequency for highpass filter
%
%       lags  - lags of the autocorrelation. Default 250ms
%       manual- (1)Use the manual selection tool
%
% OUTPUTS:
%       good - Indices of the good segments
%       bad  - Indices of the bad segments
%       Cutecgs - Segments of ECG
%       weights - Values of similarity for each segment
%
%
% This function first splits the signal 'ecg' sampled at 'fs'(Hz), into
% segments of length specified in 'segm'. Then it filters the signal using
% the parameters indicated in the structure filtro. After this, it computes
% the autocorrelation function of each segment (each column) using the
% indicated 'lags'. Then it computes the cosine similarity and the degree
% vector. From this degree vector a threshold can be determined by the
% precentile or by the interquartile range.
%
% You will be asked different questions:
% >> Method: 1 for Percentile, 0 for Interquartile range (3 to stop)=
%
%   * Press 1 if you want to use the percentile as threshold.
%     >> Percentage? (5 default) =
%     - Press enter if you want to keep the default value (5%)
%     - Indicate the percentage you wish to use in case it is different
%       than 5%
%     >> Decision: 1 to change =
%     - Press enter if you are satisfied with the threshold
%     - Press 1 if you want to change the threshold
%
%   * Press 0 if you want to use the interuantile range for the definition
%     of the threshold. This is how many times you subtract the IQR from
%     the third quartile (75%)
%     >> IQR? (1.25 default) =
%     - Press enter if you want to keep the default value (Q3 - 1.25IQR)
%     - Indicate the factor in case it is different than 1.25
%     >> Decision: 1 to change =
%     - Press enter if you are satisfied with the threshold
%     - Press 1 if you want to change the threshold
%
%   * Press 3 to stop

% Varon C., Testelmans D., Buyse B., Suykens J.A.K., Van Huffel S.   Robust
% artefact detection in long-term ECG recordings based on autocorrelation
% function similarity and percentile analysis. In proceedings of the 34th
```

```
% Annual International Conference of the IEEE Engineering in Medicine and
% Biology Society (EMBC), San Diego CA, USA, Aug. 2012.
%
% Owner of code : KU Leuven
% Developer of code: Carolina Varon  (STADIUS-ESAT-KU Leuven)
% Contact persons: Carolina Varon (carolina.varon@esat.kuleuven.be),
% Sabine Van Huffel (sabine.vanhuffel@esat.kuleuven.be)

if isempty(ecg)
    error('Please load an ECG signal as double','Bad Input','modal')
end
if isempty(fs) && isempty(segm)
    error('You must enter a numeric value for fs and segment','Bad Input','modal')
end
if ~isscalar(fs) && ~isinteger(int8(segm)) && segm<0
    error('You must enter a numeric valid value for fs and/or segment.','Bad Input','modal')
end


m = mean(ecg); s = std(ecg); % mean y std para normalizar
if size(ecg,1)>1
    ecg = ecg';
end

sze = length(ecg); segm = segm*fs;
nl=floor(sze/segm); % how many segments?
if nl==0
    error('The length of the segment is too long','Bad Input','modal')
end
% segments in the columns
Cutecgs = double(reshape(double(ecg(1:(nl*segm))),segm,nl));
Cutecgs = Cutecgs-(repmat(m,size(Cutecgs,1),size(Cutecgs,2)));
Cutecgs = Cutecgs./(repmat(s,size(Cutecgs,1),size(Cutecgs,2)));

% ******************************************************************
% Filter the ECG
if isempty(filtro.type)
    error('Please spefify the type of filter: low, high, stop60, stop50, LS, HS, HL')
end
if exist('lags','var')==0 || isempty(lags)
    lags = 0.250;
end
lg = round(lags*fs);
type = filtro.type;
hf = filtro.hf;
lf = filtro.lf;

C = zeros(size(Cutecgs,1),size(Cutecgs,2));
l = size(Cutecgs,2);
parfor i=1:l
    Fecg    = filterECG(Cutecgs(:,i),fs,type,hf,lf);
    C(:,i) = Fecg;
    a(:,i) = acrr(C(:,i),lg); % Autocorrelation
end

```

```
% ***************************************************************
% Cosine similaritt and computation of the degrees (weights)
Au = a;
idx = find(isnan(sum(Au))); lg2 = size(Au,1);
for i=1:length(idx)
    Au(:,idx(i)) = zeros(lg2,1);
end
X2  = sum(Au(2:end,:).*Au(2:end,:),1); XY=Au(2:end,:)'*Au(2:end,:);
K = XY./sqrt(X2'*X2);
for i=1:length(idx)
    K(:,idx(i)) = zeros(nl,1); K(idx(i),:) = zeros(1,nl);
end
sg  = sum(K)./max(sum(K));

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

i2 = 5;
thr = prctile(sg,i2);
i1 = 1;

% figure,
% subplot(121),plot(Au),xlabel('Time lags'),ylabel('Autocorrelation')
% subplot(122)
% plot(sg,'.-'),hold on,xlabel('Segment'),ylabel('Similarity')
%             plot([0 length(sg)],[thr thr],'--r')
%             legend('Similarity','Criterion')
%             hold off

if i1 == 1 || i1 == 0
    weights = sg;
    Id = [1:length(sg)]; bad = find(sg<thr);
    good = setdiff(Id,bad);
end

% disp(['Number of segments :      ',num2str(size(Cutecgs,2))])
% disp(['Number of segments marked as outlier: ',num2str(length(bad))])
% disp(['Number of segments marked as good:    ',num2str(length(good))])
```

B.1.14 acrr.m

```
function varargout = acrr(Series , nLags , Q , nSTDs)
%AUTOCORR Compute or plot sample auto-correlation function.
%   Compute or plot the sample auto-correlation function (ACF) of a univariate,
%   stochastic time series. When called with no output arguments, AUTOCORR
%   displays the ACF sequence with confidence bounds.
%
%   [ACF, Lags, Bounds] = autocorr(Series)
%   [ACF, Lags, Bounds] = autocorr(Series , nLags , M , nSTDs)
%
%   Optional Inputs: nLags , M , nSTDs
%
% Inputs:
%   Series - Vector of observations of a univariate time series for which the
%     sample ACF is computed or plotted. The last row of Series contains the
%     most recent observation of the stochastic sequence.
%
% Optional Inputs:
%   nLags - Positive, scalar integer indicating the number of lags of the ACF
%     to compute. If empty or missing, the default is to compute the ACF at
```

```
%      lags 0,1,2, ... T = minimum[20 , length(Series)-1]. Since an ACF is
%      symmetric about zero lag, negative lags are ignored.
%
%   M - Non-negative integer scalar indicating the number of lags beyond which
%      the theoretical ACF is deemed to have died out. Under the hypothesis that
%      the underlying Series is really an MA(M) process, the large-lag standard
%      error is computed (via Bartlett's approximation) for lags > M as an
%      indication of whether the ACF is effectively zero beyond lag M. On the
%      assumption that the ACF is zero beyond lag M, Bartlett's approximation
%      is used to compute the standard deviation of the ACF for lags > M. If M
%      is empty or missing, the default is M = 0, in which case Series is
%      assumed to be Gaussian white noise. If Series is a Gaussian white noise
%      process of length N, the standard error will be approximately 1/sqrt(N).
%     M must be less than nLags.
%
%   nSTDs - Positive scalar indicating the number of standard deviations of the
%      sample ACF estimation error to compute assuming the theoretical ACF of
%      Series is zero beyond lag M. When M = 0 and Series is a Gaussian white
%      noise process of length N, specifying nSTDs will result in confidence
%      bounds at +/-(nSTDs/sqrt(N)). If empty or missing, default is nSTDs = 2
%      (i.e., approximate 95% confidence interval).
%
% Outputs:
%   ACF - Sample auto-correlation function of Series. ACF is a vector of
%      length nLags + 1 corresponding to lags 0,1,2,...,nLags. The first
%      element of ACF is unity (i.e., ACF(1) = 1 = lag 0 correlation).
%
%   Lags - Vector of lags corresponding to ACF (0,1,2,...,nLags).
%
%   Bounds - Two element vector indicating the approximate upper and lower
%      confidence bounds assuming that Series is an MA(M) process. Note that
%      Bounds is approximate for lags > M only.
%
% Example:
%   Create an MA(2) process from a sequence of 1000 Gaussian deviates, then
%   visually assess whether the ACF is effectively zero for lags > 2:
%
%      x = randn(1000,1);              % 1000 Gaussian deviates ~ N(0,1).
%      y = filter([1 -1 1] , 1 , x);   % Create an MA(2) process.
%      autocorr(y , [] , 2)            % Inspect the ACF with 95% confidence.
%
% See also CROSSCORR, PARCORR, FILTER.

%   Copyright 1999-2003 The MathWorks, Inc.
%   $Revision: 1.1.8.2 $  $Date: 2008/06/16 16:37:57 $

%
% Reference:
%   Box, G.E.P., Jenkins, G.M., Reinsel, G.C., "Time Series Analysis:
%      Forecasting and Control", 3rd edition, Prentice Hall, 1994.

%
% Ensure the sample data is a VECTOR.
%

[rows , columns]  =  size(Series);

if (rows ~= 1) && (columns ~= 1)
```

```
    error('econ:autocorr:NonVectorInput' , ' Input ''Series'' must be a vector.'
        );
end

rowSeries   =  size(Series,1) == 1;

Series      =  Series(:);        % Ensure a column vector
n           =  length(Series);   % Sample size.
defaultLags =  20;               % BJR recommend about 20 lags for ACFs.

%
% Ensure the number of lags, nLags, is a positive
% integer scalar and set default if necessary.
%

if (nargin >= 2) && ~isempty(nLags)
   if numel(nLags) > 1
      error('econ:autocorr:NonScalarLags' , ' Number of lags ''nLags'' must be a
          scalar.');
   end
   if (round(nLags) ~= nLags) || (nLags <= 0)
      error('econ:autocorr:NonPositiveInteger' , ' Number of lags ''nLags'' must
          be a positive integer.');
   end
   if nLags > (n - 1)
      error('econ:autocorr:LagsTooLarge' , ' Number of lags ''nLags'' must not
          exceed ''Series'' length - 1.');
   end
else
   nLags  =  min(defaultLags , n - 1);
end

%
% Ensure the hypothesized number of lags, Q, is a non-negative integer
% scalar, and set default if necessary.
%
if (nargin >= 3) && ~isempty(Q)
   if numel(Q) > 1
      error('econ:autocorr:NonScalarQ' , ' Number of lags ''Q'' must be a scalar
          .');
   end
   if (round(Q) ~= Q) || (Q < 0)
      error('econ:autocorr:NegativeInteger' , ' Number of lags ''Q'' must be a
          non-negative integer.');
   end
   if Q >= nLags
      error('econ:autocorr:QTooLarge' , ' ''Q'' must be less than ''nLags''.');
   end
else
   Q  =  0;     % Default is 0 (Gaussian white noise hypothisis).
end

%
% Ensure the number of standard deviations, nSTDs, is a positive
% scalar and set default if necessary.
%

if (nargin >= 4) && ~isempty(nSTDs)
```

```
    if numel(nSTDs) > 1
       error('econ:autocorr:NonScalarSTDs' , ' Number of standard deviations ''
          nSTDs'' must be a scalar.');
    end
    if nSTDs < 0
       error('econ:autocorr:NegativeSTDs' , ' Number of standard deviations ''
          nSTDs'' must be non-negative.');
    end
 else
    nSTDs =  2;     % Default is 2 standard errors (95% condfidence interval).
end

%
% Convolution, polynomial multiplication, and FIR digital filtering are
% all the same operation. The FILTER command could be used to compute
% the ACF (by computing the correlation by convolving the de-meaned
% Series with a flipped version of itself), but FFT-based computation
% is significantly faster for large data sets.
%
% The ACF computation is based on Box, Jenkins, Reinsel, pages 30-34, 188.
%

nFFT =  2^(nextpow2(length(Series)) + 1);
F    =  fft(Series-mean(Series) , nFFT);
F    =  F .* conj(F);
ACF  =  ifft(F);
ACF  =  ACF(1:(nLags));% + 1));         % Retain non-negative lags.
ACF  =  ACF ./ ACF(1);     % Normalize.
ACF  =  real(ACF);

%
% Compute approximate confidence bounds using the Box-Jenkins-Reinsel
% approach, equations 2.1.13 and 6.2.2, on pages 33 and 188, respectively.
%

sigmaQ  =  sqrt((1 + 2*(ACF(2:Q+1)'*ACF(2:Q+1)))/n);
bounds  =  sigmaQ * [nSTDs ; -nSTDs];
Lags    =  [0:nLags]';

 if nargout == 0                        % Make plot if requested.

%
%  Plot the sample ACF.
%
   lineHandles  =  stem(Lags , ACF , 'filled' , 'r-o');
   set   (lineHandles(1) , 'MarkerSize' , 4)
   grid  ('on')
   xlabel('Lag')
   ylabel('Sample Autocorrelation')
   title ('Sample Autocorrelation Function (ACF)')
   hold  ('on')
%
%  Plot the confidence bounds under the hypothesis that the underlying
%  Series is really an MA(Q) process. Bartlett's approximation gives
%  an indication of whether the ACF is effectively zero beyond lag Q.
%  For this reason, the confidence bounds (horizontal lines) appear
%  over the ACF ONLY for lags GREATER than Q (i.e., Q+1, Q+2, ... nLags).
%  In other words, the confidence bounds enclose ONLY those lags for
```

```
% which the null hypothesis is assumed to hold.
%

    plot([Q+0.5 Q+0.5 ; nLags nLags] , [bounds([1 1]) bounds([2 2])] , '-b');

    plot([0 nLags] , [0 0] , '-k');
    hold('off')
    a = axis;
    axis([a(1:3) 1]);

else

%
% Re-format outputs for compatibility with the SERIES input. When SERIES is
% input as a row vector, then pass the outputs as a row vectors; when SERIES
% is a column vector, then pass the outputs as a column vectors.
%
    if rowSeries
       ACF    = ACF.';
       Lags   = Lags.';
       bounds = bounds.';
    end

    varargout = {ACF , Lags , bounds};

end
```

B.1.15 filterECG.m

```

function [Fecg] = filterECG(ecg,fs,type,hf,lf);

% INPUT:   ecg  - Signal
%          fs   - Sampling frequency
%          type - Filter type 'low', 'high', 'stop50', or 'stop60'.
%                 'LS50' low and stop50, 'LS60' low and stop60
%                 'HS50' high and stop50, 'HS60' low and stop60
%                 'HL' high and low
%          hf   - cutoff frequency for lowpass filter
%          lf   - cutoff frequency for highpass filter
%
% OUTPUT:  Fecg - Filtered ECG
%
%

% Copyright (c) 2015, Carolina Varon, KULeuven-ESAT-SCD
% This software is made available for non commercial research purposes only
% under the GNU General Public License. However, notwithstanding any provision
% of the GNU General Public License, this software may not be used for commercial
% purposes without explicit written permission after contacting
% carolina.varon@esat.kuleuven.be
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Nf = fs/2;  % Nyquist frequency


if strcmp(type,'low') || strcmp(type,'All') || strcmp(type,'LS') || strcmp(type,'HL')
    Cf = hf/Nf;         % Cutoff freq. Passband corner freq. 100Hz
```

```

    [bl,al] = butter(4,Cf,'low');     % Low pass filter
    yl       = filtfilt(bl,al,ecg);
    ecg      = yl;
end

if strcmp(type,'high') || strcmp(type,'All') || strcmp(type,'HS') || strcmp(type,'HL')
    Cf = lf/Nf;          % Cutoff freq. Passband corner freq. 0.5Hz

    [bh,ah] = butter(2,Cf,'high');    % High pass filter
    yh       = filtfilt(bh,ah,ecg);
    ecg      = yh;
end

if strcmp(type,'stop50') || strcmp(type,'All') || strcmp(type,'HS50') || strcmp(type,'LS50')
    Cf = [45 55]/Nf;     % Cutoff freq. Passband corner freq. 0.5Hz

    [bs,as] = butter(6,Cf,'stop');    % stopband filter
    ys       = filtfilt(bs,as,ecg);
    ecg      = ys;
end

if strcmp(type,'stop60') || strcmp(type,'All') || strcmp(type,'HS60') || strcmp(type,'LS60')
    Cf = [55 65]/Nf;     % Cutoff freq. Passband corner freq. 0.5Hz

    [bs,as] = butter(6,Cf,'stop');    % stopband filter
    ys       = filtfilt(bs,as,ecg);
    ecg      = ys;
end

Fecg = ecg;
```

B.1.16 ECGFILTER.m

```
%%
% Output:
% filteredECG: The filtered ECG signal
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% The algoritm filters the incoming ECG signal. The powerline Noise, the
% Baseline Wander and frequency components higher than 150 Hz are removed
% removed from the incoming ECG.
%%

function filteredECG = ECGFILTER(Fs,iECG,Fnotch,Fnotch2,Fnotch3,Order1,fcut,Order2,fcut2)
%% Powerline Noise removal

     BW = 1; %Bandwidth
     [b, a] = iirnotch (Fnotch/ (Fs/2), BW/ (Fs/2));
     y1=filtfilt(b,a,iECG);

     [b2, a2] = iirnotch (Fnotch2/ (Fs/2), BW/ (Fs/2));
     y2=filtfilt(b2,a2,y1);
```

```
    [b3, a3] = iirnotch (Fnotch3/ (Fs/2), BW/ (Fs/2));
    y3=filtfilt(b3,a3,y2);



%% Baseline Removal
    [b4,a4] = butter(Order1,fcut/(Fs/2),'high');
    y4 = filtfilt(b4,a4,y3);

%% Removal of frequencies higher than 150Hz
    [b5,a5] = butter(Order2,fcut2/(Fs/2),'low');
    y5 = filtfilt(b5,a5,y4);
    %% UNKNOWN YET
    BW2 = 4;
    [b6, a6] = iirnotch (120/ (Fs/2), BW2/ (Fs/2));
    filteredECG=filtfilt(b6,a6,y5);

end
```

B.2 respiratory system design code

B.2.1 mainrespiratory.m

```
%% Main respiratory
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% This algoritm supplies the Respiratory.m with its inputs.
%
%
%%

clear all;
load('Data_Respiration.mat')

A = Resp_SA;
patientdata = A{1,7};
t = 80;  %Time length in sec
Nsegment =t*fs_resp;
p = 1;
q = 125;  % p and q are the ratio of downsampling used on the function resample
Order1 = 6;
fcut = 0.5;
Order2 = 4;
fcut2 = 0.1;

lengthsignal = length(patientdata);
number_segments = floor(lengthsignal/Nsegment);
processed_signal = cell(1,number_segments);

for i = 0:(number_segments - 1)

respirationsegment = patientdata(1+(i*Nsegment):(Nsegment*(1+i)));
respirationsegmentdetrend = detrend(respirationsegment);

[Resp,RR,SQI] = Respiratory(t,fs_resp,p,q,Order1,fcut,Order2,fcut2,
    respirationsegmentdetrend,respirationsegment);
```

```
    processed_signal{1,(i+1)} = Resp;
    processed_signal{2,(i+1)} = RR;
    processed_signal{3,(i+1)} = SQI;
end
```

B.2.2 Respiratory.m

```
%%
%
%  Output:
%  Resp: The filtered respiratory signal
%  RR = Respiratory rate obtained from the respiratory signal
%  SQI = Signal quality indicator assigned to the filtered respiratory.
%
%  Author(s): Enes Kinaci
%             Talha Kuruoglu
%
% This algoritm performs all the computations made in the respiratory
% signal system. The following scrips are used to make the computations in
% this algorithm: NaNorNotResp.m, Powerresp.m, Filterresp.m and
% RespiratoryRate.m. The NaNorNotResp function is used to detect possible NaNs in the respiratory signal
% before making computations. Filterresp function is used to filter the
% respiratory signal.The Powerresp function performs the power
% quality check on the downsampled and filtered respiratory signal.
% Afterwards the RR is calculated with the RespiratoryRate function.
% Afterwards the second quality check is performed where it is checked
% wheter the breath taken in the respiratory is taken within 10 seconds or
% longer than 10 seconds.
%

%%

function [Resp,RR,SQI] = Respiratory(t,fs_resp,p,q,Order1,fcut,Order2,fcut2,respirationsegmentdetrend,respirationsegment)

NaN = NaNorNotResp(respirationsegmentdetrend); % NaN check

if (NaN == 0)
%% Filtering and downsampling

[filteredrespiration] = Filterresp(respirationsegmentdetrend,Order1,fcut,Order2,fcut2,fs_resp);
respdownfiltered = resample(filteredrespiration,p,q);
respdownnormal = resample(respirationsegmentdetrend,p,q);
fs_resp_down = fs_resp * (p/q);

%% Power Quality Indicator Check
[Ipower] = Powerresp(fs_resp_down,respdownfiltered);
if (Ipower == 1)

    iy2 = (-respdownfiltered);
    [invminresp, posinitieelmin] = findpeaks(iy2);
    [maxresp_initieel, posinitieel] = findpeaks(respdownfiltered); % Detection of 'peaks' in the respiratory signal

    [RR,positionmax] = RespiratoryRate(respdownfiltered,fs_resp_down,t,
```

```
        maxresp_initieel,invminresp); % Calculation of respiratory rate
    distanceMaxPeak = diff(positionmax);
    %% Second Quality indicator
    if distanceMaxPeak > 10
        SQI = 0;
        Resp = filteredrespiration;
    else
        SQI = 1;
        Resp = filteredrespiration;
    end


else %If power quality is bad
    iy2 = (-respdownfiltered);
    [invminresp, posinitieelmin] = findpeaks(iy2);
    [maxresp_initieel, posinitieel] = findpeaks(respdownfiltered);

    Resp = filteredrespiration;
    [RR,positionmax] = RespiratoryRate(filteredrespiration,fs_resp_down,t,
        maxresp_initieel,invminresp);
    SQI = 0;
end
else %When there is a NaN
  Resp = respirationsegment;
  RR = 0;
  SQI = 0;
end
end
```

B.2.3 NaNorNotResp.m

```
%%
% Output:
% SQI_NaN - signal quality indicator which is an indication of the possible
    presence of a NaN/NaNs.
%
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
%This algoritm checks whether the incoming Respiratory signal signal has a NaN
    or NaNs inside of it.
%Depending on the presence of a NaN or NaNs, a SQI is sent as output.
%SQI = 1, means that a NaN is detected, while SQI = 0, means that no NaNs are
    detected.
%%
function SQI_NaN = NaNorNotResp(respirationsegmentdetrend)
i =1;
SQI_NaN =0;
    Index = isnan(respirationsegmentdetrend(i:1));
    if Index == 1
        SQI_NaN = 1;
    else
        SQI_NaN =0;
    end

end
```

B.2.4 Filterresp.m

```
%%
%
% Output:
% filteredrespiration: The filtered respiratory signal
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% This algoritm implements the bandpass filter of 0.1 Hz - 0.5 Hz on the
% respiratory signal.
%
%%

function [filteredrespiration] = Filterresp(respirationsegmentdetrend,Order1, fcut,Order2,fcut2,fs_resp)
%% Bandpass filter between 0.1 Hz and 0.5 Hz

[b,a] = butter(Order1,fcut/(fs_resp/2),'low');
y1 =filtfilt(b,a,respirationsegmentdetrend);

[b2,a2] = butter(Order2,fcut2/(fs_resp/2),'high');
filteredrespiration = filtfilt(b2,a2,y1);

end
```

B.2.5 Powerresp.m

```

%%
% Output:
% Ipower = Signal quality indicator after the power quality check
%
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% This algorithm is the contains the power quality check in which the percentage of the power in the band of interest of the
% respiratory signal is compared with its whole spectrum. Depending on this percentage, a Ipower = 0 or I_power = 1 is send as output.

%%
function [Ipower] = Powerresp(fs_resp_down,respdown)

%% input
windowlength = length(respdown);   %Length of the window
overlap = 0.7*length(respdown);          %number of overlap samples
ndft     = length(respdown);          %number of DFT points
%% Plotting the PSD using Welch Method

[pxx,f] = pwelch(respdown,windowlength,overlap,ndft,fs_resp_down);

%% Calculating the total power

P0 = trapz(pxx); %Whole spectrum power
pxxresp = pxx(6:30);
Presp = trapz(pxxresp);
frac = ((Presp)/P0)*100;

```

```
if (frac > 80)
    Ipower = 1;
else
    Ipower = 0;
end

end
```

B.2.6 RespiratoryRate.m

```
%% Respiratory Rate Calculation.
%
% Output:
% RR = Respiratory Rate of respiratory signal
% positionmax = Position of the maximum values respiratory signal.
%
% Author(s): Enes Kinaci
%            Talha Kuruoglu
%
% The Respiratory Rate is calculated by looking at the amount of breaths
% taken in the respiratory signal, 'breaths' that are taken faster than 2
    seconds are ignored in the
% respiratory rate calculation.
%
%% Determining of RR rate with Respiratory signal estimate

function [RR,positionmax] = RespiratoryRate(respdownfiltered,fs_resp_down,t,
    maxresp_initieel,invminresp)

% Minimum distance is determined:
[maxresp, positionmax] = findpeaks(respdownfiltered, fs_resp_down, '
    MinPeakDistance', 2); %Check if breath is taken faster than 2 seconds

%making sure that inhale and exhale parts in respiratory signal is in
%pairs, to truely detect breathing
N = length(maxresp_initieel) - length(maxresp);
lengthMinimum = length(invminresp) - N;

%Calculation RR
if (lengthMinimum >= length(maxresp))
    bpm = length(maxresp);
    RR = bpm/t;
else
    bpm = lengthMinimum;
    RR = bpm/t;
end
end
```