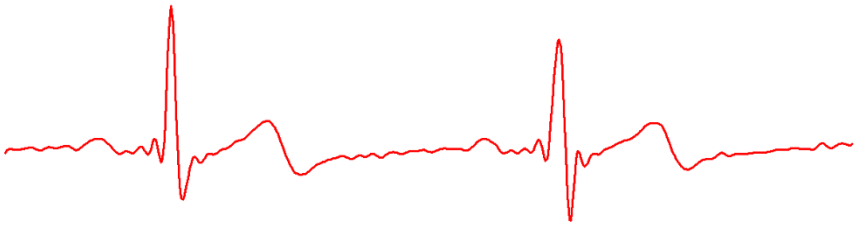


ECG Compression and QRS Detection: an IoT Approach

Hugo Miguel Oliveira de Sousa
Mestrado em Engenharia Matemática
Departamento de Matemática
2019

Orientador
Ana Paula Rocha, Professora Auxiliar, Faculdade de Ciências da Universidade do Porto

Orientador de Estágio
Jonathan Tooley, Sócio, JTA: The Data Scientists

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____

Universidade do Porto

Masters Thesis

ECG Compression and QRS Detection: an IoT Approach

Author:
Hugo Sousa

Supervisor:
Ana Paula Rocha

Internship Supervisor:
Jonathan Tooley

A thesis submitted in fulfilment of the requirements
for the degree of MSc. Mathematical Engineering

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

December 2, 2019

Acknowledgements

This work is more than the five months that I spent in JTA: The Data Scientist, it is the result of the five years that I spent at the University of Porto. For that, I would like to thank all my colleagues but specially my friends for all the help that they gave me. Also acknowledge all the professors particularly Professor Ana Paula Rocha who guided me in this project.

My thanks are extended to the team from JTA for all the assistance during my internship. To João Freitas, who worked directly with me in this project and Jonathan Tooley my supervisor, and leader of this project. In addition, I would like to thank the remaining partners of JTA, Jonathan Rignall and Duarte Gomes for their support and guidance. For quite some time I have known what exponential growth is, thanks to JTA now I know how it feels like.

Finally, I wish to thank my parents for their support and encouragement throughout my studies.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

MSc. Mathematical Engineering

ECG Compression and QRS Detection: an IoT Approach

by Hugo Sousa

During my five month internship at JTA: The Data Scientists we were challenged to build a compression scheme for electrocardiogram (ECG) signals to be implemented in a portable ECG reader. We created a lossless compression algorithm and implemented it on a prototype device that we built. The algorithm presented a compression ratio of 5.65 in the longest recording that was made (about 39 minutes long). We also ran the compression scheme on the MIT-BIH Arrythmia database and obtained an average compression ratio of 5.61. Besides that, we developed a real-time QRS complex detection algorithm. This algorithm was tested on the MIT-BIH Arrhythmia database succeeded in the detection of 98.83 percent of the complexes.

Key-Words: Electrocardiogram, Signal Processing, QRS Complex Detection, Compression

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto
Departamento de Matemática

Mestrado em Matemática Aplicada

Compressão de ECG e Deteção de QRS: uma Abordagem IdC

por Hugo Sousa

Durante o meu estágio de cinco meses na JTA: The Data Scientists, fomos desafiados a construir um esquema de compressão para sinais de eletrocardiograma (ECG) a ser implementado num leitor de ECG portátil. Criamos um algoritmo de compressão sem perdas e implementamo-lo no protótipo que construímos. O algoritmo apresentou uma taxa de compressão de 5.65 na gravação mais longa efectuada com o protótipo (cerca de 39 minutos). Executamos ainda o esquema de compressão na base de dados "MIT-BIH Arrythmia"obtendo uma taxa de compressão média de 5.61. Para além disso, desenvolvemos um algorítmo de detecção do complexo QRS em tempo real. Este algoritmo foi testado na base de dados "MIT-BIH Arrythmia", detetando corretamente 98.83 por cento dos complexos.

Palavras-Chave: Electrocardiograma, Processamento de Sinal, Deteção de Complexo QRS, Compressão

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An electrocardiogram (ECG) is a clinical tool used to monitor the electrical activity of the heart. The electrical potentials generated by the heart are measured through electrodes placed on the surface of the skin. Those measurements result in a graph of voltage versus time, that graph is called the ECG. The first practical ECG was made by the Dutch physician and physiologist Willem Einthoven in 1895. Later, in 1924, he received the Nobel Prize in Physiology or Medicine for it.

Due to its non invasive nature, immediate availability and high versatility, the ECG is one of the most common medical studies in the assessment of cardiovascular disease.

The Holter monitor is a portable device that measures and records ECG continuously for a period of 24 hours or longer. Usually, 10 electrodes are placed at certain locations on the chest, arms, and legs. From the 10 electrodes we get a total of 12 ECG curves that look at the electrical activity of the heart from different angles.

In the era of the Internet of things it is natural that entrepreneurs and companies see an opportunity in updating devices like the Holter. At the time I started my internship at JTA: The Data Scientists we were approached by the Marketing Strategist of a company that operates in the health market . They were developing a wireless ECG monitor that would record a 1-lead ECG for an entire week. At the time, the challenge presented to us was to build a compression algorithm to implement on the device, so that it could save a week of ECG in the smallest possible space. We saw that as an opportunity to enter into the Health industry. After we built a prototype of the wireless device with the compression algorithm embedded, we started studying the possibility of also implementing real time QRS complex detection. By implementing such an algorithm the device would be

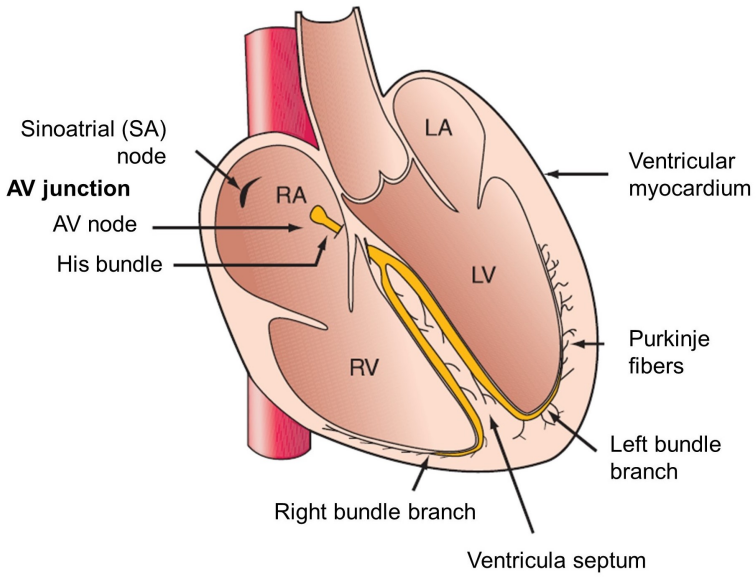
able to identify heart problems such as tachycardia, bradycardia or arrhythmia and send immediate feedback to a smartphone.

In Chapter 2 is explained what an ECG records with main focus on the lead that our device is meant to record. In Chapter 3 is presented the prototype that was built to make our ECG recordings and to implement the compression algorithm. This prototype was built according to the indications given to us by our client. In Chapter 4 is presented the data used to test the performance of the algorithms. The ECG compression scheme and the QRS detection algorithm are presented in chapters 5 and 6 respectively. In those, we also present portions of the code that we used as well as the results that the algorithms had on the datasets presented in Chapter 4. In the last chapter are presented thoughts about what could be done in the future.

Chapter 2

The Electrocardiogram

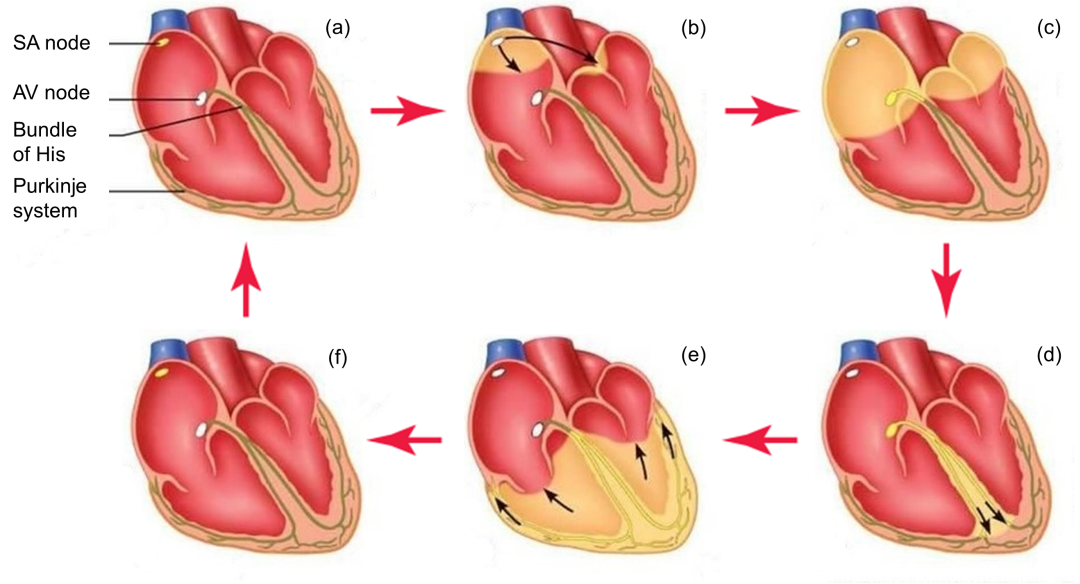
The heart is composed of different cells with distinct and very specialized functions, but all are electrically active [1]. The signal of a normal heart beat originates in a group of cells high in the right atrium. That group of cells is called the sinoatrial (SA) node (see figure 2.1), they depolarize spontaneously and the signal is then spread to the rest of the heart in two manners: by the cells that generate the force of contraction, but also by cells that form a specialized conducting pathway. The specialized conduction system is called cardiac conduction system and its composed by three main parts: the SA node, the atrioventricular (AV) node; and the His-Purkinje fiber system.



Source: Kasper, Dennis L., et al. "Harrison's principles of internal medicine." (2015).

Figure 2.1: Conduction pathways through the heart

The SA node works as the hearts natural pacemaker firing off action potentials at a regular (figure 2.2 (a)), intrinsic rate that is usually between 60 and 100 times per minute for an individual at rest [2]. First, that wave is spread through the right and left atria (figure 2.2 (b)) , followed by atrial contraction making all the blood to move from the atrius to the ventricles. Then, the impulse stimulates the tissues in the AV node and His-bundle areas (figure 2.2 (c)); together these two regions constitute the AV junction. The bundle of His bifurcates into two main branches, the right and left bundle (figure 2.2 (d)), which rapidly transmit the wave to the ventricular myocardium by the Purkinje fibers. The wave then spreads through the ventricular wall triggering the ventricular contraction which pumps the blood to the body (figure 2.2 (e)). After that, occurs the ventricle repolarization and the cycle restarts (figure 2.2 (f)).



Adapted from:
https://i.pinimg.com/originals/4a/b4/2f/4ab42f99692b22795deb14ff73ca4693.jpg

Figure 2.2: Sequence of the action potencial in the cardiac conduction system

Since the cardiac depolarization and repolarization waves have direction and magnitude, they can be represented by vectors. The electrocardiogram is the result of recording the amplitude of those vectors. Each lead looks at those vectors from diferent directions. Fundamentally, an ECG is the temporal recording of the sum of the electrical potentials on the surface of the body. To sense that, potential electrodes are placed to the extremities and chest wall.

To obtain a standard 12-lead ECG, one places two electrodes on the upper extremities, two on the lower extremities, and six on standard locations of the chest. In various combinations, the electrodes on the extremities generate the six limb leads (three standard and three augmented), and the chest electrodes produce six precordial leads. In a lead, an electrode is treated as the positive side of the voltmeter and one or two electrodes as the negative side. Therefore, a lead records the fluctuation in voltage difference between positive and negative electrodes. By variation of which electrodes are positive and which are negative, a standard 12-lead ECG is recorded.

Each lead looks at the heart from a unique angle and plane. The limb leads record potentials transmitted onto the frontal plane, and the chest leads record the potentials transmitted onto the horizontal plane. The six chest leads (also called precordial leads) are unipolar recordings obtained by six electrodes, the leads are named from V1 to V6 and are placed as follows (see figure 2.3):

V1 - Fourth intercostal space on the right sternum

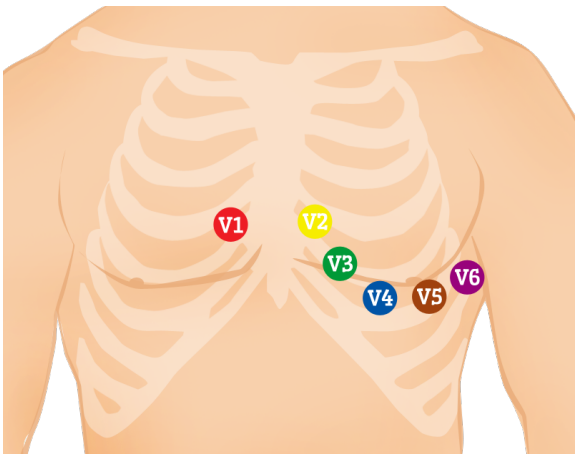
V2 - Fourth intercostal space at the left sternum

V3 - Midway between placement of V2 and V4

V4 - Fifth intercostal space at the midclavicular line

V5 - Anterior axillary line on the same horizontal level as V4

V6 - Mid-axillary line on the same horizontal level as V4 and V5



Source: https://cdn.shopify.com/s/files/1/0059/3992/files/rib.png?v=1476197594

Figure 2.3: Placement of the precordial electrodes

From the four limb electrodes the right leg (RL) is used for electrical grounding and the other three - right arm (RA), left arm (LA) and left leg (LL) - form the Einthoven's triangle (see figure 2.4). The Einthoven's triangle is used to explain how six leads are extracted from four electrodes. The RA and LA electrodes usually are placed in the shoulders joints because is electrically equivalent to attach it to the arm. By convention, the left leg represents the groin.

Three of the six leads are bipolar leads, that is, they only use the difference in voltage between two electrodes.

I - LA is the positive connection and RA is negative. This lead defines an axis in the frontal plane at 0 degrees.

II - LL is the positive connection and RA is negative. This lead defines an axis in he frontal plane at 60 degrees.

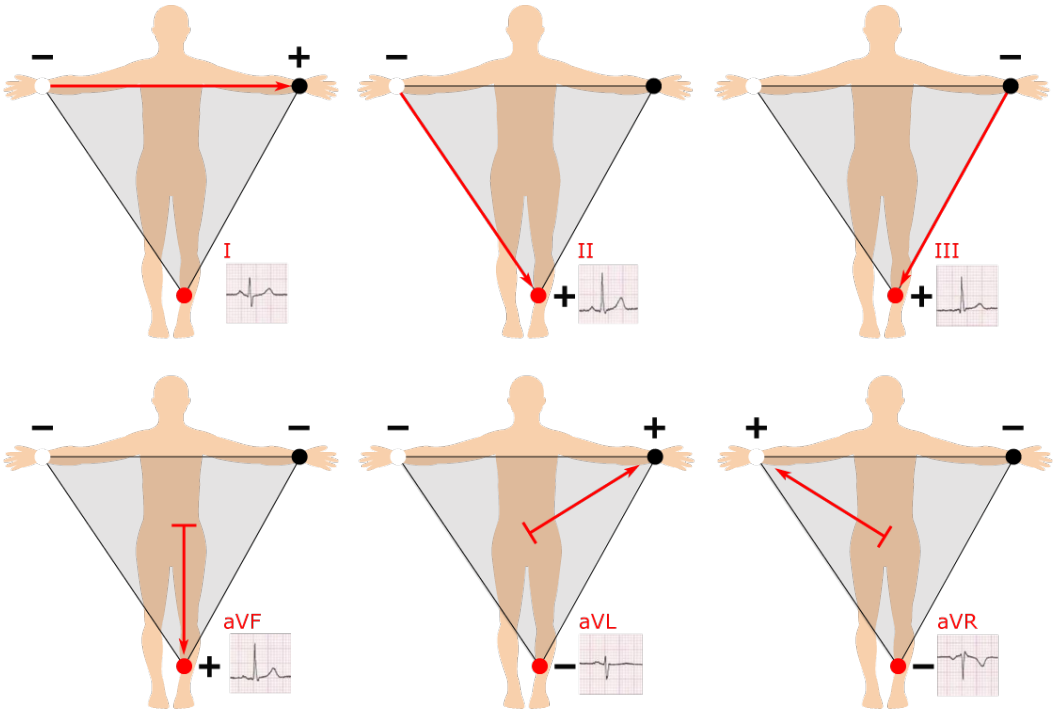
III - LL is the positive connection and LA is negative. This lead defines an axis in he frontal plane at 120 degrees.

The other three leads are the augmented unipolar leads: aVR, aVL and aVF. The a stands for augmented, the V represents the unipolar and the R, L and F represent right, left and foot respectively. They are termed unipolar leads because there is a single positive electrode that is referenced against a combination of two others limb electrodes.

aVR (augmented unipolar right) RA is the positive connection and the negative is the combination of LA and LL. This lead defines an axis in the frontal plane at 150 degrees.

aVL (augmented unipolar left) LA is the positive connection and the negative is the combination of RA and LL. This lead defines an axis in the frontal plane at -30 degrees.

aVF (augmented unipolar foot) LL is the positive connection and the negative is the combination of RA and LL. This lead defines an axis in the frontal plane at 90 degrees.

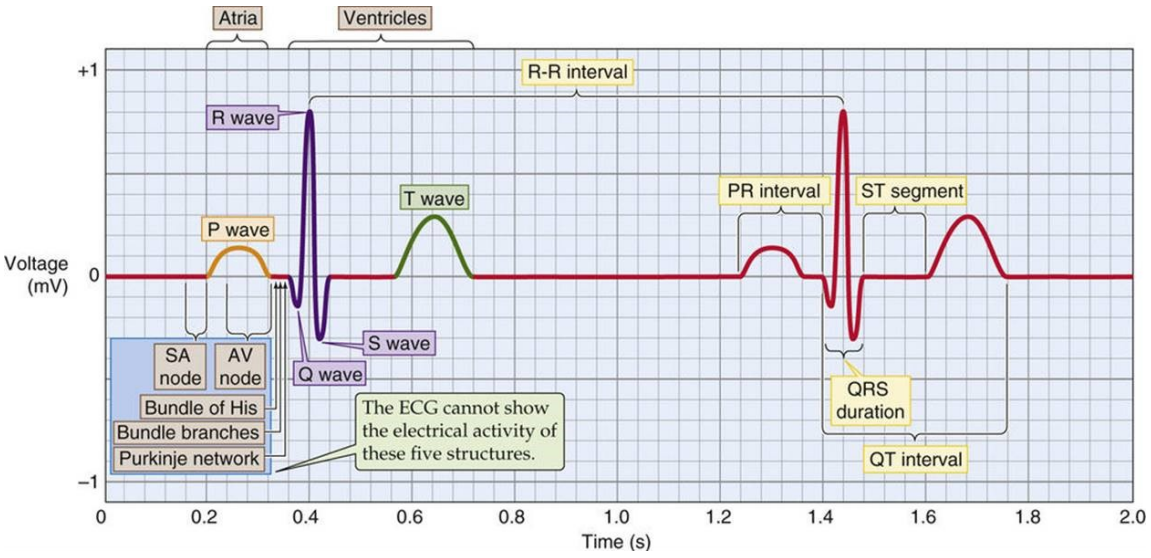


Source: https://cdn.shopify.com/s/files/1/0059/3992/files/Image_6.png?v=1476241491

Figure 2.4: Graphical representation of Einthoven's triangle

Each deflection in an ECG recording is called a wave. There are up to six waves that can appear in a ECG. The waves are named alphabetically starting with the P wave (see figure 2.5), which represents atrial depolarization. The waves Q, R and S together form the QRS complex that represents ventricular depolarization. The T wave represents the repolarization of the ventricles. Finally, the U wave, which isn't detected most of the times due to its small amplitude, reflects the repolarization of the papillary muscles, a group of small muscles in the ventricles.

As mentioned previously, each lead gives its own perspective about the electrical activity of the heart. That is, each lead will have its own ECG trace. Figure 2.4 shows the expected ECG tracing of each of the limb leads. Our client wanted their device to record lead II because it's the lead that, by itself, gives more information about the electrical activity of the heart.



Source: Boulpaep, Emile L., et al. "Medical physiology a cellular and molecular approach." (2009)

Figure 2.5: Components of an ECG recording

The ECG is usually presented on a special graph paper that is divided into 1 mm^2 grid-like boxes (figure 2.5). For readability, every 5 mm the lines are bolder. Each millimeter in the horizontal scale represents 40 ms and every five represent 200 ms because historically, the paper moved at 25 mm/s. The vertical axis is calibrated so that each millimeter represents 0.1 milivolts.

The recording of an ECG on standard paper allows the time taken for the various phases of electrical depolarization to be measured. As showed in figure 2.5, there are four major ECG intervals:

- RR interval - is defined as the time distance between the peak of two consecutive R waves. It is from this interval that the instant heart beat is computed.
- PR interval - this interval represents the time separating atrial and ventricular depolarization. It is measured from the beginning of the P wave to the first deflection of the QRS complex and has a normal range between 120 and 200 ms (three to five small squares on the ECG paper).
- QRS duration - represents the duration of ventricular depolarization. It is measured from the first deflection of the Q wave until the end of the S wave and has a normal range up to 110 ms (3 small squares on ECG paper).

- QT interval - this interval includes both ventricular depolarization and repolarization times. It is measured from the first deflection of the QRS complex to the end of the T wave at isoelectric line and has a normal range up to 440 ms. This interval gets shorter as the heart rate increases.

- ST segment - this interval represents the time between the ventricles depolarization and repolarization. It starts at the end of the S wave and it ends at the beginning of the T wave and has a normal range between 5 to 150 ms.

It is the analysis of these intervals and amplitudes of the waves that the physicians use to infer about the health of the patient. Alterations in the normal ECG may represent arrhythmias, conduction disturbances, and other life-threatening disturbances. For example, a prolongation of the QRS complex may represent intraventricular conduction disturbances and a tight atrial overload may lead to an increase in the P-wave amplitude.

There is another interval of interest in our work. After a QRS complex (the ventricular depolarization) there is a time window in which it is physiologically impossible that another ventricular depolarization occurs. That time window is called refractory period and is usually of 200 ms [3]. In Chapter 6 we will see the importance of this interval in the construction of the QRS complex algorithm.

Chapter 3

Device Overview

To test the implementation of the compression scheme and to get our own ECG recordings we developed a prototype based on the one that our client used. The device had a PIC32MX570F512L microcontroller (MCU for microcontroller unit) [4] on a Explorer 16/31 Development platform [5], both devices are from Microchip Technology. To read the electrical activity of the heart we used the ECG Click Board [6] from MikroElektronika. Lastly, to connect the device with a smartphone we used the Bluetooth HC-05 transmission module [7] from Velleman. The device is shown in figure 3.1.

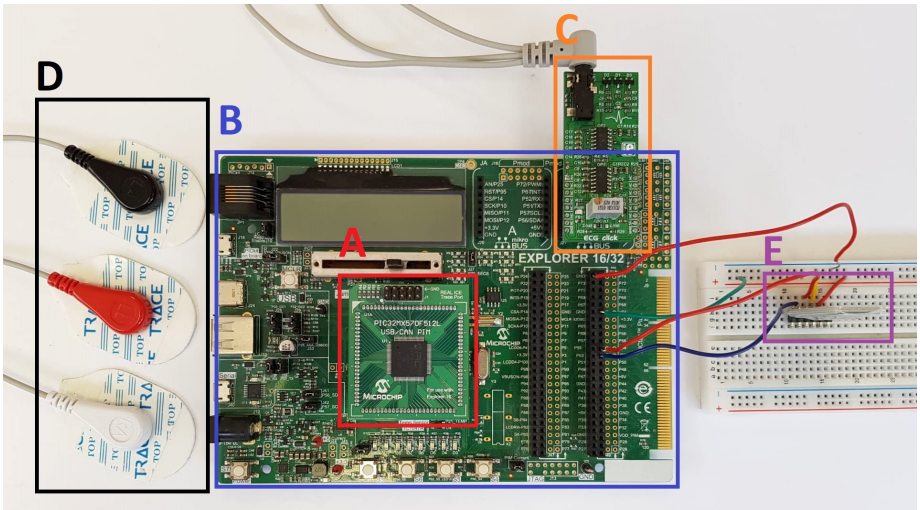


Figure 3.1: Device prototype. A is the MCU, B is the development board, C is the ECG click, D are the electrodes and E is the transmission module

To deploy the software in the MCU we adopted the MPLAB X Integrated Development Environment (IDE) [8] with the MPLAB XC32 Compiler [9].

The ECG click measures the electrical activity of the heart through the three electrodes taped to the skin of the patient. In the ECG click, the signal received by the electrodes is processed to remove certain frequencies of noise and amplify the signal. The ECG click is built in a way to produce the lead II ECG.

The signal is then sent to the PIC32 through the analog port present in the mikroBus A of the Explorer 16/32 board. The PIC32 then reads the analog signal and converts it to digital using the 10-bit Analog-to-Digital converter (ADC).

For the PIC32 to know when to read the values from the ECG Click we set one of the timers of the MCU to emit an alert to the central processing unit (CPU) 200 times per second. These kind of alerts are called interrupts and are used very often in system programming. When an interrupt is set a priority level must be assigned. In the PIC32 the priority level ranges from one, the lowest priority level, to seven, the highest priority level. We set the priority level of the timer to six since we wanted that the highest priority level to be occupied by the interrupt responsible for the transmission of the signal to the smartphone.

To retain the signal we used the non volatile memory of the MCU. This memory has a capacity of 512 kilobytes. Microchip rates the flash memory to endure at least 20000 erase/write cycles. This memory allows to write one word at a time (a word is a 32-bit instruction), or write a row, which is 128 words. Eight rows make a page, which is the smallest unit of memory that can be erased at a single time.

After all, the communication between the MCU and the bluetooth is made through the Universal Asynchronous Receiver Transmitter (UART) module available in the PIC32. The UART receiver interrupt is enabled with the priority level set to seven. A simple prototype application for an android phone was build using the MIT App Inventor [10].

Adapted from: https://www.searchpng.com/wp-content/uploads/2018/12/Samsung-Galaxy-S8-Black-transparent-background-715x1090.png

Figure 3.2: The application has two buttons: the ”Connect” button takes to a list of Bluetooth devices available; the ”Download” button starts the download

The application has a ”Download” button (figure 3.2) which sends the value ”1” through Bluetooth for the device. When that value reaches the UART receiver the interrupt is set and the MCU starts sending the data that has in the flash memory to the application.

Chapter 4

Data Description and Application

The importance of the training dataset/database should not be neglected in the development of algorithms. In this work we have to evaluate the performance of two algorithms. The first is the ECG compression scheme. This is a very specific algorithm since it was built thinking about the device that it would be implemented on, that is, a portable device with a 10-bit precision that has to last a week without recharge. Although the best way to test its performance is through data that was recorded in those conditions, in the development process, it is important to do further testing to understand the limitations, and possible defects of the algorithm. The second is the QRS detection algorithm. Testing this algorithm is more complicated since it needs a database with the positions of the R waves noted. Fortunately, PhysioNet [11] offers free access to a large collection of physiological and clinical data, including annotated ECG databases. It also has datasets of noise that are common on ECG signals, which is of great importance since noise is the principal source of error on the detection of R waves.

4.1 Our Data

As a proof of concept we made a longer recording using the prototype. That record has a total of 465120 data points digitized at 200 samples per second, which is almost 39 minutes of recording. Figure 4.1 shows five seconds of that recording.

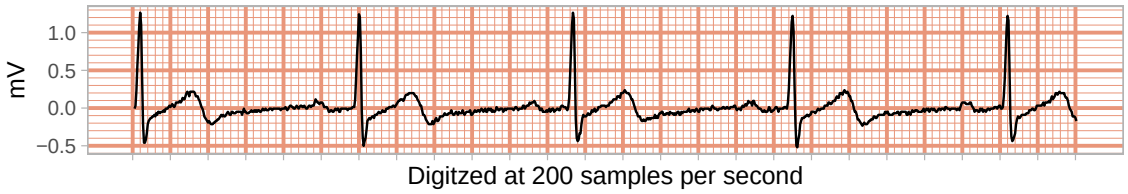


Figure 4.1: Segment of the long recording from the prototype

This record was important to have a real example of the expected compression that we would get with our compression algorithm in recordings from the device.

4.2 MIT-BIH Arrhythmia Database

The MIT-BIH Arrhythmia Database [12] [13] was first released in 1980 and contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979. The recordings were digitized at 360 samples per second per channel with 11-bit resolution over a 10 mV range. Each record of the dataset was independently annotated by two or more cardiologists. Nowadays this database is available in PhysioNet. Figure 5.1 shows five seconds from one of the records from the database.

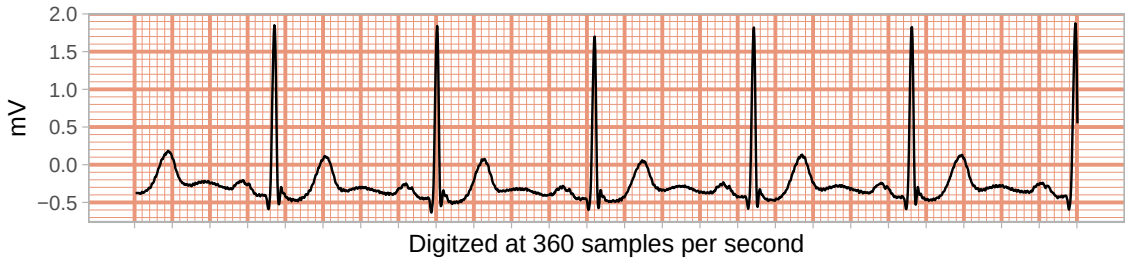


Figure 4.2: Segment of the record 103 from the MIT-BIH Arrhythmia Database

In most records, the first channel is a limb lead II, and the second channel is usually the chest lead V1 (occasionally V2 or V5, and in one instance V4). Since our device is meant to record limb lead II we only used the first channel of each record. Although in records 102 and 104, the first channel is lead V5 instead of lead II (because of surgical dressings on the patients) we still used these records in our analysis.

An important aspect of an QRS detection algorithm is its ability to deal with noise. In PhysioNet is also available a database that includes three half-hour recordings of noise typical in ambulatory ECG recordings, the MIT-BIH Noise Stress Test Database [14] [15]. The three types of noise (figure 4.3) are classified by source rather than by its frequency-domain characteristics. The three categories are:

- Baseline wander (BW), is the familiar low-frequency signal usually caused by motion of the subject or of the leads.
- Electrode motion (EM) artifact is often the most difficult to handle since it can closely mimic elements of the ECG signal. It is usually the result of intermittent mechanical forces acting on the electrodes.
- Muscle noise (MA), has a spectrum which overlaps that of the ECG, but which extends to higher frequencies.

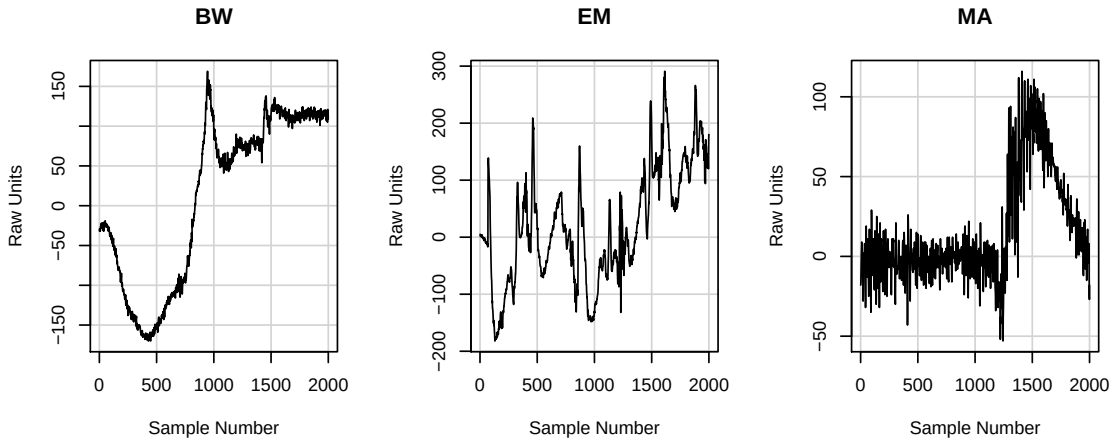


Figure 4.3: Samples of the three noise types from the MIT-BIH Noise Stress Test Database

We created a new database that was built by adding the three types of noises to each record of the MIT-BIH Arrhythmia Database. That database has a total of 144 datasets.

To develop and test the compression scheme we used the record from our prototype and a rescaled version of the MIT-BIH Arrhythmia Database. The rescaled was made to make the data have a precision of 10-bits, to do that we multiplied all the values by $1023/2047$ and round them to the nearest unit.

For the development of the QRS detection algorithm we used the original MIT-BIH Arrhythmia Database. In order to understand the limitations of the algorithm we performed and analyzed the detection on the contaminated MIT-BIH Arrhythmia Database.

Chapter 5

Compression Scheme

The oldest ECG compression algorithm available in the IEEE digital library [16] dates back to 1975 [17] and, although storage space became relatively cheap since then, the compression methods have gained importance in recent years. We are producing more data than ever and as more devices are connected to the internet (IoT) the tendency is that the rate of data produced everyday increases. So in the modern era data compression is mandatory either to reduce storage requirements or to improve transfer speeds.

There are mainly two types of data compression: lossless compression, where all the original data remains intact and lossy compression, where there is a corresponding trade-off between preserving information and reducing size.

In our case we didn't want to lose any information of the original signal, since any small detail in the ECG, could be important to diagnose a heart disease. So our algorithm had to be lossless.

By default PIC32 saves the values in 32 bits, so an ECG with a sampling rate of 200 Hz for seven days would generate almost half a gigabyte of data.

5.1 Signal Compression

Naturally we will need to have an algorithm that works with any possible data output from the device, but in designing an approach it is good practice to consider the normal output. When we look at a typical ECG waveform we mostly see a reasonably flat line and then the sudden activity of the QRS complex followed again by a relatively flat line. This behavior suggests that the change in voltage between consecutive samples will vary a lot less than the absolute voltages in the ECG. In El B'charri et al. [18] the authors developed

a lossless compression and decompression scheme that took advantage of that property. We based our approach on their work.

The implemented algorithm is divided in three sequential main parts: delta encode, signs encode and data encode. Since the compression was implemented in the MCU the code presented in this section is written in C.

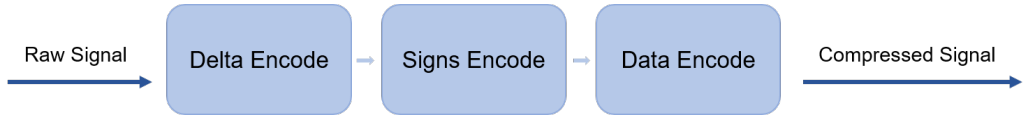


Figure 5.1: Signal compression diagram

Delta Encode

Delta encoding is a widely used technique in compression algorithms. The idea is to compute the difference between consecutive samples and let the first value remain unchanged so that the signal can be reconstructed later. To make the compression in real time we need to define a size for the block that will be compressed at a time. We choose a window of size 255 samples. The reason to choose this window size will be clear later in this section.

Figure 5.2 has two plots: the top one corresponds to the first five seconds of the dataset that was recorded with the prototype (is the same signal plotted in figure 4.1 but in raw units); and the second plot shows the result of subtracting two consecutive samples (and removing the first value).

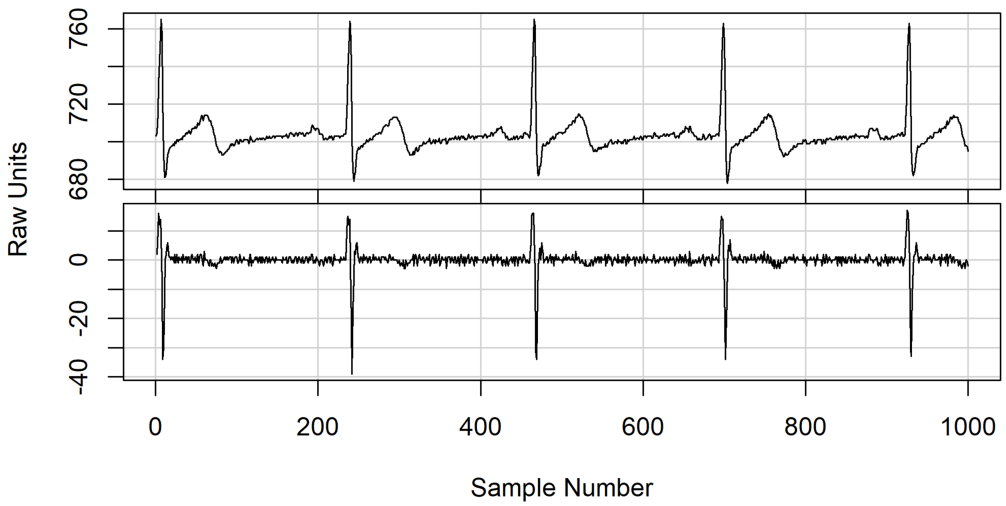


Figure 5.2: First five seconds of the original signal recorded with the prototype (top panel) and the signal obtained by differentiating the original signal (lower panel)

So we see that, by taking the consecutive differences, we have not only reduced the overall variability but the distribution of the values becomes much tighter. All this means that storing the consecutive differences will occupy far less memory than the original values. In fact for the majority of the observations we can pack two observations into one byte.

As a result of the compression we want to output an array with 8-bit numbers. Since the first value is a 10-bit number we decompose it into the quotient and remainder of the division by 69. By doing that we are assuring that the quotient is at maximum 14, which is represented by 0E in hexadecimal, and the remainder is equal or less than 68 which can be written in 8-bits. Further note that by doing that division we expanded the size of the array to be compressed to 256. This division is more important than it seems, as we will explain later.

```
int16_t delta[256];
uint16_t i;

delta[0] = floor(stream[0] / 69);
delta[1] = stream[0] % 69;

for(i = 1; i < 255; i++){
    delta[i + 1] = stream[i] - stream[i - 1];
}
```

Listing 5.1: Delta encoding

Signs Encode

To save the signs of each value in a more compact way we encode them in thirty two 8-bit numbers. For each block of eight data points from the delta array we assign a integer number. If the value is positive we assign a 0 otherwise we assign a 1. To clarify, consider the array d presented in figure 5.3.

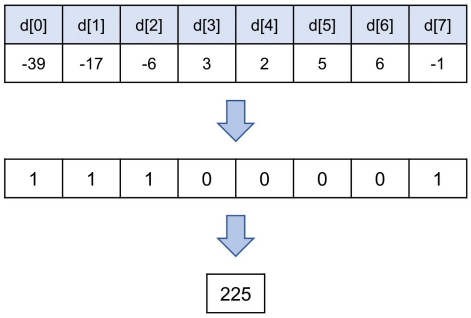


Figure 5.3: Illustrative example of sign encoding step

This step is important because it allows us to work with unsigned integers, so we can fit bigger numbers in less bits. While encoding the signs, the absolute values of the delta array are saved in another array.

```
uint8_t sign = 0x00;
uint8_t signs[32];
uint8_t abs_delta[256];

uint16_t j;

for(i = 0; i < 256; i += 8)
{
    for(j = i; j < i+8; j++)
    {
        if (delta[j] < 0)
        {
            sign ++;
            abs_delta[j] = - delta[j];

        } else
        {
            abs_delta[j] =  delta[j];
        }
        sign <<= 1;
    }
    signs[i/8] = sign;
}
```

Listing 5.2: Sign encoding

Data Encode

This is the part of the algorithm in which most of the compression is made. As mentioned before, most of the values fit in 4-bit unsigned integer. Since the smallest object that can be saved in the memory is a byte, we encode the data in a new array. That new array is constructed by either concatenating or not two consecutive samples.

For each pair of values in the first array (without overlapping) we check if both values fit in a 4-bits, if that's the case shift the first value 4-bits to the left and concatenate the two values. For example, if we want to concatenate a 10 (0A in hexadecimal) and a 2 (02 in hexadecimal) the result would be 162 which is A2 in hexadecimal. Since most of the values "fit" in 4-bits we see major improvement in compression.

In the reconstruction we need to know which values were concatenated, for that, each time a pair is merged a 1 is attributed if they aren't merged a 0 is attributed. For each 16 values we will have eight of these flags that are saved in a 8-bit number called the guide. After all the values are encoded we will have sixteen guides.

To clarify this step consider the example presented in figure 5.4.

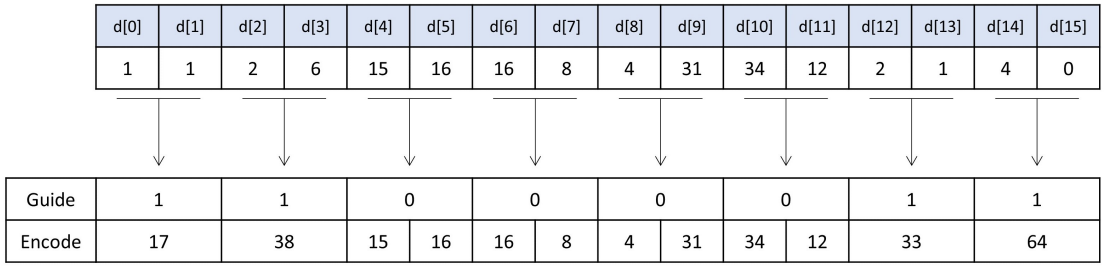


Figure 5.4: Illustrative example of data encoding

In the example presented in figure 5.4 the guide would be saved as 11000011 in binary which is 195 in base-10.

Finally, after encoding all the block of data, we will have three different arrays to save: the signs array with 32 bytes, the encoded data array with size that will vary according to the concatenation possibilities, and the guides array that has 16 bytes. The order in which we store the arrays is of particular importance. Considering we can determine the size of the encoded data array from the guides array, we save them in the following order: guides array, encoded data array and signs array. By saving in that manner the reconstruction procedure is really simple.

5.2 Signal Reconstruction

Since the reconstruction phase is meant to be done in the cloud the code presented in the next section is written in R, which is a language available in most of the cloud storage services.

After the data is uploaded to the device we can start the signal reconstruction. The decompression scheme follows the reverse logic of the compression.

The first sixteen values of the compressed signal are guides. Since the guides hold the information if the value is the merge of two samples or a pair of single samples we can decode the encoded data.

```
guides <- stream[block:(block + 15)]
pointer <- 1
delta <- NULL

block <- block + 15

for(guide in guides){
  for (j in 8:1) {
    if (bitwAnd(guide, bitwShiftL(1, j - 1)) == 0) {
      delta <- c(delta, stream[block + pointer])
      pointer <- pointer + 1
      delta <- c(delta, stream[block + pointer])
      pointer <- pointer + 1
    } else {
      delta <- c(delta,
                 bitwShiftR(stream[block + pointer], 4),
                 bitwAnd(stream[block + pointer], 0x0F))
      pointer <- pointer + 1
    }
  }
}
```

Listing 5.3: Data decoding

By the time we got all the data decoded we know that the next thirty two values hold the information of the signs. So we can add the signs to the values of the delta array.

```
signs <- stream[(blockstart + runpointer):(blockstart + runpointer + 31) ]
signvect <- NULL

for(sb in signs){
  for (j in 8:1) {
    if (bitwAnd(sb, bitwShiftL(1, j - 1)) == 0) signvect <- c(signvect, +1)
    else signvect <- c(signvect, -1)
  }
}
```

Listing 5.4: Sign decoding.

After that we need to reconstruct the first value of the signal by multiplying the first value of delta by 69 and add the value in the second position in the delta array. After that we just need to do the cumulative sum of the delta array. By doing that we decompressed the first block of our sample.

Then we move to the next block and repeat the process. This process is repeated until all data is decompressed.

5.3 Exception Handler

Although very uncommon, is possible that in the delta encoding phase the difference between two consecutive samples cloud be greater than 255, which is the biggest number that an 8-bit unsigned integer can allocate. In our tests this only happened in the presence of a lot of noise. So we had the need to implement an exception handler to take care of these cases. If at any point in the delta encoding the difference between two consecutive samples is greater than 255 we save in the PIC32 non volatile memory the value 255, which is FF in hexadecimal, followed by each sample of that block in two bytes. For example, the value 789 would be saved as {3,21} in the memory. Note that in the normal compression the first compressed value can never be greater than 239 since we divide the first sample by 69.

In the decompression process if we find the value 255 at the beginning of a block we know that the block was compressed by the exception handler and the decompression is straightforward.

5.4 Results

The quality of a compression algorithm is measured by two important components: the compression ratio and the reconstruction error.

As mentioned before, our algorithm is lossless so the reconstructed signal is guaranteed to be the same as the original one. The only way that some signal could be lost is if the number of samples to be compressed isn't a multiple of the compression block (in our case is 255), otherwise, the reconstruction error is zero. This limitation can easily be overcome by padding the sample to always be a multiple of 255. In our device, this padding is made by waiting until the last block to be compressed is complete, since we have a sampling rate of 200 Hz this padding would never take more than 1.27 seconds.

The compression ratio (CR) is the fraction between the size of the file with the original signal and the size of the file with the compressed signal. It is computed by:

$$CR = \frac{B_o}{B_c}, \tag{5.1}$$

where B_o is the number of bits in the original file and B_c is the number of bits in the compressed file.

To evaluate our algorithm, we run the compression scheme in the records of the rescaled MIT-BIH Arrythmia database. This database was rescaled to have a 10-bit precision. Since each record from the database has $650,000$ data points, which isn't a multiple of 255, we left out the last five samples of each record. As previously stated, by default the MCU saves the values in 4 bytes. By assuming that each record of the database would take 2.6 megabytes of data and the database would require a total of 124.8 megabytes. After running the compression in all the records we had approximately 22.8 megabytes of data, making an overall compression ratio of 5.61:1 (the average compression ratio is the same as the overall). The lowest compression ratio obtained was of 5.39:1 from record 213 and the highest was 5.76:1 from record 121. Figure 5.5 shows the distribution of the compression ratios.

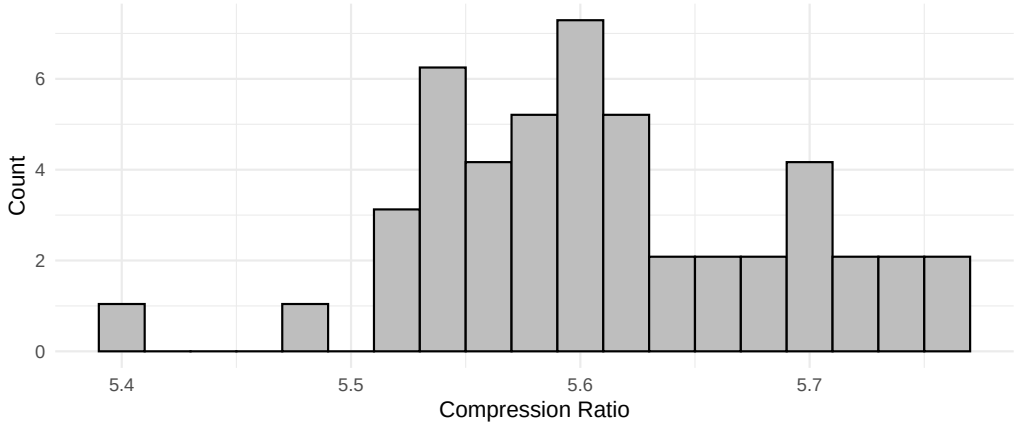


Figure 5.5: Histogram of the compression ratios obtained in the MIT-BIH database

The compression ratio of our algorithm is dependent of the amount of numbers that fit in 4-bits after the delta encode. One of the factors that mainly influences the compression ratio is the amount of QRS complexes in the signal. Figure 5.6 shows the relation between the number of QRS complexes and the compression ratio in the database.

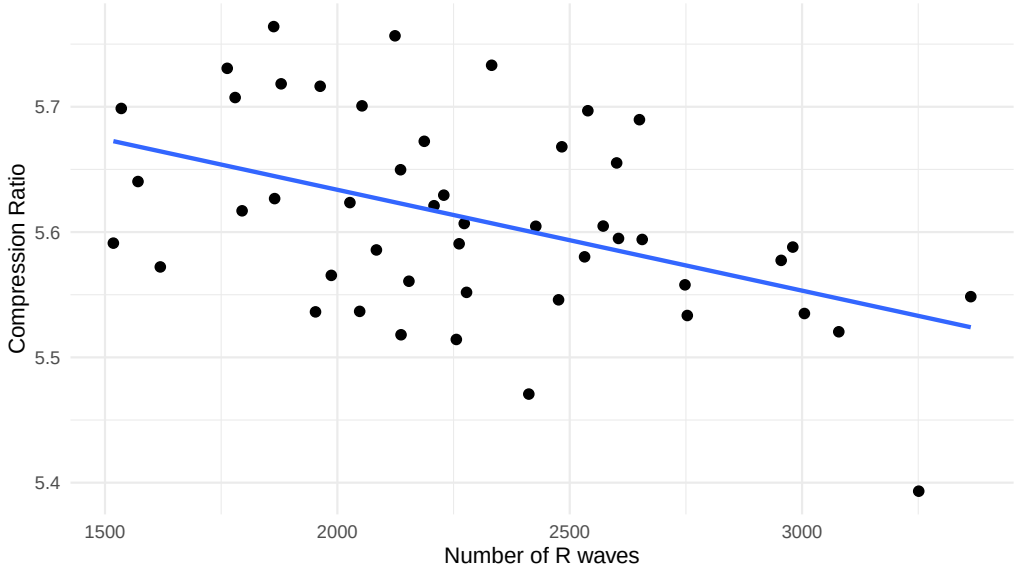


Figure 5.6: Relation between number of R waves and Compression Ratio in the database

Another phenomenon that will affect the compression ratio is the noise artifacts present in the signal.

When we tested the decompression all the data were reconstructed as expected.

We also run the algorithm with the data recorded with the prototype device. The record has a total of $465,120$ that, by default, would be saved in 32-bit numbers, making a total around 1.86 megabytes. When we run the compression we get an array with $329,385$ 8-bit values, which would occupy $329,385$ bytes of memory. Resulting in a compression ratio of approximately 5.65. So the vector resulting from the compression only occupied 18% of the original signal.

Assuming that this compression ratio would maintain in a week, instead of needing half gigabyte of memory we would only need about 86 megabytes.

5.5 Final Comments

Note that the compression scheme present was thought for an ECG in its raw format. Other publications like El B'charri et al. [18] and Mukhopadhyay et al. [19] present better compression ratios, 18.84:1 and 7.18:1 respectively, but they where made to compress data from the ECG in its voltage format. In the MIT-BIH database the values in the voltage format range from -5 to 5 mV with a precision of three decimal places. Those numbers usually are saved in 8 bytes, which gives bigger margin for greater compression ratio. Our

algorithm could be easily adapted to perform the compression of the ECG signal in the voltage format.

Chapter 6

QRS Complex Detection

Most of the Holter monitors have a built in QRS detection algorithm so that they export the signal with the detected R peaks. Our goal was to detect cardiac problems like tachycardia or arrhythmia in real time and send immediate feedback to the user. With that in mind we started to study the possibility of implementing a real time QRS complex detection in the MCU. We came up with an algorithm that was inspired by the one that Pan and Tompkins [3] published in 1985. Like them, we divided the algorithm in three parts: signal processing, peak detection and T and P wave discrimination.

To test and develop the algorithm we used the software R.

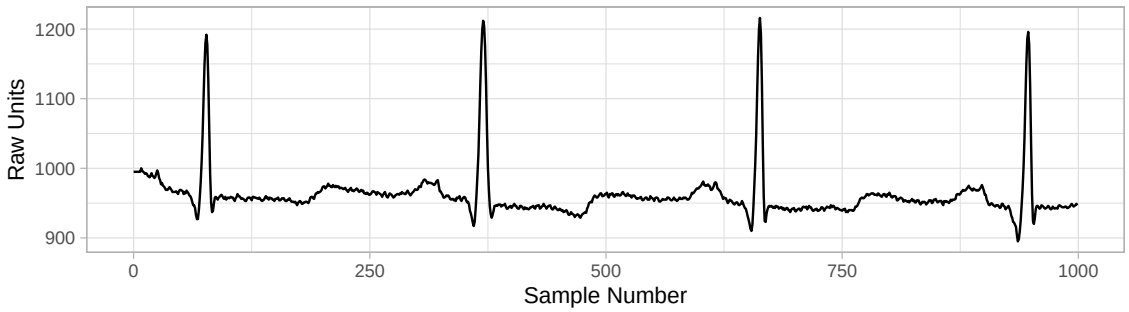


Figure 6.1: Sample of a signal from the MIT database

6.1 Signal Processing

The signal resulting from the read of the ECG Click passes through a sequence of processing steps. The first step is the computation of the five-point derivative (also known as 2 point central difference) of the signal. The difference equation is given by

$$y[n] = -x[n-2] - 2x[n-1] + 2x[n+1] + x[n+2], \tag{6.1}$$

where $x[n]$ represents the nth value of the signal x and $y[n]$ is the nth value of the derivative vector. By doing this we get information about the slope of each wave. Since the slope of the R wave is typically much greater than the slopes of the P and T wave this is a very important step. The delay of this operation is two samples.

If we compute the difference of the signal in figure 6.1 we get the signal presented in 6.2.

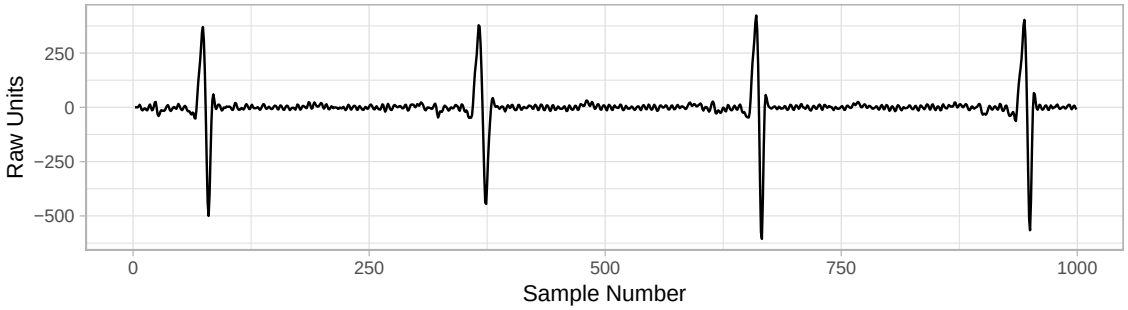


Figure 6.2: Output of differentiator

Then, every value of the differentiated signal is squared. The equation of this operation is

$$y[n] = (x[n])^2. \tag{6.2}$$

This amplifies in a nonlinear form the highest peaks in the output of the derivative signal. Note that by squaring all the peaks turn positive, so it amplifies both positive and negative deflections. By squaring every point of the wave presented in figure 6.2 we get the signal presented in figure 6.3

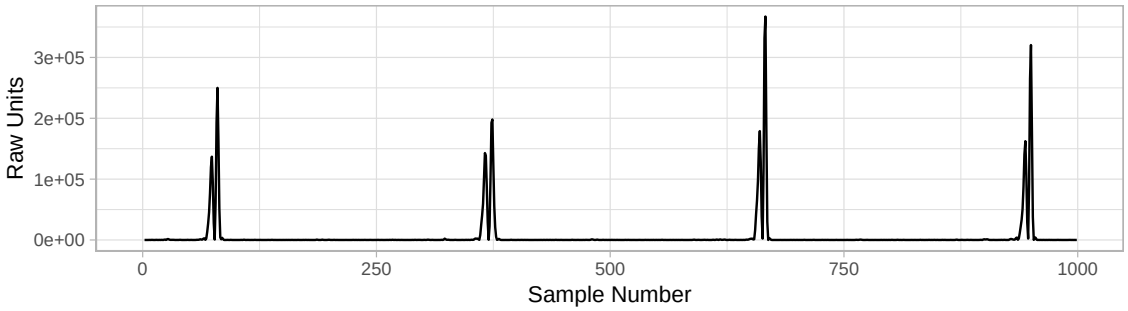


Figure 6.3: Output of squaring

The detection could be performed on the squared signal, but in order to attenuate the detection of P and T waves as QRS complexes we apply a moving-window integration to the squared signal. The equation for the moving window integration is given by

$$y[n] = \frac{1}{N}(x[n - [N-1]] + x[n - [N-2]] + \cdots + x[n]), \tag{6.3}$$

where N is the number of samples in the width of the integration window. The width of the integration window is of great significance. Since the goal is to reduce the amplitude of the T and P waves, the width of the integration window is chosen to be the size of the widest possible QRS complex in a regular sinus rhythm (between 60 - 100 beats per minute). In that case the QRS takes less than or equal to 0.11 seconds. Since the dataset was recorded at 360 samples per second N is approximately equal to 40 samples. Figure 6.4 shows the result of applying equation 6.3 to the signal in figure 6.3.

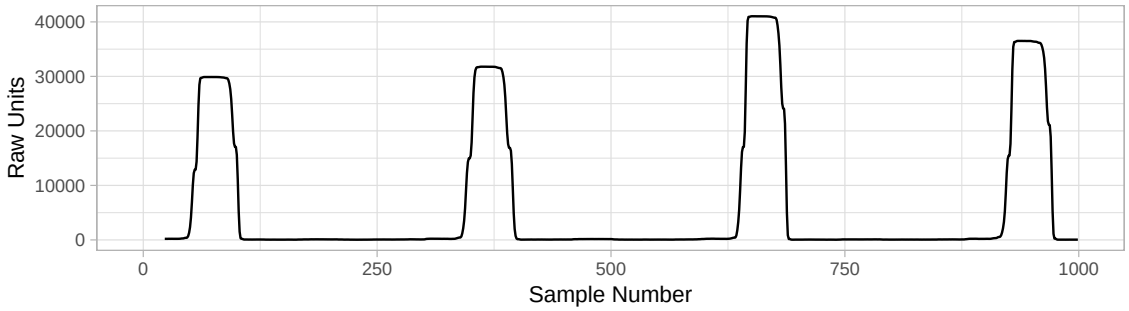


Figure 6.4: Output of moving window integration

The delay of this operation is of the size of the integration window.

The function presented below does all the signal processing. It takes the raw signal as input and outputs the integration wave.

```
singalProcessing <- function(Signal, QRS_width){
  # differentitior
  dif <- data.table::shift(Signal, c(2,1,-1,-2))
  Dif = - dif[[1]]- 2*dif[[2]] + 2*dif[[3]] + dif[[4]]

  # squaring
  Sqr = Dif^2

  # moving window integration
  Flt <- rep(1/QRS_width, QRS_width)

  Int <- floor(filter(Sqr, Flt, side=2))

  return(Int)
}
```

Listing 6.1: Function signalProcessing

6.2 Peak Detection

The detection of the QRS complexes is made in the integration waveform using an adaptive threshold technique. This section can be divided in three main parts: starting phase, training phase and detection phase.

Starting Phase

The starting phase finds the first signal peak (SP) and the first noise peak (NP). For that, we assume that the maximum value of the first 500 samples is a signal peak. Than we find an estimate of the noise peak. We call a noise peak any deflection in the signal that is not related to the QRS complex (for example a T or P wave). Since after each QRS complex exists a 200 ms refractory period in which is physiologically impossible to happen another ventricular depolarization, we search for that noise peak in the window between the last SP position plus the QRS width and the SP position plus the refractory period. An example of the starting phase is presented in figure 6.5.

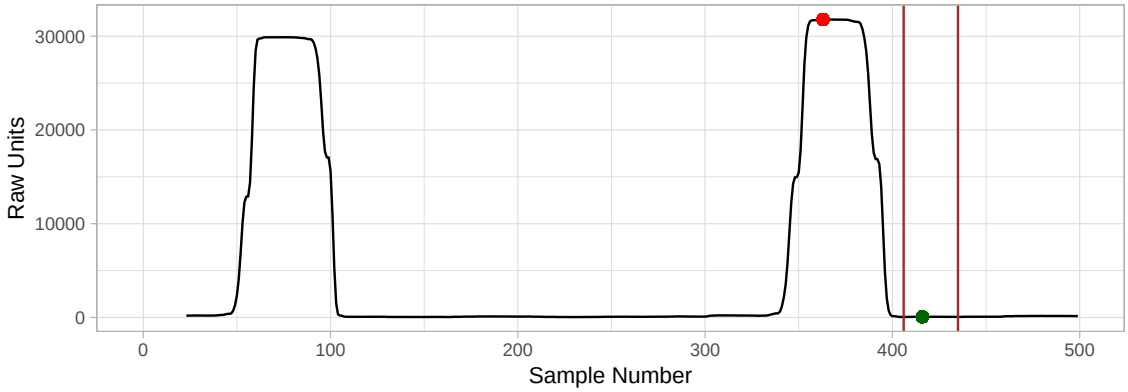


Figure 6.5: Example of the starting phase of the detection. The red point represents the maximum of the first 500 samples, the vertical brown lines represent the window in which the noise peak is searched, the green point is the established noise peak

The threshold (TH) is based on the values of SP and NP. The idea is that the threshold floats over the noise of the signal. The threshold is computed using the following equation

$$\text{TH} = \text{NP} + \text{S(SP - NP)}, \tag{6.4}$$

where S is the sensibility of the detection. This value should be adjusted based on the signal to noise ratio (SNR) that is expected. For example, the SNR in an ECG form lead-II is expected to be greater than the one from the lead augmented vector left (aVL). We set the sensibility to 0.2.

The function startDetection (presented below) computes the first threshold.

```
start_detection <- function(Signal, QRS_width, Refractory, Sensibility){
  # find the first signal peak
  SP_position <- which.max (Signal[1:500])
  SP <- Signal[SP_position]

  # find the first noise peak
  NP <- max( Signal[(SP_position + QRS_width):(SP_position + Refractory)])

  # compute the threshold
  th <- NP + Sensibility * SP

  return(th)
}
```

Listing 6.2: Function startDetection

Training Phase

After getting the first threshold we start the training phase. This phase lasts until nine signal peaks are found. We start at the first value of the integration waveform and increment by one the position of search until the value of the wave is greater than TH. When that happens, we search for the maximum value in the next 0.11 ms (which is the QRS window). That value is established as signal peak and from there we detect the noise peak in the same manner as in the starting phase. The threshold is updated in the following equation,

$$\text{TH} = \text{NPeak} + \text{S(SPeak - NPeak)}, \tag{6.5}$$

where SPeak and NPeak are the estimate of the signal and noise peak respectively. Those are computed by the followinw equations:

$$\text{SPeak} = 0.125\ \text{SPeak} + 0.875\ \text{SP}, \tag{6.6}$$

$$\text{NPeak} = 0.125\ \text{NPeak} + 0.875\ \text{NP}. \tag{6.7}$$

By using the previous equations, the signal and noise peak estimates keep some historic of the last peaks.

Note that equation 6.5 is similar to the equation used in the starting phase but instead of using the most recent signal and noise peaks we use the peak estimates.

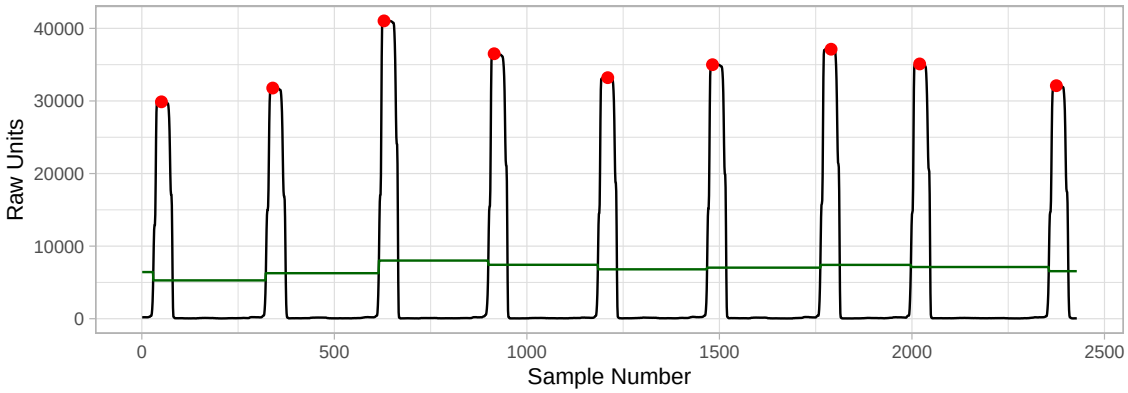


Figure 6.6: Example of the training phase of detection. The red point represents the established QRS complexes and the green line shows the evolution of the threshold value

Note that the threshold is always updated after the refractory period. An implementation of the training phase is presented bellow.

```
position = 1
while(length(SP_list) < 9){
   if (signal[position] > th){
      # find the signal peak
      win <- position : (position + QRS_width)
      SP0 <-  which.max( signal[win] ) + win[1] - 1
      SP_list <- c(SP_list, SP0)

      # find noise peak amplitude
      win <-  (SP0 + QRS_width) : (SP0 + Refractory)
      NP <- max( signal[win])

      # update threshold
      SPeak <- 0.125 * SPeak + 0.875 * signal[SP0]
      NPeak <- 0.125 * NPeak + 0.875 * NP

      th <- NPeak + Sensibility * (SPeak - NPeak)

      th_list <- c(th_list, th)

      position <- position + Refractory
   }
   position <- position + 1
}
```

Listing 6.3: Training phase

Detection Phase

As soon as we have the first nine signal peaks, we enter in the detection phase. In order to reduce false negatives and for the algorithm to be able to adapt to quickly changing or irregular heart rates, we keep track of a specific RR-interval average (RR). Not the simple average of eighth most-recent beats but the average of the eighth most-recent beats that fall in a specific interval

$$RR = \frac{1}{8}(RR'[n-7] + RR'[n-6] + \cdots + RR'[n]), \tag{6.8}$$

where $RR[n']$ is the most recent intervals that fall between the acceptable low and high RR-interval limits. These limits are given by

$$RR_{low} = 0.92RR, \tag{6.9}$$

$$RR_{hig} = 1.16RR. \tag{6.10}$$

If all the eight most recent beats fall within RR_{low} and RR_{hig} we consider that the heart is beating at a normal sinus rhythm. If at any point the algorithm doesn't detect any candidate in the window of the size of 166% of RR we perform a back search. In the back search we use a threshold that is half the size of the current TH. The maximum point that is greater than half of TH in the interval between the last refractory period and the current position is considered a signal peak.

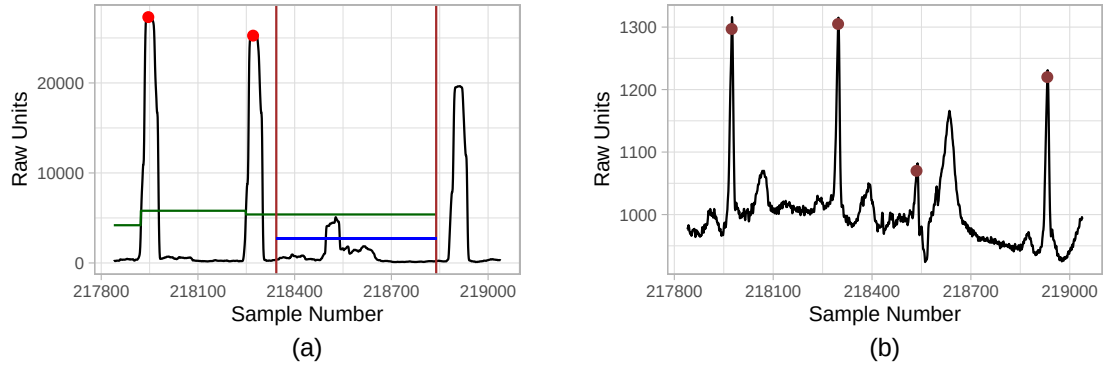


Figure 6.7: Example of back search.(a) the back search in the integration waveform: the red dots are peaks that the algorithm already established as signal peaks, the brown vertical lines represent the search window, the green line is the evolution of TH and the blue line is half of TH.(b) the raw signal with the noted QRS complexes from the database

Figure 6.7 illustrates how the back search works. Also in 6.7 is interesting to notice that although the amplitude of the T wave, in the raw signal, is almost two times greater than the amplitude of the R wave, in the integration waveform, the biggest peak is from the R wave.

When a signal peak is detected with the back search we update the estimate of SPeak by the following equation

$$\text{SPeak} = 0.25\ \text{SPeak} + 0.75\ \text{SP}, \tag{6.11}$$

the noise peak estimate and the threshold are still updated by equations 6.7 and 6.5 respectively.

```
dist_beat <- position - SP0
if (dist_beat >= RR_missed){
  th2 <- th / 2

  # window of search
  win <- (SP0 + Refractory) : position

  if ( any(signal[win] > th2) ){
    # find the signal peak
    SP0      <- which.max( signal[ win ]) + win[1] - 1
    SP_list  <- c(SP_list, SP0)

    # get the position of the previous signal
    SP1 <- SP_list[length(SP_list) - 1]

    # compute instant beat
    i_beat  <- SP0 - SP1

    # find noise peak
    win <- (SP0 + QRS_width) : (SP0 + Refractory)
    NP  <- max( signal[win] )

    # update instant beat average and RR intervals
    if ( i_beat > RR_limits[1] & i_beat < RR_limits[2] ){
      inst_beat <- c(inst_beat, i_beat)
      average_beat  <- mean( inst_beat[ length(inst_beat) - (9:1) ] )

      RR_limits <- c(0.92, 1.16) * average_beat
      RR_missed <- 1.66 * average_beat
    }

    # update threshold
    SPeak <- 0.25 * SPeak + 0.75 * signal[SP0]
    NPeak <- 0.125 * NPeak + 0.875 * NP

    th <- NPeak + Sensibility * (SPeak - NPeak)

    th_list <- c(th_list, th)

    # update position
    position <- SP0 + Refractory
  }
}
```

Listing 6.4: Back Search

There are cases in which the signal peak is abnormally higher than all the others. Due to the importance that we give to the most recent peak, the value of TH gets much greater

the next peaks. An exception handler was implemented to deal with those cases. If the algorithm doesn't find any signal peak in a window of five seconds we restart the threshold by doing a new estimate of SP and NP. The new estimates are made as in the starting phase but the window of 500 data points starts after the refractory period of the last signal peak.

6.3 T and P Wave Discrimination

If a peak is identified between 0.2 and 0.36 seconds of the last signal peak there are three possibilities to judge:

- the first candidate is a P wave and the second a QRS complex ((a) and (d) in figure 6.8);
- the first is a QRS complex and the second is a T wave ((b) and (e) in figure 6.8);
- or they are both QRS complexes ((c) and (f) in figure 6.8).

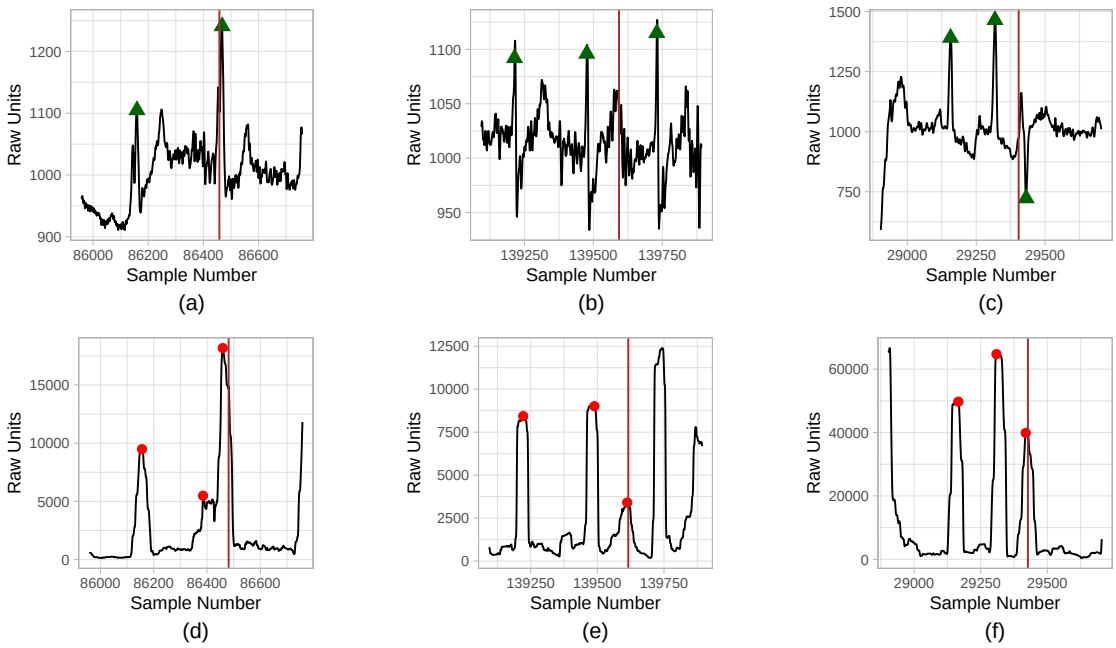


Figure 6.8: The three possible cases in P and T wave discrimination. (a), (b) and (c) are the result of the processing of the signals in (d), (e) and (f) respectively. The brown vertical line represents the current position in the detection. The points in red in the integration waveform are peaks that the algorithm has detected. The green triangles in the raw signals are the QRS complexes annoted in the database

In those cases we use the following logic:

- if the first candidate has less than half the amplitude of the second candidate, the first is considered a P wave and the second a QRS complex;
- if the second candidate has less than half the amplitude of the first candidate, the second is considered a T wave and the first a QRS complex;
- if none of the previous cases is true they are both considered QRS complexes.

```
# check if it is a p or t wave
PT_window <- round(Sample_Freq * 0.36)

if (i_beat < PT_window){
    if ( signal[SP0] < signal[SP1]/2 ){
      # SP0 is a T wave
      SP_list  <- SP_list[ -length(SP_list) ]
      position <- position + Refractory
      next

    } else if ( signal[SP0]/2 > signal[SP1] ){
      # SP1 is a P wave
      SP_list <- SP_list[-(length(SP_list) - 1)]
      SP1     <- SP_list[ length(SP_list) - 1]

      i_beat    <- SP0 - SP1
    }
}
```

Listing 6.5: P and T wave discrimination

This procedure helps in the reduction of false positives.

6.4 Results

To test the performance of the algorithm we compared the peaks detected by the algorithm with the annotations from the MIT-BIH Arrythmia database. The results are condensed in table 6.1.

In a total of 109966 beats the algorithm wrongly detected 886 and failled to detect 523 getting an accuracy of 98.83 percent. In figure 6.9 is presented a boxplot with the accuracy, recall and precision obtained in all the records from the MIT-BIH database.

Table 6.1: Results of the QRS complex detection in the MIT-BIH Arrythmia database

Record	FP	FN	No. Beats	Failed Detection (%)
100	0	0	2273	0,00
101	7	4	1865	0.59
102	0	0	2187	0.00
103	0	0	2084	0.00
104	50	16	2229	2.96
105	48	22	2572	2.72
106	1	13	2027	0.69
107	0	9	2137	0.42
108	163	56	1763	12.42
109	2	0	2532	0.08
111	4	2	2124	0.28
112	0	0	2539	0.00
113	0	0	1795	0.00
114	7	1	1879	0.43
115	0	0	1953	0.00
116	5	20	2412	1.04
117	3	1	1535	0.26
118	2	0	2278	0.09
119	0	0	1987	0.00
121	3	1	1863	0.21
122	1	1	2476	0.08
123	0	3	1518	0.20
124	2	0	1619	0.12
200	31	14	2601	1.73
201	0	33	1963	1.68
202	0	7	2136	0.33
203	71	124	2980	6.54
205	0	3	2656	0.11
207	78	192	2332	11.58
208	13	22	2955	1.18
209	4	1	3005	0.17
210	8	41	2650	1.85
212	2	1	2748	0.11
213	1	2	3251	0.09
214	5	5	2262	0.44
215	4	11	3363	0.45
217	1	5	2208	0.27
219	0	1	2154	0.05
220	0	0	2048	0.00
221	1	8	2427	0.37
222	6	2	2483	0.32
223	1	3	2605	0.15
228	58	72	2053	6.33
230	0	0	2256	0.00
231	0	0	1571	0.00
232	9	0	1780	0.51
233	0	4	3079	0.13
234	0	0	2753	0.00
48 patients	591	700	109966	1.17

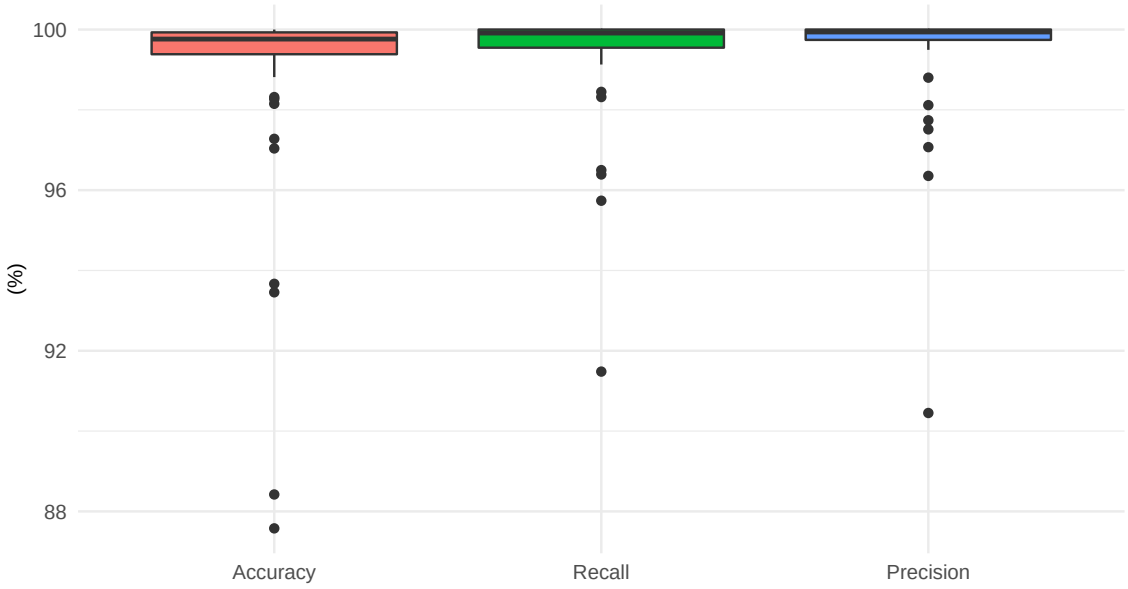


Figure 6.9: Boxplot of Accuracy, Recall and Precision of the QRS complex detection on the MIT-BIH database

the recall is computed by

$$Recall = \frac{TP}{TP + FN}, \tag{6.12}$$

and precision is computed by

$$Precision = \frac{TP}{TP + FP}, \tag{6.13}$$

where TP, FN and FP are the true positives, false negatives and false positives respectively.

From the boxplot we can see that the algorithm has an accuracy above 99% for most of the records. The values of precision and recall tell has that the algorithm has a tendency to generate a few more false negatives than false positives, that is, it doesn't identify a QRS complex when there is one more times than the contrary.

The biggest outliers (in accuracy) presented in the boxplot are from the records 108 and 207. As it can be seen in figure 6.10 (a), the record 108 has unusually tall P waves that the algorithm established as QRS complexes, making a big count of false positives. Record 207 presents stretches that don't have an ECG morphology, that is, it would be

considered noise in normal conditions (figure 6.10 (b)). If we remove this records from the database the total accuracy raises to 99.24%.

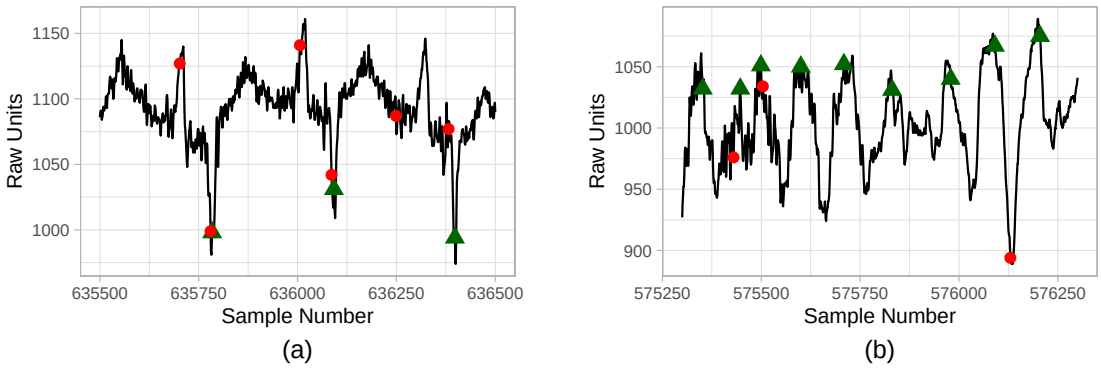


Figure 6.10: Sample of record 109 (a) and record 207 (b). The red dots are the waves that the algorithm established as QRS complexes and the green triangles are the annotations from database

In the other records most of the false negatives are a consequence of a false positive. Note that if a P wave is detected as a candidate and has a slope greater than half of the following QRS complex the algorithm will establish the P wave as a QRS complex and that makes a FN-FP pair in our evaluation.

To test even further the algorithm we added the three noise records from the MIT-BIH Noise Stress Test database to each record of the MIT-BIH database. The images in figure 6.11 shows the first 1000 samples of record 100 from the MIT-BIH database contaminated by the noise.

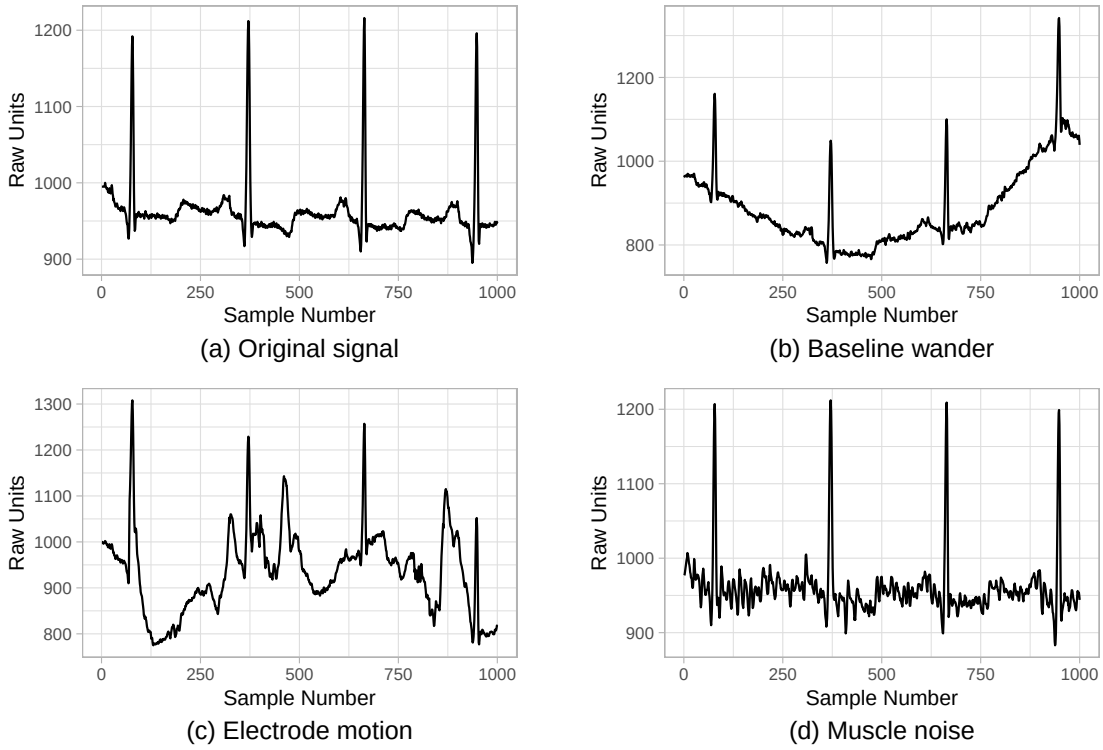


Figure 6.11: Record 100 from the MIT-BIH database with added noise

The algorithm got an overall accuracy of 98.48, 64.36 and 89.90 percent when added the baseline wander, electrode motion and muscle noise respectively. So the baseline wander has almost no impact on the detection and, as expected, the electrode motion has the biggest impact in the detection.

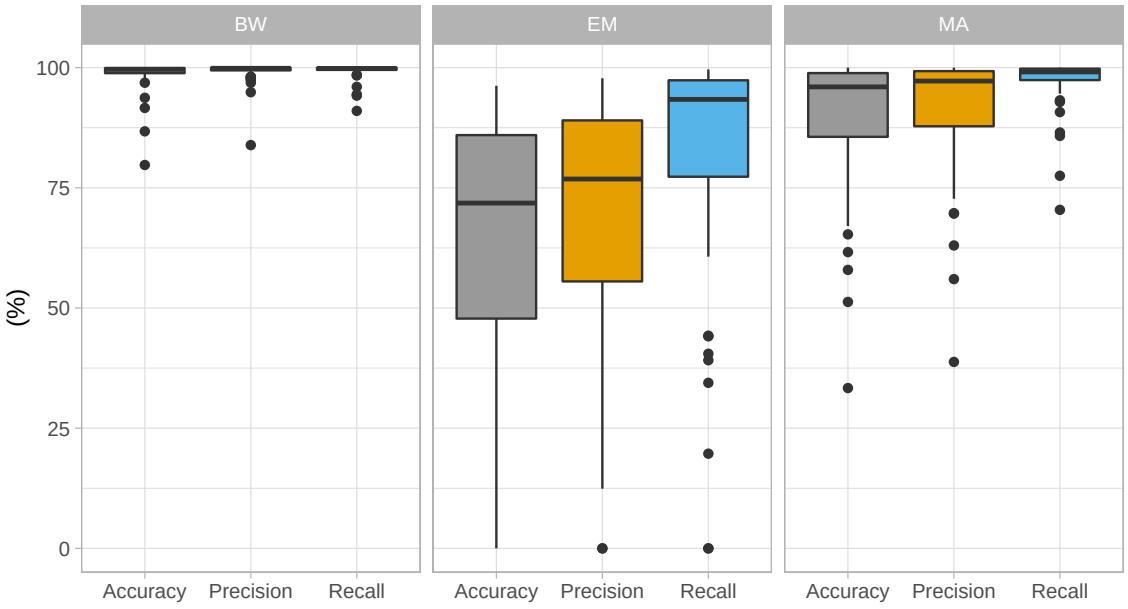


Figure 6.12: Boxplot of the accuracy, precision and recall of detection in the MIT dataset contaminated noise

Looking at the values of precision and recall in the boxplot in figure 6.12 we can see that the algorithm keeps the tendency to generate more false positives than false negatives.

The records that the algorithm had the worst performance under the noise are the records that the QRS complex amplitude is closer to the one from the amplitude of the P and T waves. Note that the only noise attenuation performed by the algorithm is in the moving average integration. One way to improve the detection under noisy conditions would be to implement a band pass filter.

6.5 Final Comments

Although the algorithm published by Pan and Tompkins [3] has a better performance (a detection failure of 0.675 percent in the MIT-BIH database) than the one presented in this chapter, theirs is computationally more complex. They start their algorithm with a low pass filter followed by a high pass filter and perform detection in two signals. In the tests performed by our client the device battery only lasted for three days, so computational power of our device would be limited since to achieve the goal of recording for seven days uninterrupted we would have to underclock the MCU.

Chapter 7

Final Reflections

This project was delayed by our client, but at JTA: The Data Scientists we had many ideas for the future development and improvement of this product.

Although we were approached by our client to consider the compression scheme in their device, we could further adapt the algorithm to be implemented in the cloud which would result in a better compression ratio since the length of the block to be compressed could be bigger. With the number of users that they were expecting to have , a specialized compression scheme would have a major impact on company finances.

The QRS algorithm detection was never implemented on the device but since both algorithms start by differentiating the signal there was the idea of merging them to reduce the power needed by the CPU.

Another idea that was discussed was to build an algorithm to classify each beat of the ECG to be implemented in the cloud. To develop this algorithm we could use the labeled annotations from the MIT-BIH Arrythmia Database. We know that our client would be interested in this effort. Later we could build a simple software where their cardiologists could correspond each heart beat to a cardiac rhythm variation / disturbance. Then we would have a database of annoted signals where we could further improve the algorithm performance.

Bibliography

[1] D. Kasper, A. Fauci, S. Hauser, D. Longo, J. Jameson, and J. Loscalzo, Harrison's principles of internal medicine, 19th ed. McGraw-Hill Education - Europe, 2015.

[2] W. F. Boron and E. L. Boulpaep, Medical physiology: a cellular and molecular approach, 2nd ed. Elsevier health sciences, 2009.

[3] J. Pan and W. J. Tompkins, "A real-time QRS detection algorithm," IEEE Trans. Biomed. Eng, vol. 32, no. 3, pp. 230–236, 1985.

[4] "PIC32MX570F512L," https://www.microchip.com/wwwproducts/en/PIC32MX570F512L, accessed: 2019-09-23.

[5] "Explorer 16/32 Development Kit," https://www.microchip.com/developmenttools/ProductDetails/PartNo/DM240001-2?utm_source=MicroSolutions&utm_medium=Article&utm_content=DevTools&utm_campaign=StandAlone, accessed: 2010-09-30.

[6] "ECG click," https://www.mikroe.com/ecg-click, accessed:2019-09-23.

[7] "HC-05 Transmition Module," https://www.velleman.eu/products/view/?id=435518, accessed:2019-09-23.

[8] "MPLAB X Integrated Development Environment," https://www.microchip.com/mplab/mplab-x-ide, accessed:2019-09-23.

[9] "MPLAB XC Compilers," https://www.microchip.com/mplab/compilers, accessed:2019-09-23.

[10] "MIT App Inventor," http://ai2.appinventor.mit.edu/, accessed:2019-09-23.

[11] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "PhysioBank,

PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals," Circulation, vol. 101, no. 23, pp. e215–e220, 2000.

[12] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH arrhythmia database," IEEE Engineering in Medicine and Biology Magazine, vol. 20, no. 3, pp. 45–50, 2001.

[13] "MIT-BIH Arrhythmia Database," https://www.physionet.org/content/mitdb/1.0.0/, accessed:2019-09-23.

[14] G. B. Moody, W. Muldrow, and R. G. Mark, "A noise stress test for arrhythmia detectors," Computers in cardiology, vol. 11, no. 3, pp. 381–384, 1984.

[15] "MIT-BIH Noise Stress Test Database," https://www.physionet.org/content/nstdb/1.0.0/, accessed:2019-09-23.

[16] "IEEE Xplore Digital Library," https://ieeexplore.ieee.org/Xplore/home.jsp, accessed: 2019-09-23.

[17] N. Ahmed, P. J. Milne, and S. G. Harris, "Electrocardiographic data compression via orthogonal transforms," IEEE Transactions on Biomedical Engineering, no. 6, pp. 484–487, 1975.

[18] O. El B'charri, R. Latif, A. Abenaou, A. Dliou, and W. Jenkal, "An Efficent Lossless Compression Scheme for ECG Signal," INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, vol. 7, no. 7, pp. 210–215, 2016.

[19] S. K. Mukhopadhyay, S. Mitra, and M. Mitra, "A lossless ecg data compression technique using ascii character encoding," Computers & Electrical Engineering, vol. 37, no. 4, pp. 486–497, 2011.