2022-2023

Final Year Project

Presented to obtain the following degree:

ELECTRICAL AND ELECTRONIC ENGINEER
FACULTY OF ENGINEERING – Third Branch
Specialty: Telecommunications

Done by:
Karim Sleiman

Classification of QRS complexes of ECG traces

Under the supervision of:
Dr. Youssef HARKOUS

Defended on the 12th of July in front of the following jury:

Dr. Mohamad Aoudé (President)

Dr. Mazen Ghandour (Member)

Dr. Youssef Harkouss (supervisor)

Preface

This thesis is presented as a partial fulfillment of the requirements for a Diploma in Electrical and Communication Engineering (Telecommunication) at the Faculty of Engineering III of the Lebanese University. The research work was conducted between February and June 2023. The primary objective of this project is to conduct an extensive investigation, incorporating technical aspects and simulation, on the classification of QRS complexes within the domain of biomedical engineering. Furthermore, it aims to explore the advantages associated with the integration of Artificial Intelligence and machine learning techniques specifically tailored for this project.

Acknowledgements

I would like to express my sincere gratitude to Dr. Youssef Harkouss for his unwavering support and guidance throughout this project. His invaluable advice and encouragement have been instrumental in shaping my research journey. I would also like to extend my heartfelt thanks to the members of the jury, the president of the jury Dr. Mohamad Aoudé and Dr. Mazen Ghandour and other members for their presence during the thesis presentation. Special thanks are due to the dedicated instructors and coordinators who have played a significant role in assisting me during this project. I am deeply grateful to Dr. Stephane Binczak, Dr. Stephanie Bricq, and future doctor Mahya Faraji Zamharir for their continuous assistance and unwavering support. Their expertise and commitment have been invaluable in navigating the challenges and complexities of this research endeavor.

Summary

This project is an integrated approach for cardiac analysis, it is provided through a concise overview focusing on the classification of QRS complexes using machine learning algorithms, and a strategic use of algorithms to simulate ECG signals. Following the medical knowledge of the heart and focusing on the electrical cardiac system, entering the biomedical engineering field through this domain was simple through complicated algorithms, they were made easy to use by the python libraries. Understanding the electricity of the heart is not enough to complete the endeavors encountered below. Therefore, in a first chapter, the medical science behind the heart will be explained, emphasizing on the concepts linked to the electrical cardiac system. In the second chapter, the focus is on the strategic use of algorithms and machine learning techniques to achieve great results in the classification process. In a final chapter, the simulation of ECG signals is introduced. Many types of simulations will be present.

Table of contents

List of figures

List of Abbreviations

- **ECG:** Electrocardiogram
- **VT:** Ventricular tachycardia
- **PVC:** Premature Ventricular Contractions
- **RVOT:** Right Ventricular Outflow Tact
- **LVOT:** Left Ventricular Outflow Tact
- **SA:** Sino-Atrial
- **AV:** Atrio-Ventricular
- **Bpm:** beats per minute

GENERAL INTRODUCTION

The heart muscle is one of the most complex parts of the human body, it is essential not only for medical doctors to understand it, but it is also required for any biomedical and scientific engineer to understand it for them to work on the new discovers that are affecting the medical field. The electrical activity of the heart plays a vital role in its proper functioning. One of the activities that are embarking this field is the presence of deep learning inside the biologic systems of the human body. This study in particular shows how the cardiac problems could be existing and cured and how machine learning technics could make this job easier for medical operators. With that being said we will focus on the electrical cardiac system. Understanding the intricate workings of the cardiac electrical system is crucial for diagnosing and treating various cardiovascular conditions. In this research project, we delve into the fascinating field of cardiac electrophysiology to explore the mechanisms underlying the generation and propagation of electrical signals within the heart. The project is divided into three chapters, each focusing on a distinct aspect. The first chapter explains the heart and all of its systems, it focuses on the cardiac electrical system and the cardiac components. The second chapter explains how deep learning technics are used to classify QRS complexes and have a positive impact on the world of heart diseases. The third and final chapter shows three types of simulations that could be put to use in any biomedical engineering problem, and how these simulations could help us achieve greater results in our specific problem.

CHAPTER: I. THE HEART MUSCLE

I.1 The heart of the project

The heart is a muscle, located in the middle of the chest, slightly to the left and responsible of pumping blood around the body. The blood exiting the heart is carrying oxygen and nutrients throughout the body and carries away waste and carbon dioxide.

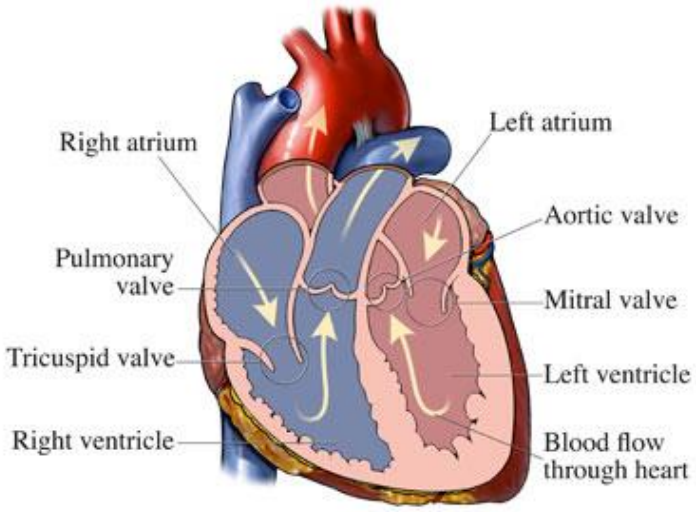


Figure 1 Heart Muscle

Studying the heart requires the knowledge of different systems and functionalities, including: circulatory system, blood vessels, blood pressure and the electrical system. Understanding the complexity of the heart is a must. Nevertheless, we will focus in this chapter on the connection between the anatomy of the heart and how it works and the electrical system of that muscle.

I.2 Circulatory system

The cardiovascular system consists of the heart and the circulatory system. That system takes care of pumping blood in about 5 liters around the body. The right side of the heart receives blood from the body (brain and other body parts). This blood has a low oxygen ratio since the oxygen was used in the body. The fresh oxygen from the lungs enriches this blood and it returns to the left side of the heart being ready to be pumped back into the variety of the body parts.

I.3 Blood vessels

The network of blood propagation through the whole body is the blood vessels: Arteries are the largest blood vessels connected to the capillaries which are also connected to the veins. The blood vessels get smaller as they get further from the heart.

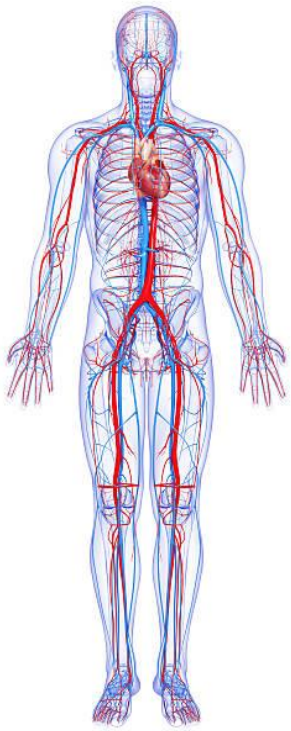


Figure 2 Blood vessels

I.4 Blood pressure

It is the pressure measured inside the arteries. For a healthy blood pumping throughout the body, this pressure should be under control. The blood pressure is measured depending on the heart's pumping action, the size and stretchiness of the blood vessels and the thickness of the blood itself. A heartbeat is a single cycle in which the heart muscle contacts, pumps blood and relaxes. The normal human heartbeat is approximately between 60 and 100 bpm. The pumping cycle is devised into two major steps: Systole is when the heart contracts and pumps blood out of the chambers. Diastole is when the myocardium of the heart relaxes between contractions and the chambers fill with blood.

I.5 Electrical system

Since the heart is a muscle, it receives an electrical signal from the nervous system to function regularly. These signals are the main control actions of the heart. Moreover, the heart is known as a muscle that contracts by its own. Starting the Cardiac cycle requires an electric impulse, this is initiated in the heart itself. That can explain the fact that a human heart can still beat outside the human body. The nervous system therefore controls how fast or slow the heart rate is but cannot control its starting point or ending at that matter. Understanding the electrical system of that muscle requires more detailed explanation. This will be explained more later on.

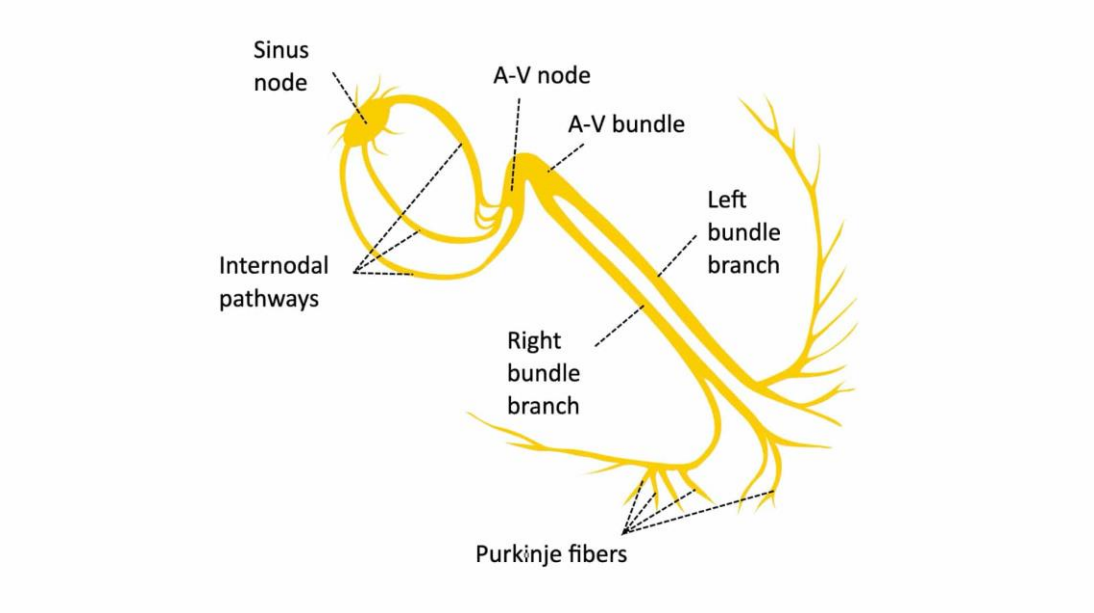


Figure 3 Electrical Cardiac System

I.6 Cardiac Cycle:

As depicted above, a cardiac cycle is devised into two major smaller steps: Systole and Diastole. Between the contraction of the heart, the difference in the pressure in the variety of chambers of the muscle causes the valves inside the heart to open or close accordingly. The valve responsible for filling blood inside the Ventricles (lower two chambers) from the atrium (higher two chambers) is the atrioventricular valves or AV valve. The valve that is responsible of pumping blood into the arteries from the ventricles are the aortic valves. The cardiac cycle can be generally observed in 6 major phases:

Atrial depolarization/Contraction: The cycle is initiated with the action potential in the Sinoatrial node or SA node. This causes the atria to depolarize (Seen later as the P-wave on the ECG) which forces blood to enter the ventricles. The ventricles full of blood, after the contraction of the atrium, have higher pressure that the atrial chambers, thus the atrioventricular valves close.

Isovolumetric Contraction: The first heart sound is heard and marks the end of diastole and the beginning of systole. The ventricles depolarization (represented by the QRS complex in the ECG) is halfway, the ventricles then contract building pressure rapidly, but no blood is pumped in this phase at the exact small interval of time, thus its title.

Rapid Ejection: Once the ventricles pressure exceeds the atrial pressure, the valves inside the aortic and pulmonic blood vessels open and blood is ejected out of the ventricles.

Reduced Ejection: As the ventricles repolarize (reflected by the T-wave on the ECG) ventricular pressure starts to fall and the force of ejection is reduced. Once the pressure inside the ventricles drops below the aortic/pulmonic pressure the aortic valves close marking the end of systole and the beginning of diastole once again. The closure of these valves produces the second heart sound.

Isovolumetric relaxation: Diastole is then initiated; its first part is isovolumetric as the ventricles relax with the valves closed. The ventricular pressure drops whilst atrial pressure rises slowly as its being filled with blood.

Ventricular filling: Once the ventricular pressure is below that of the atrium the ventricles start filling with blood after the opening of the AV valves. In a dedicated interval of time, the ventricles are being filled passively. The atria contracts marking the end of the cycle and the beginning of the other cycle.

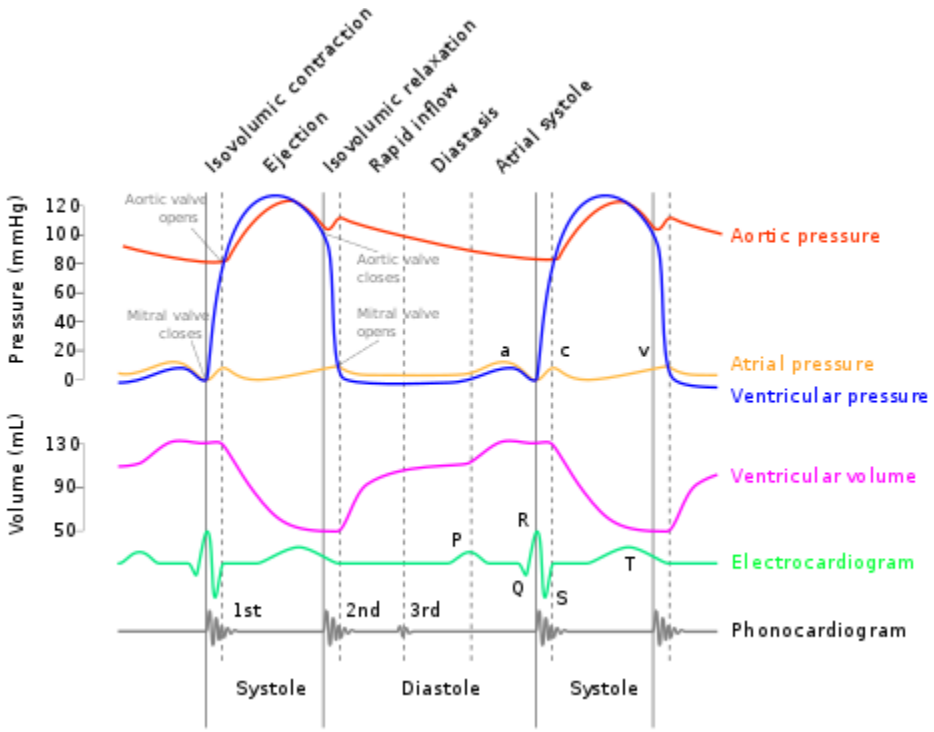


Figure 4 Detailed Cardiac Cycle

I.7 Cardiac action potential:

The heart is a muscle that contracts and pumps blood. It consists of specialized muscle cells called cardiac myocytes. The contraction of these cells is initiated by the electrical impulses named action potentials. Normal skeletal muscles need to be stimulated by the nervous system; the heart generates its own electrical stimulation. Pacemaker cells which do not contract but initiate the cardiac conduction system. The primary pacemaker is the SA node, it initiates all heartbeats and controls heart rate. Its main function is to fire spontaneously generating action potential spreading through the contractile myocytes of the atria. This action potential then arrives in the ventricular myocytes. The conduction of the action potential is essential for all myocytes to act in synchrony. The electrical system basic properties are the depolarization/polarization of the pacemaker cells & myocytes cells. This causes the voltage across the cells to change and thus allowing us to follow the steps of the cardiac cycle.

Pacemaker cells action potential steps:

- In blue: Pacemaker potential phase.
- In red: Rising
- In green: Falling.

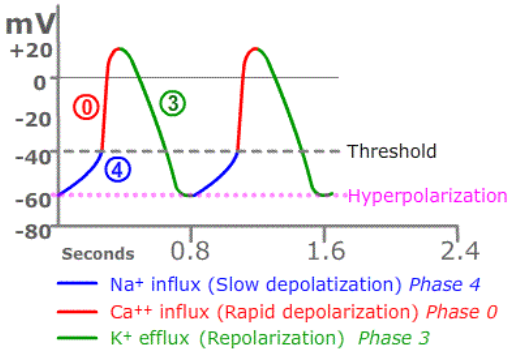


Figure 5 Pacemaker action potential

Myocytes cells action potential steps:

- In green at first & red at the end: Resting
- In green: Depolarization
- In Yellow: early repolarization
- In blue: Plateau
- In orange: Repolarization

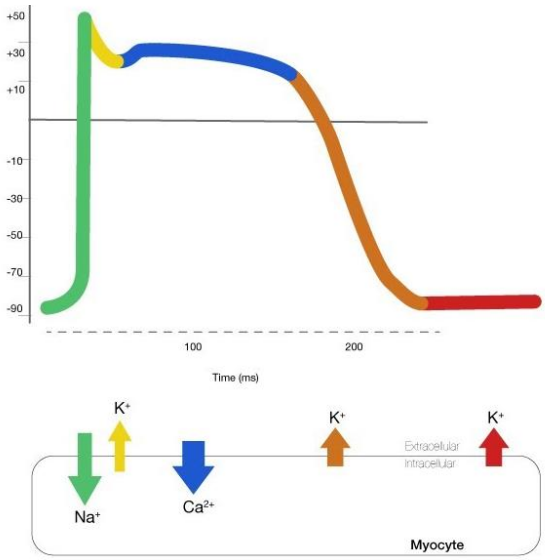


Figure 6Myocytes action potential

I.8 Cardiac conduction system & ECG :

The cardiac conduction system consists of the following components:

- Sino atrial node or SA node;
- Atrioventricular node or AV Node;
- Atrioventricular bundle
- Right & Left bundle branches

These nodes and branches can be recorded by an electrocardiogram or ECG/EKG.

ECG outputs a signal wave detailed in waves & segments, which corresponds each to a certain event of the cardiac electrical cycle.

- Once the atria are full of blood, the SA node fires a signal and the atrium is depolarized: **P-wave.**
- **The P-Q segment** indicates the conduction from the SA node to the AV node, the Atrium is contracted.
- **The QRS complex** indicates the firing of the AV node and the ventricular depolarization. The **Q-wave** implies the depolarization of the interventricular septum. The **R-wave** implies the depolarization of the main mass of the ventricles. And, the **S-wave** implying the depolarization of the ventricles at the base of the heart. Atria repolarization occurs during this time as well but the signal is obscured by the QRS complex.
- **The S-T segment** indicates the plateau phase in the myocardium action potential, marking the contraction of the ventricles and blood pumping.
- **T-wave** indicates ventricular repolarization before ventricular relaxation or diastole.

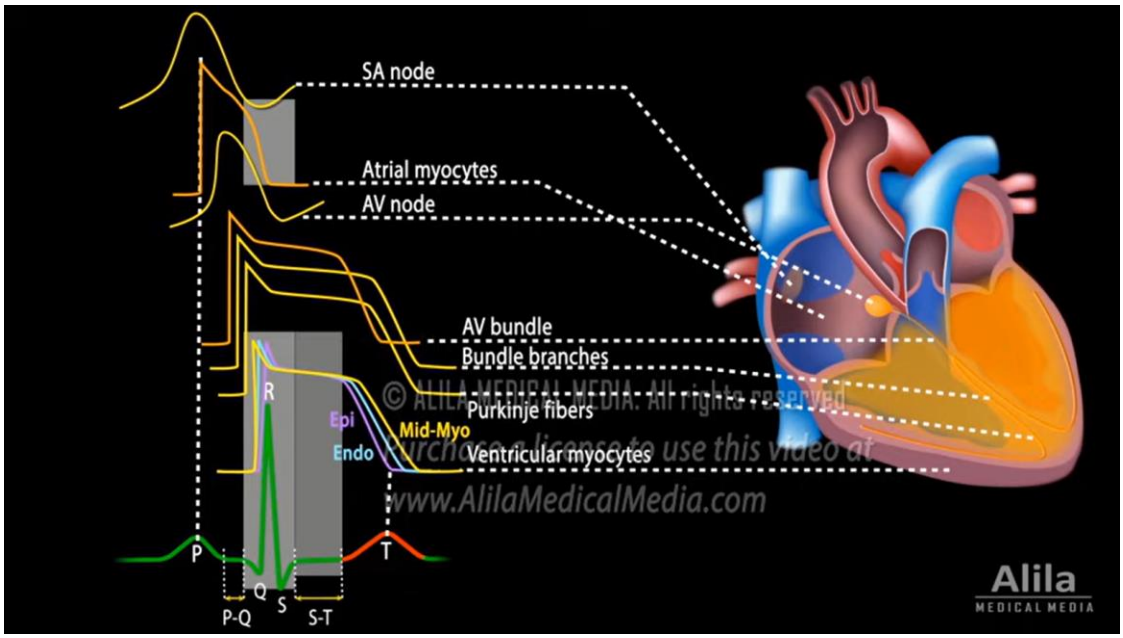


Figure 7 Origin of the heartbeat

I.9 12-Lead ECG:

Electrical activities of the heart can be picked up on the skin via electrodes. An ECG machine records these activities and displays them graphically. The 12-Lead ECG places 10 electrodes on the body of the patient to record multiple ECG signals from a variety of points of view.

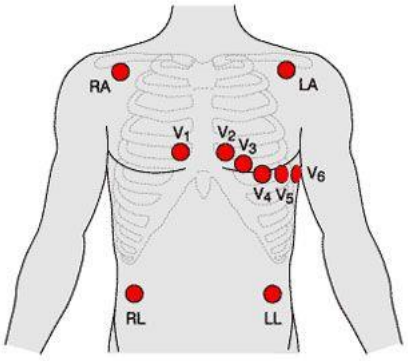


Figure 8 Placements of the 12 leads

The electrodes as shown in the picture are placed on each limb & the six others are strategically placed on the chest. The 4 limb leads help us observe an ECG in the vertical plane, the right leg is considered an earth electrode.

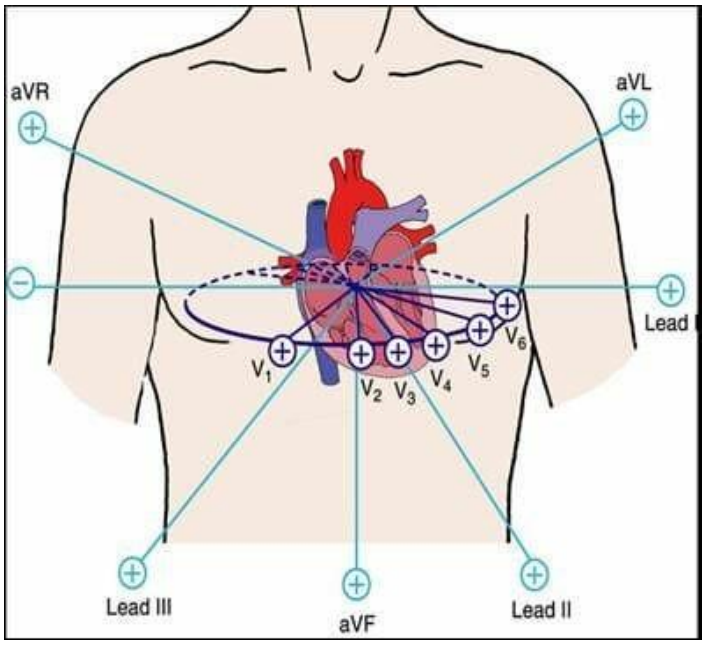


Figure 9 Point of view of each lead

The I, II and III leads are bipolar and are the ECGs between two limbs, one of them positive and the other is negative. The AvR, AvL and AvF are unipolar, they are measured by placing 1 limb node as positive and the average of the two other limbs as the negative.

The 6 chest leads indicate a horizontal plane study, they are also unipolar. The negative pole is the average from all the electrodes inputs from the limb leads.

I.10 How to read an ECG/EKG:

Among the 12 leads of the ECG, Lead II is the most popular. That's because the conduction of the heart pulses is directed towards lead II marking the best general view. In fact, a good analyst of the ECG needs to understand the variety of the different leads, we will focus on explaining through Lead II as it simplifies the explanation.

Heart rate: Identifying the QRS complex and calculating the heart rate using the general formula: $Heart\ Rate = \frac{1500}{Interval\ between\ two\ QRS}$. Although abnormal ECG signals need other formulas, the heart rate should be between 60 bpm and 100 bpm.

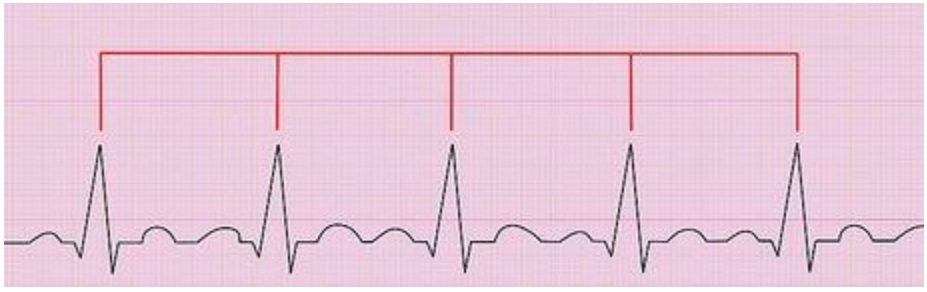


Figure 10 Heart rate

Heart rhythm: Intervals between QRS complexes need to be stable as its variation may be a cause of abnormal heart rhythm. The variation of once its large enough can be considered irregular and need treatment.

P-Wave: It marks the depolarization of the atria. Normal Presence of a P-wave indicates Sinus/regular rhythm. The irregularities that could occur in the P-waves are, for example, atrial enlargement. Since both left and right atrial have a signal that sums up to the P-signal, an enlargement of one of them will cause a disfiguration in the P-wave. If the P-wave is inversed, this could mean a non-sinus rhythm.

PR-interval: Responsible of indicating if the rhythm is started within the atria or in the AV-node. This interval indicates therefore the conduction through the AV node. A longer than normal PR interval signifies an AV Block. Consistent AV blocks are risky and called a first-degree heart block. If the PR interval is increasing and then followed by and absence of P-wave, it is a second-degree AV Block type 1. A shorter than usual PR interval signifies an early depolarization of the ventricles.

QRS Complex: Represents the depolarization of the ventricles. A normal QRS complex is narrow (70-100ms) Wider QRS complex indicates slow ventricular depolarization. This may cause problems to the patient.

S-T Segment: At the end of the S-Wave to the beginning of the T-wave is the S-T segment. A normal S-T segment is mostly flat. An elevation or depression of this segment could occur indicating heart problems such as myocardial infraction.

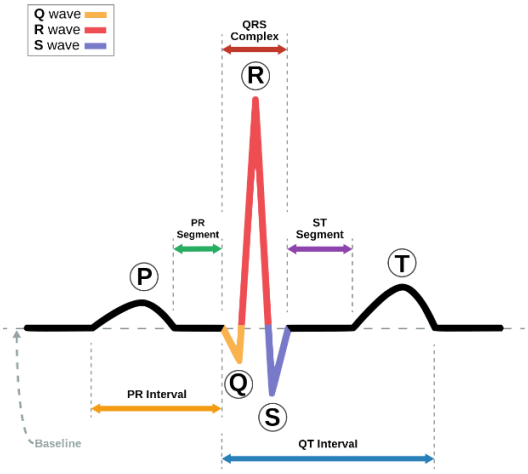


Figure 11 QRS complex signal

I.11 Cardiac Arrhythmias:

Classified by site of origin, cardiac arrhythmias are the abnormalities of the electrical signals in various regions of the heart.

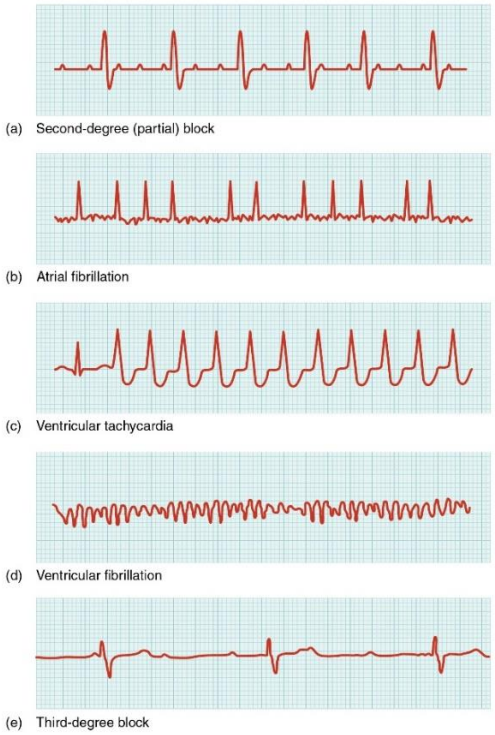


Figure 12 Cardiac Arrhythmias

Sinus Rhythm (located in the SA node: Normal sinus rhythm is between 60 and 100 firing every minute, resulting in a normal heart rate. An abnormal sinus rhythm could be the result of Sinus Bradycardia (<60 SA node firing every minute) which is normal during sleep. It could be also the result of a Sinus Tachycardia (>100 SA node firing every minute) which is normal during physical exercise.

Atrial Rhythms that are abnormal are generally resulting in super ventricular tachycardia.

Ventricular Rhythms are the most dangerous. Ventricular Tachycardia are the most famous arrhythmias located in the ventricles. It is most commonly caused by a single strong firing site or circuit in one of the ventricles. Usually occurs from a scarring, from previous heart attack, heart problems or surgery. Impulses starting in the ventricles produce ventricular premature beats that are regular and fast ranging from 100 to 250 bpm.

There are two types of VTs that could alternate and change the rhythms of the heart's electrical signal significantly:

Monomorphic VT is the one where the abnormality in the heart signal is periodic and changes the form of the QRS complexes. Patients who suffer from monomorphic VTs have symptoms that endangers their lifestyle and could in some sever cases be deadly.

Polymorphic VT is the type of VT where the signal has abnormal QRS complexes in an aperiodic and a more randomized abnormal signal.

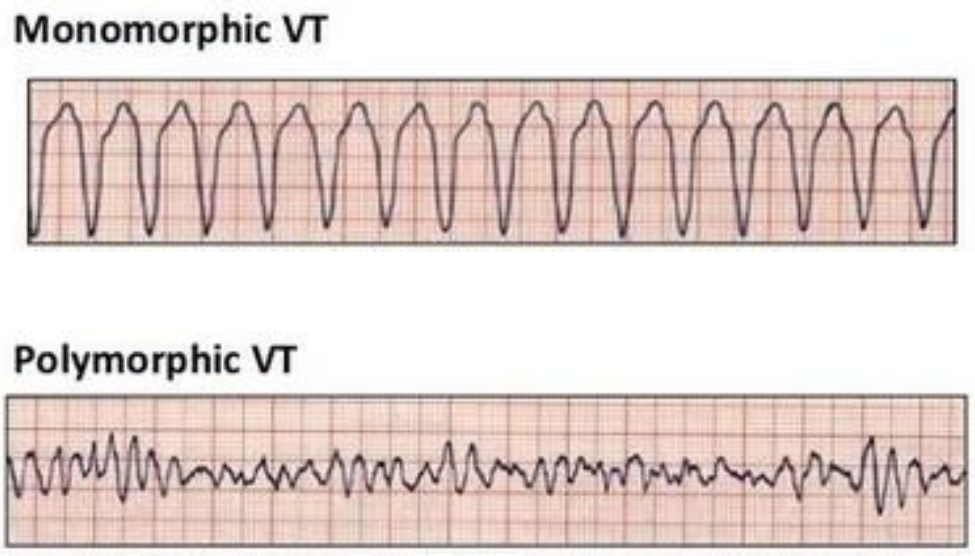


Figure 13 Difference: Monomorphic and polymorphic VT

On an ECG, V-tach is characterized by wide bizarre looking QRS complexes and an absent P-Wave. V-tach could occur in short episodes of less than 30 seconds and cause no to few symptoms. More than 30 seconds requires immediate treatment to prevent cardiac arrest. V-tach could also progress into Ventricular fibrillation.

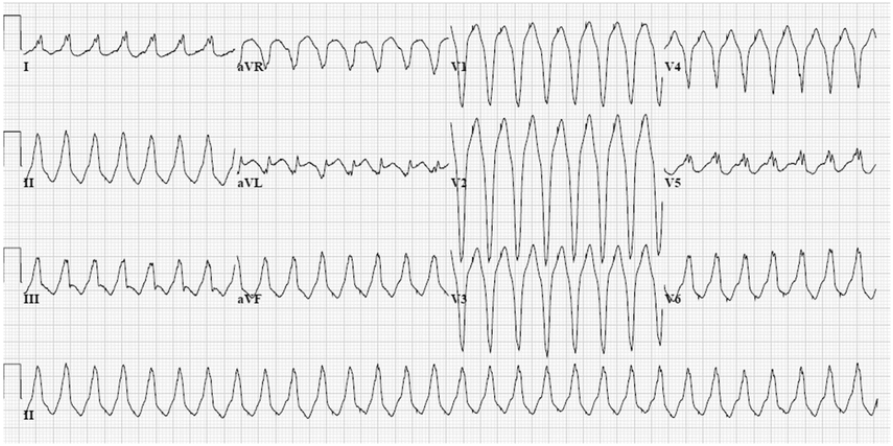


Figure 14 Monomorphic VT

I.12 Premature Ventricle Contraction or PVC

Originating from the lower chambers of the heart. PVCs can be caused from consumption of drugs & alcohol, medications, excessive exercise or damaged cardiac tissue. PVCs are very common but often produce no to few symptoms. When present, symptoms may include skipped heartbeat, lightheadedness and/or more. In normal conduction, action potentials start in the SA node depolarizes the atria and passes through the AV node to activate the ventricles. PVCs occur when the ventricles are activated prematurely caused by an abnormal firing site. Because ventricular depolarization does not come from the atria, PVC complexes are not preceded by P-Waves.

The signal of the PVC is conducted therefore in the ventricular myocytes. It Propagates slowly, causing a broader QRS complex. Depending on the location of the premature action potential, the QRS complex may be taller or deeper than usual.

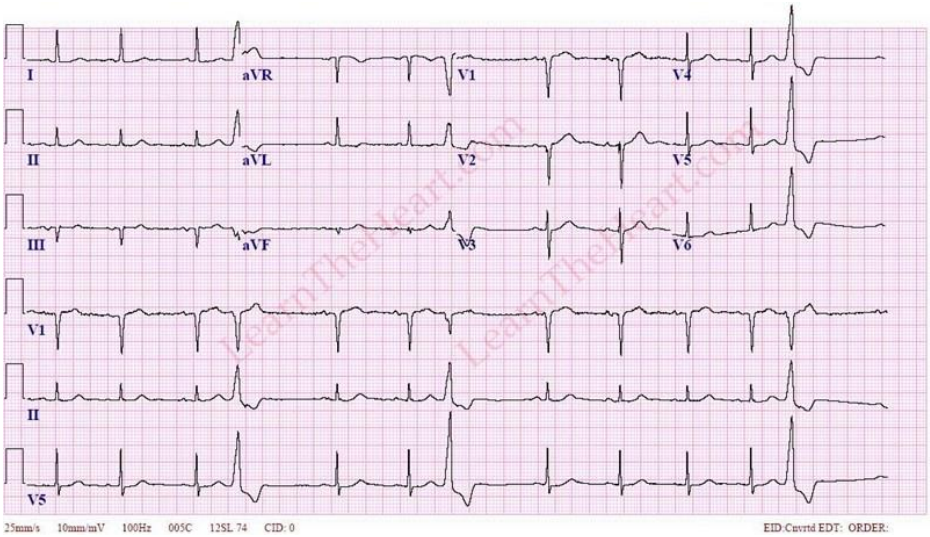


Figure 15 - 12 lead ECG of a PVC

PVCs typically do not conduct back to the atria which implies that the SA node firing is usually not affected. Nevertheless, P-wave that follows a PVC may not result in a beat if it happens too closely to the PVC. There are three mechanisms by which a PVC may occur:

- Increased atomicity: Ventricle myocytes are responsible sometimes of premature generation of action potentials, generally associated with calcium overload from diverse sources such as medicaments.
- Re-entry circuit: (Caused usually by a damaged heart muscle or a scar) This causes slow conduction of the signals followed by electrical signals that avoid crossing the scar resulting in possible loops.
- Triggered beats: Early or Delay after depolarization of the ventricles.

I.13 Cardiac ablation:

By using cold or heat energy to create tiny scars in the heart, the irregular electrical signal in the heart is stopped. This operation is done in order to correct heart rhythm problems. But since tiny scars are being applied to the heart muscle, PVCs could occur. The procedure is done via an IV inserted in the forearm. The catheter or sometimes multiple catheters inserted in the body. Contrast is then injected in the catheters to deliver an extreme cold (cryoablation) or heat (radiofrequency energy) resulting in a strategic tiny scar. In the end, as depicted previously, cardiac ablation could enhance and improve the quality of life of the patient but also could result in the return of irregular heartbeats.

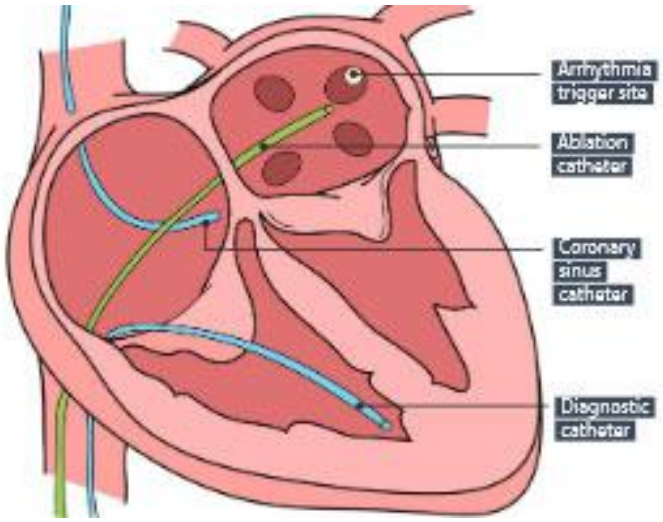


Figure 16 Cardiac ablation

CHAPTER: II. CLASSIFICATION OF QRS COMPLEXES

II.1 Introduction

Just like any scientific theory, for example neglecting the frictions in physics, for the purpose of simplifying a scientific activity to understand more precisely a phenomenon. In this particular case we won't neglect anything materialistic, but we will try to simplify the problem in order for us to have an easier start. The classification of QRS complexes was done originally by medical doctors and ECG spcialists, but since it is done by humans, human error and inaccuracy is inevitable. Now, it is precise and concise but using algorithms and machine learning techniques could show a better result in the accuracy of the classification. In regards, of the simplification, we will be focusing on two particular heart arrrythmias, Ventricular Tachycardia and PVCs, these types of abnormalities in the heart muscle, in the ventricles chambers more specifically, will be studied and analysed in order to achieve greater classification. As shown in figure below, there are two types of ventricular chambers and regions, where the abnormality could occur, thus the second simplification prevails itself, the classification will be binary.

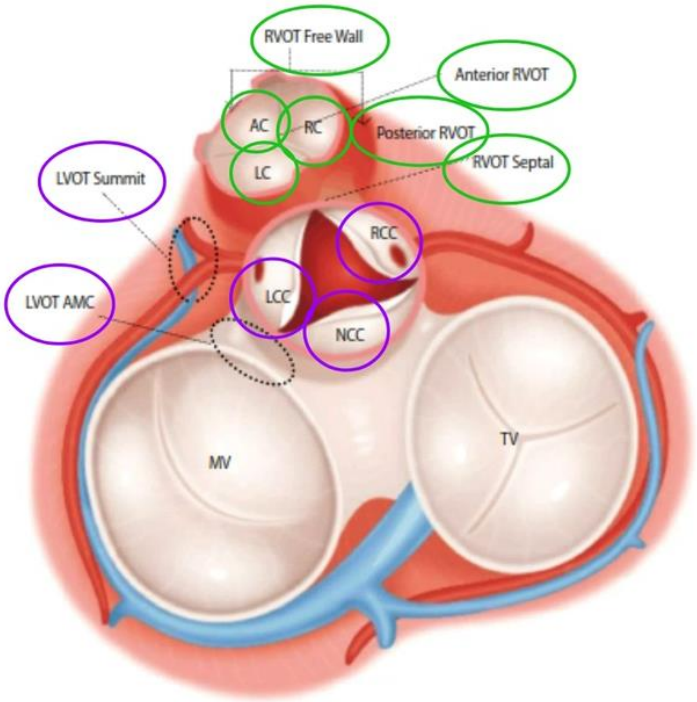


Figure 17 Binary decomposition of Ventricular regions

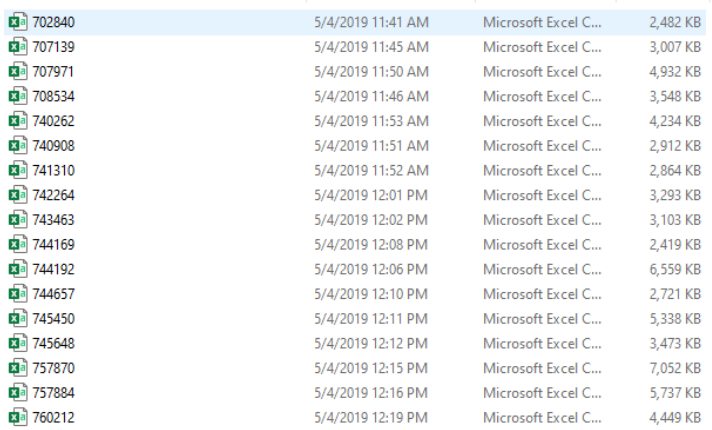
In green, the right side of all the regions of the right ventricles, in purple the left side. Having a binary classification: either left or right is enough, even if it is not precise, if a cardiac ablation should be done to prevent the VTs or the PVCs, the medical operator should know the region in which the scar or damaged tissue in the heart is located before doing the operation.

Normally, the doctors analyze the ECGs of the patients but rarely end up knowing accurately the source of the problem in the ventricles, so they do the operation looking for the scar and give up a very significant element in all medical operations: time. Thus, trying to help these doctors achieve cardiac ablation in a faster manner is the general goal of this project.

What is the method that is used to know if it is a left or right source of abnormality, we can ask ourselves? The 12-lead ECG is a 3D view of the electricity in heart. Simply, it is the lead that has an abnormality that shows in which direction we should look to see the source of the signal. With that logic, there are many other components that lead us to understand the steps taken to know the source of the scar. The most important one in the horizontal leads is the predominant transition of the signals in the V1 to V6 leads. For example, a transition zone in V1 or V2 may indicate a right ventricular origin, while a transition zone in V5 or V6 may suggest a left ventricular origin. To add, in any other lead if we have an axis deviation we can conclude the origin of the problem. A positive axis deviation (greater than +90 degrees) may indicate a left ventricular origin, while a negative axis deviation (less than -30 degrees) may suggest a right ventricular origin.

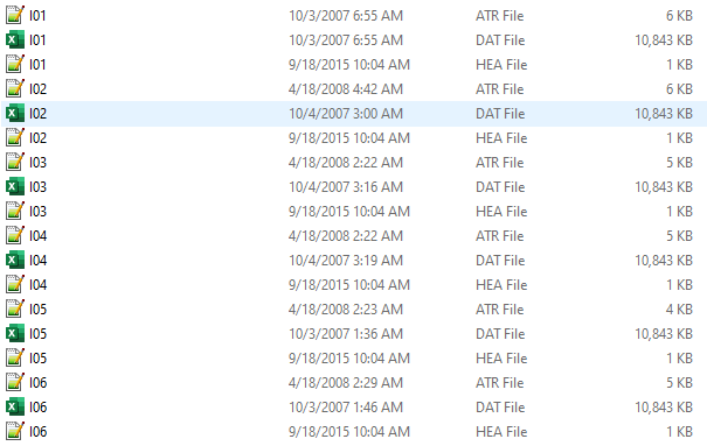
II.2 Data

Before explaining the data set used, we should remember that even that there are two types of VTs, monomorphic and polymorphic. For that, two data sets were chosen, the first originating from 334 patients suffering from monomorphic VTs and PVCs. The second data set is the one contains signals describing the ECGs of patients suffering from polymorphic VTs, they are 32 patients having a total of 75 annotations since some patients have more than one annotation, each annotation is a 30 mins signal recording with a sampling frequency of 257 Hz. Both data sets are 12-lead ECGs.



702840	5/4/2019 11:41 AM	Microsoft Excel C...	2,482 KB
707139	5/4/2019 11:45 AM	Microsoft Excel C...	3,007 KB
707971	5/4/2019 11:50 AM	Microsoft Excel C...	4,932 KB
708534	5/4/2019 11:46 AM	Microsoft Excel C...	3,548 KB
740262	5/4/2019 11:53 AM	Microsoft Excel C...	4,234 KB
740908	5/4/2019 11:51 AM	Microsoft Excel C...	2,912 KB
741310	5/4/2019 11:52 AM	Microsoft Excel C...	2,864 KB
742264	5/4/2019 12:01 PM	Microsoft Excel C...	3,293 KB
743463	5/4/2019 12:02 PM	Microsoft Excel C...	3,103 KB
744169	5/4/2019 12:08 PM	Microsoft Excel C...	2,419 KB
744192	5/4/2019 12:06 PM	Microsoft Excel C...	6,559 KB
744657	5/4/2019 12:10 PM	Microsoft Excel C...	2,721 KB
745450	5/4/2019 12:11 PM	Microsoft Excel C...	5,338 KB
745648	5/4/2019 12:12 PM	Microsoft Excel C...	3,473 KB
757870	5/4/2019 12:15 PM	Microsoft Excel C...	7,052 KB
757884	5/4/2019 12:16 PM	Microsoft Excel C...	5,737 KB
760212	5/4/2019 12:19 PM	Microsoft Excel C...	4,449 KB

Figure 18 Data set of monomorphic VT



I01	10/3/2007 6:55 AM	ATR File	6 KB
I01	10/3/2007 6:55 AM	DAT File	10,843 KB
I01	9/18/2015 10:04 AM	HEA File	1 KB
I02	4/18/2008 4:42 AM	ATR File	6 KB
I02	10/4/2007 3:00 AM	DAT File	10,843 KB
I02	9/18/2015 10:04 AM	HEA File	1 KB
I03	4/18/2008 2:22 AM	ATR File	5 KB
I03	10/4/2007 3:16 AM	DAT File	10,843 KB
I03	9/18/2015 10:04 AM	HEA File	1 KB
I04	4/18/2008 2:22 AM	ATR File	5 KB
I04	10/4/2007 3:19 AM	DAT File	10,843 KB
I04	9/18/2015 10:04 AM	HEA File	1 KB
I05	4/18/2008 2:23 AM	ATR File	4 KB
I05	10/3/2007 1:36 AM	DAT File	10,843 KB
I05	9/18/2015 10:04 AM	HEA File	1 KB
I06	4/18/2008 2:29 AM	ATR File	5 KB
I06	10/3/2007 1:46 AM	DAT File	10,843 KB
I06	9/18/2015 10:04 AM	HEA File	1 KB

Figure 19 Data set of Polymorphic VT

The monomorphic data set is a total of 334 csv files that have similar composition: a table of 12 columns having the values per sample in microvolts of the signals. Whereas for the polymorphic data set, the same type of tables is inside a DAT file labeled and described inside ATR and HEA files. Reading the tables is encrypted and encoded, using notepad++ to the see the values inside these tables was not successful since the values are not in the decimal format but were hexadecimal.

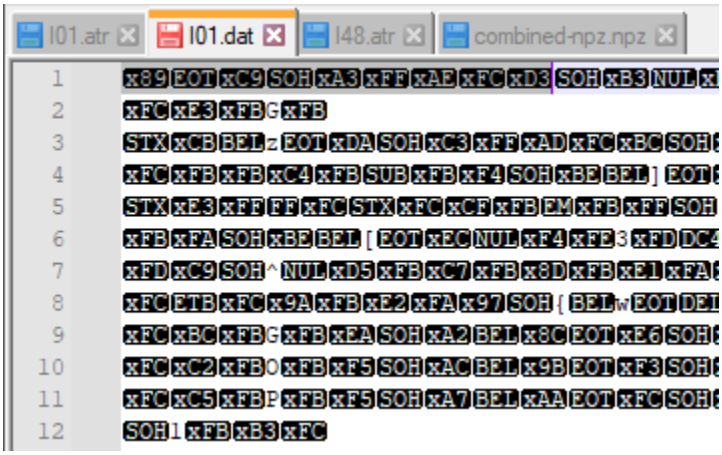


Figure 20 Encoded Polymorphic data

II.3 Visualizing and analyzing the signals

We will focus on the polymorphic data set for the next parts. But the visualization of the monomorphic 12-lead ECGs was done using the matplotlib library in python. Following, the Spyder IDE was used not only for this task but for the all of the project. To initiate the IDE, the use of an environment for specific library download was used and starting the Spyder IDE was made through the Anaconda Prompt. The environment was called tf.

```
(base) C:\Users\USER>conda activate tf

(tf) C:\Users\USER>python -m spyder.app.start
```

Figure 21 Prompt to activate the right environment (tf) in Spyder IDE

Since the polymorphic data set was not readable at first, it was visualized using the WFDB library in python which is the waveform-database library.

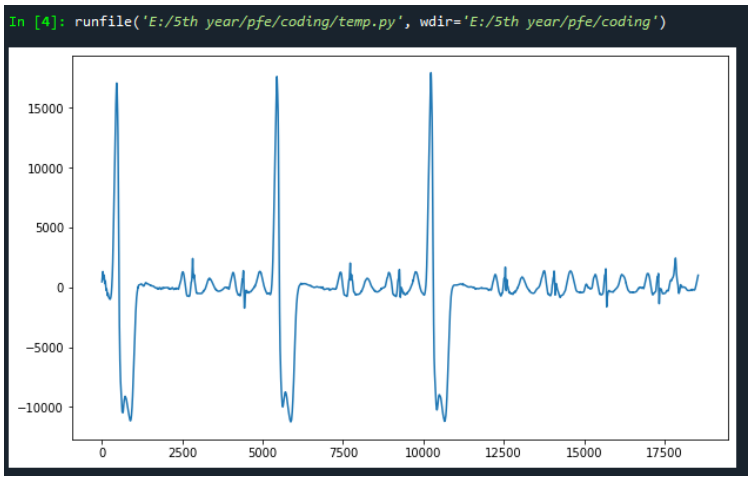


Figure 22 Lead I of the patient 1067472, vertical axis: microvolts, horizontal axis: samples.

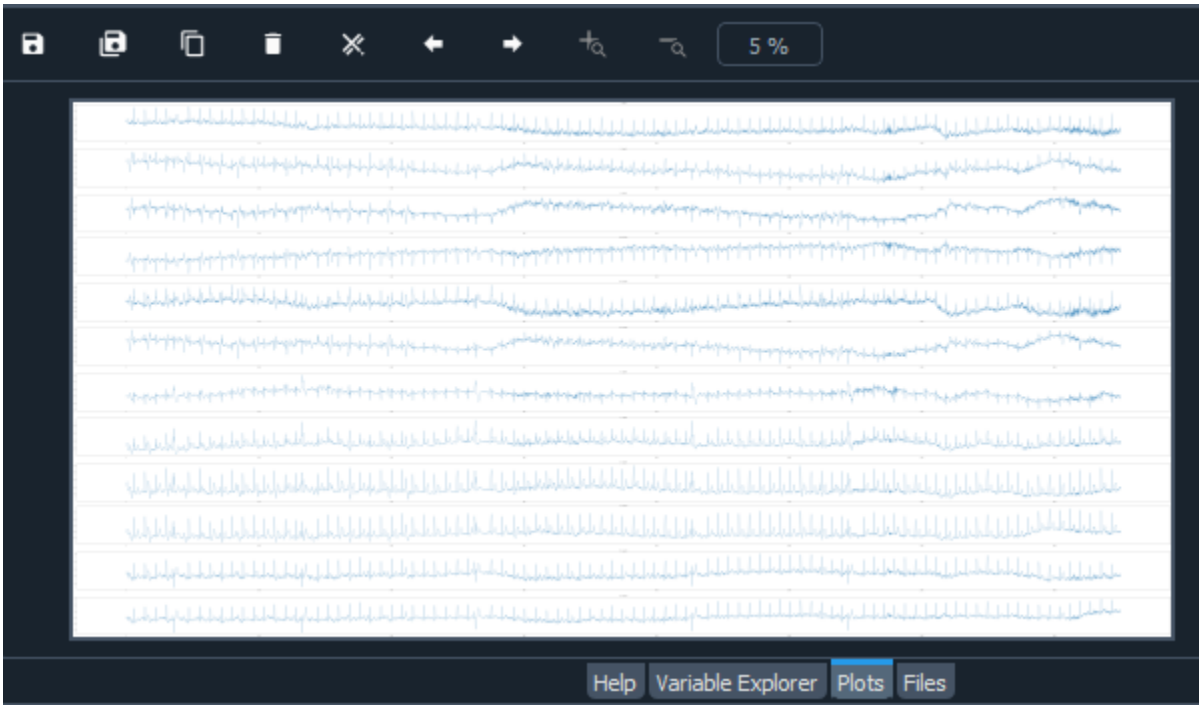


Figure 23 The first 15000 sample of the 12-lead ECG of the first annotation.

Continuing through the polymorphic data base, since it is more precise to cut the signals into an exact number of samples, the signals were cut into 15000 sample slices. The exact number of

minutes for any annotation is 30 minutes which is equivalent to 1800 seconds. The frequency is 257 Hz, we can deduce the total number of samples for each annotation:

$$N_{total} = 257 * 1800 = 462{,}600\ sample.$$

Since the slice of 15000 sample was chosen, what is that in minutes or in seconds:

$$T_{slice} = \frac{1800 * N_{oneslice}}{N_{total}} = \frac{1800 * 15000}{462600} = 58.3657\ seconds \approx 1\ minute$$

II.4 Cleaning and choosing

We aim to have a more simplified data set as we mentioned previously, the third simplification is to choose the signals or the slices that contain a certain abnormality. The annotation contains 30 minutes of a recording and the VT is not shown throughout the total recording, some annotations show low to none VT signal descriptions. With that being said, choosing the right slices for the purpose of achieving better results later was done.

Annotation	n of good slices
I01	10
I02	9
I03	0
I04	6
I05	3
I06	0
I07	0
I08	15
I09	2
I10	0
I11	0
I12	25
I13	0
I14	0
I15	4
I16	0
I17	5
I18	10
I19	0
I20	11
I21	4
I22	5
I23	0
I24	0

I25	0
I26	7
I27	0
I28	0
I29	14
I30	18
I31	8
I32	0
I33	0
I34	0
I35	15
I36	17
I37	0
I38	23
I39	4
I40	8
I41	0
I42	13
I43	0
I44	0
I45	0
I46	13
I47	13
I48	18
I49	0
I50	13
I51	5
I52	0
I53	9
I54	11
I55	2
I56	0
I57	30
I58	11
I59	11
I60	3
I61	0
I62	1
I63	0
I64	0
I65	3

I66	0
I67	19
I68	2
I69	1
I70	1
I71	0
I72	0
I73	0
I74	4
I75	0

Total good slices	406

Total slices	2250

Percentage	18.04444444 %

Figure 24 Table showing the selection of all of the annotations

The process of choosing the right slice to use was done on two phases, first the slices were plotted in a PNG format to obtain a total of 2250 slice in average assuming that every annotation is 30 slices. From the explanation of the polymorphic VT, is does not take an expert to know if it is really a polymorphic description of VT.

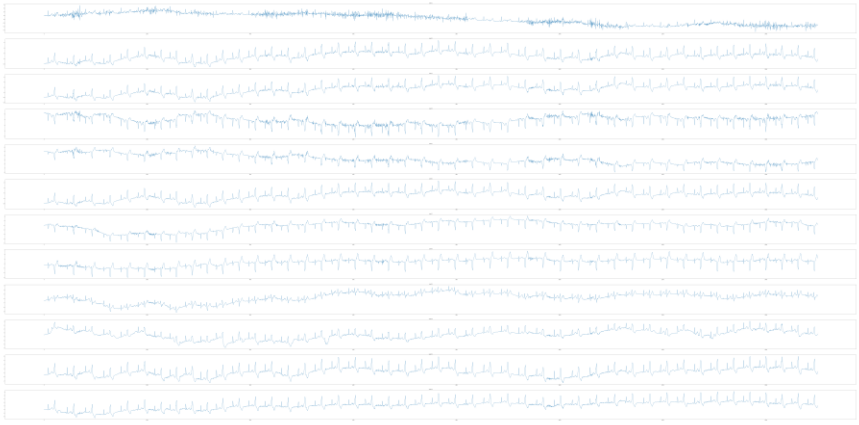


Figure 25 Annotation I53

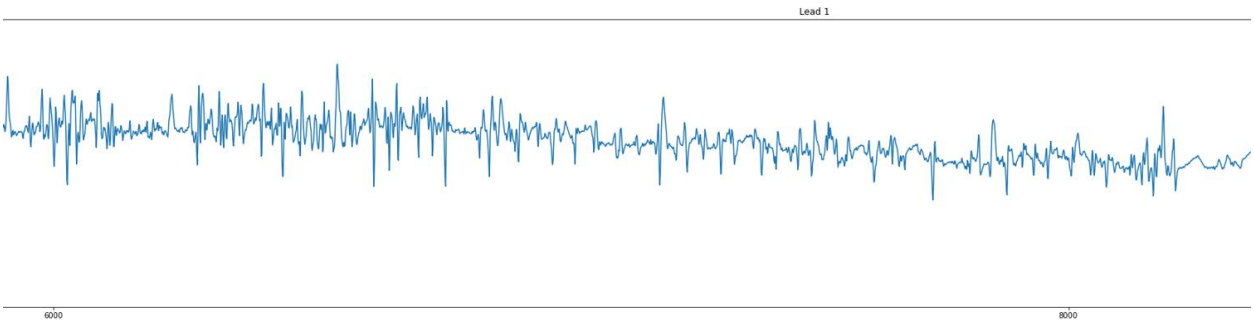


Figure 26 Zooming into the annotation I53 (polymorphic VT)

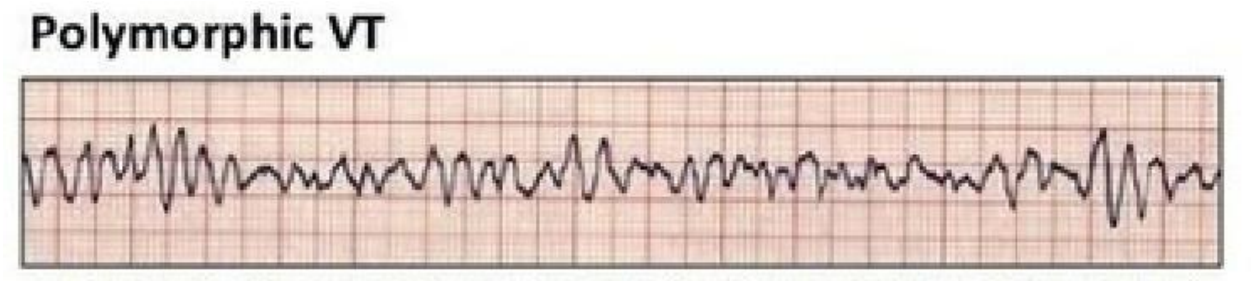


Figure 27 Polymorphic VT

Figure 28 Example-1 of a non-polymorphic data slice.

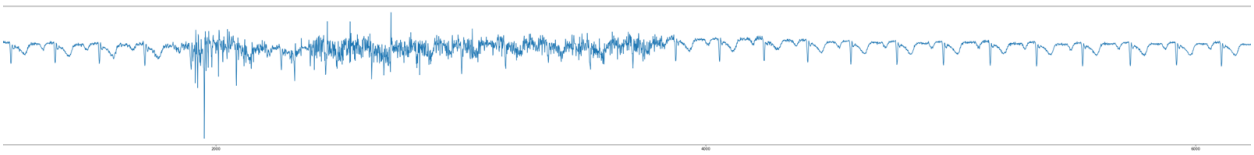


Figure 29 Example-2 of a non-polymorphic data slice.

In other words, we choose the data slices that contain in the totality of the 15000 sample, a polymorphic VT description and discard the other slices.

The second phase of the selection was made from the already selected 500 slices, to remove slices that were not purely usable or had any doubts like not having complete signals, or having an abnormality different from the polymorphic VT. We end up at the end with 406 slice that are somewhat perfect to use later on.

II.5 Pre-processing and preparation

Before starting with the main algorithm, the data should be pre-processed for it to be smoother and having less artifacts. These noises and artifacts could change the results drastically since we are doing signal processing techniques. Since we have 406 slice of data that are usable, we aim to pre-process them all. Knowing that if we work with the original data, we will have problems regarding the speed and the accuracy of the procedure. To solve that, the 406 slices that were chosen had two indications that profiled them, the annotation number and the sample range. Using these two values we can find the exact slices at any point of time. Using this, the data was converted to NPZ files.

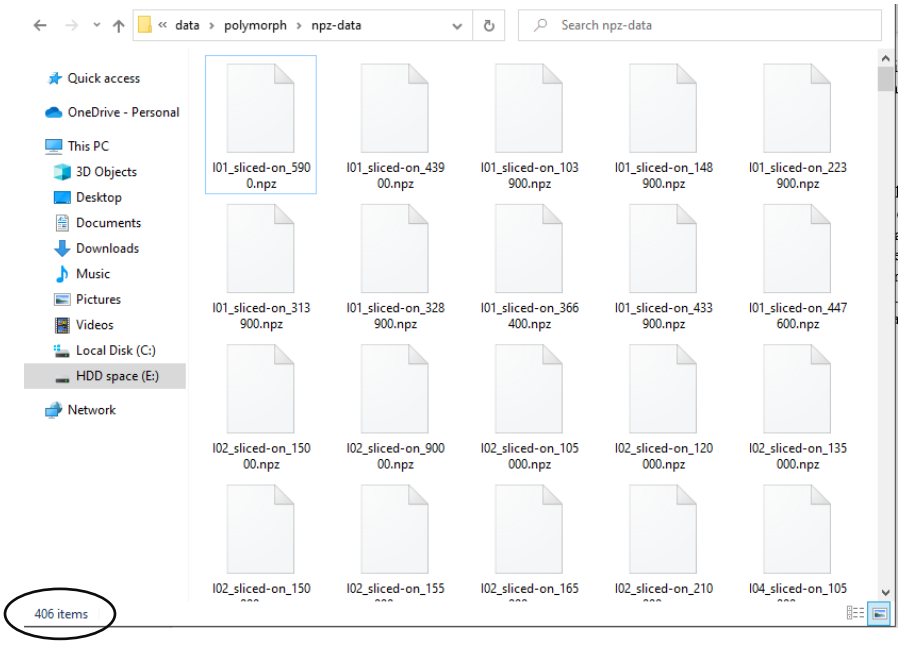


Figure 30 Chosen Slices

These files were all combined into 1 NPZ file called combined-npz. Now we can apply the preprocessing step which requires applying a smoothing filter and removing artifacts and noises and then normalizing the signal. To do that, the choice of the filter was not random, but strategic.

Savitzky–Golay filter:

The most used filter in biomedical engineering and more precisely in ECG processing activities is the Savitzky–Golay filter. This filter is used for the smoothing of the signal. It is a digital filter and it operates by fitting a polynomial function to a small window of data points and estimates the central point of the window based on the polynomial used and the window size. These two variables play a huge role in the precision of the smoothing. It is used in the processing study of ECGs in general because of its preservation of important features, which is essential to negate information loss. It is an accurate estimation since the smoothing is done on a very small window size. The algorithm step-by-step is as follows:

1. Select a window size (odd number) and polynomial degree.
2. Slide the window across the data points.
3. For each window position, fit a polynomial to the data points within the window using least squares regression.
4. Estimate the central point of the window based on the polynomial fit.
5. Repeat steps 3 and 4 for all data points.
6. The resulting smoothed data constitutes the output of the Savitzky-Golay filter.

Luckily, we are not focusing on the pre-processing techniques and using this filter in python is simplified by the use of the savgol_filter library inside the scipy library.

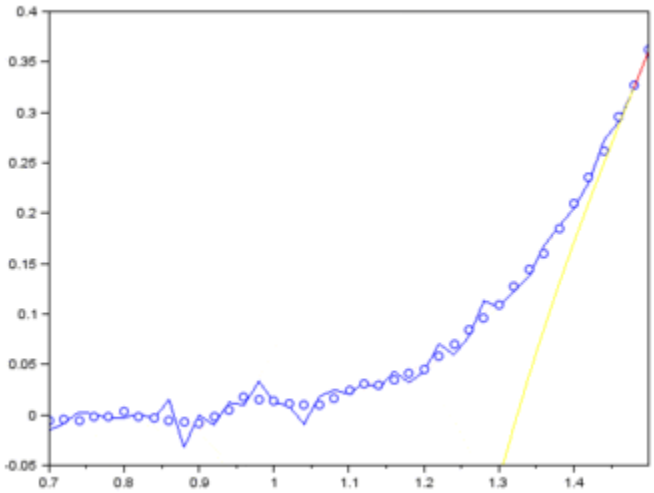


Figure 31 Savitzky–Golay filter

Normalizing the signal after that as preparation, we calculate the mean value and deduct this value from the original filtered signal.

II.6 CNN algorithm

The goal was to use deep learning techniques to classify the ECG data set that was preprocessed and ready for use. For that, the Convolutional Neural Networks algorithm was used based on its popularity and great resulting output in the field of machine learning.

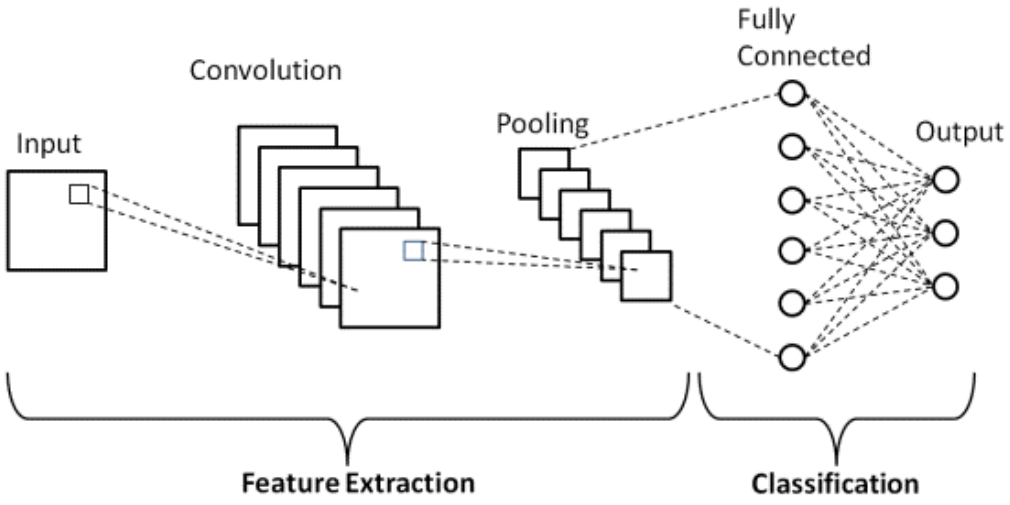


Figure 32 Convolutional Neural Networks

Convolutional Layers are the building blocks of the CNNs, they apply filters to the input data in a sliding window manner. Each filter is a small matrix of weights or kernel. The convolutional operation involves element-wise multiplication of the filter weights followed by a summation.

$$Output[i,j] = sum(Input[i',j'] * Filter[k])$$

This operation is performed for each location in the input data, resulting in a feature map that captures learned patterns.

Pooling Layers downsample the feature maps obtained from the previous layers. Many types of pooling could be used, either max pooling or average pooling. The type of pooling affects the values selected inside the output originating from the convolutional layers.

$$Max\ Pooling: Output[i,j] = \max(Input[i',j']),$$

$$Average\ Pooling: Output[i,j] = average(Input[i',j'])$$

$$where\ i',j'\ are\ appropriate\ indices\ within\ the\ pooling\ window$$

Fully connected layers are close mathematically to the convolution layer's job, but in the connected layers every neuron inputted to that layer is connected to every other neuron in previous layers.

$$Output\ Activation(W * Input + b)$$

W is the weight matrix, Input is the input vector, b is the bias vector and Activation is the activation function.

The activation function is used for the model to learn more complex patterns and non-linearity. There is Step, Sigmoid, ReLU, Tanh and SoftMax functions.

ReLU is an activation function that outputs the input directly if it's positive, and 0 otherwise.

$$ReLU(x) = max(0, x)$$

SoftMax is an activation function that is commonly used in the output layer for multi-class classification problems. The output becomes a set of probabilities, with each class probability summing up to 1. This logic enables it to add a non-linearity to the network and learn more complex relationships in the data.

$$softmax(x\_i) = exp(x\_i) / sum(exp(x\_j)), for\ all\ i\ in\ the\ output\ layer$$

Receptive field refers to the area of the input space that affects the output of a particular neuron. In CNNs, the receptive field grows larger as the network deepens.

Weights represent the parameters learned by the network during training. Each neuron in the network has associated weights that determine the contribution of its inputs to the output. The weights are adjusted during training using optimization algorithms such as backpropagation to minimize the loss function.

Adam optimizer is an optimization algorithm that adapts the learning rate based on the gradient of the weights during training.

$$m\_t = beta1 * m\_t - 1 + (1 - beta1) * g$$

$$v\_t = beta2 * v\_t - 1 + (1 - beta2) * g\text{^}2$$

$$m\_hat = m\_t / (1 - beta1\text{^}t)$$

$$v\_hat = v\_t / (1 - beta2\text{^}t)$$

$$W = W - learning\_rate * m\_hat / (sqrt(v\_hat) + epsilon)$$

M represents the first moment estimate (mean) of the gradients, v represents the second moment estimate (uncentered variance) of the gradients. beta1 and beta2 are the decay rates for the first and second moments, t is the iteration step, learning rate is the step size for weight updates and epsilon is a small constant to prevent division by zero.

Applying the CNN on the preprocessed data set was made possible with the efficient use of the TensorFlow library, after loading the preprocessed data, converting it to a NumPy array and reshaped to a proper input for the CNN algorithm, the input was passed through an architecture where the testing was made, consisting of two Conv1D layers with ReLU activation MaxPooling1D for downsampling, a flatten layer to convert the 2D feature maps into a 1D vector and 2 Dense Layers for activation, the final dense layer before the output uses SoftMax activation function for multiple (binary) classification.

$$Input$$

$$|$$

$$Conv1D\ (32\ filters, kernel = 3, ReLU)$$

$$|$$

$$MaxPooling1D\ (pool\_size = 2)$$

$$|$$

$$Conv1D\ (64\ filters, kernel = 3, ReLU)$$

$$|$$

$$MaxPooling1D\ (pool\_size = 2)$$

$$|$$

$$Flatten$$

$$|$$

$$Dense\ (128\ neurons, ReLU)$$

$$|$$

$$Dense\ (num\_classes, Softmax)$$

The architectural neural network was then compiled then trained with a validation split 0.2 to evaluate the model.

II.7 Results

```
Test Loss: 1.5799003839492798
Test Accuracy: 0.8167077302932739
```

Figure 33 Results of the CNN algorithm

At the end of the algorithm, receiving a small test loss as 1.6 which has no physical unit but is a value representing classification accuracy which is considered quite low. The test accuracy is 82%.

II.8 Conclusion & Interlude

Obtaining an 81% accuracy is quite high in a binary classification of a simple CNN algorithm for the ECG signals. But it is not 100%, and since we are working on a problem where very high accuracy is needed and very low none error should occur, a creation of an artificial intelligence capable of pulling of the task of classification without human assistance is a much more complicated study. Regarding this logic, the problem was simplified 3 times as well from choosing the slices that were usable to using not a random data-set, the accuracy of this product on a random patient will be much lower. Thus, working on our model and making it better is essential for a more intellectual study. For that, since we only used 18% of the data obtained from patients already having polymorphic VT, what if we could enlarge our data set and make it greater in size.

CHAPTER: III. SIMULATION OF ECG SIGNALS

III.1 The importance of simulation

The Simulation has many purposes, especially in trying to create a precise replica of a heart beat signal. Creatively, it has many advantages. In this study, we are trying to make a simulation that could benefit our study, it could be then used as a set of values for different models in the vision of making them more accurate. The simulation was made possible using different approaches. First, we could use the model already trained in the previous chapter to simulate a heart signal. Although the model is 81% the simulation algorithm is still viable for use when the accuracy of the model becomes higher. Second, knowing and analyzing polymorphic VTs we could deduce some components to make a mathematical model for that particular heartbeat. Finally, making a simulation of a QRS complex with precise descriptions and evaluations will be key to any simulation regarding ECG in future works, since we dive into the base of the most precise components of the QRS complex.

III.2 Simulation using the prediction of the model

Using the TensorFlow functions and managing the data set to create and predict a new table of values is one way to approach this study. The model is already trained and the creation of a new set of values is then done to obtain an automated simulation. Using the code Simulation-1st found in the ANNEXE we obtain the following plot.

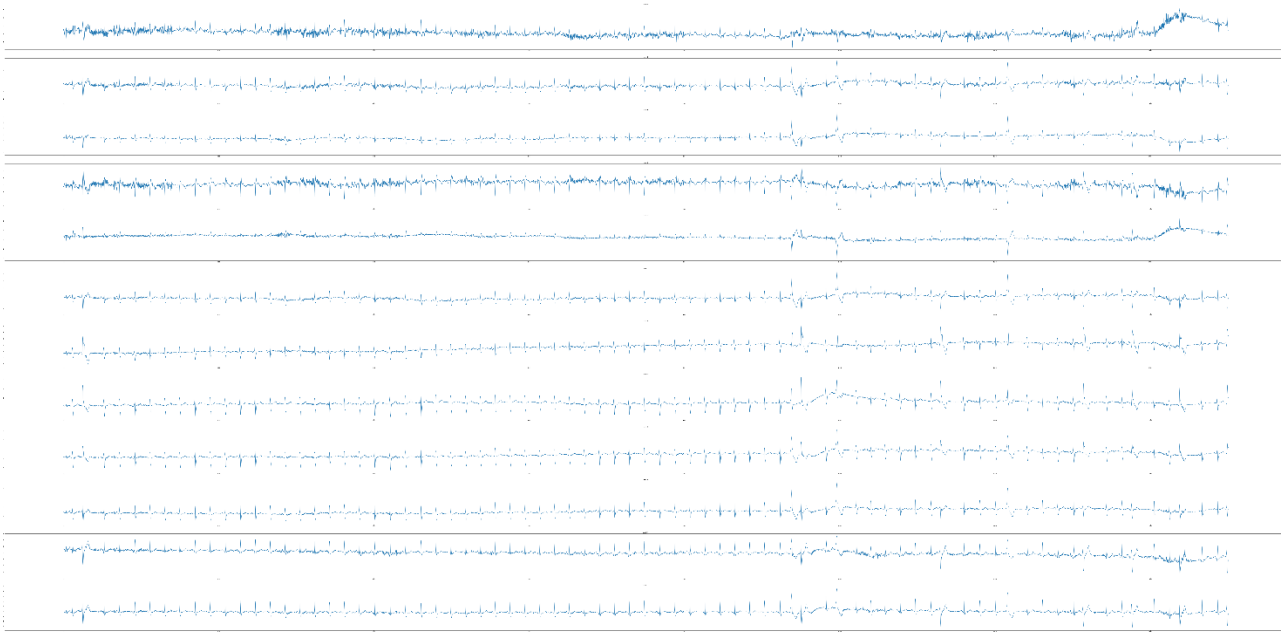


Figure 34 Simulation 1st

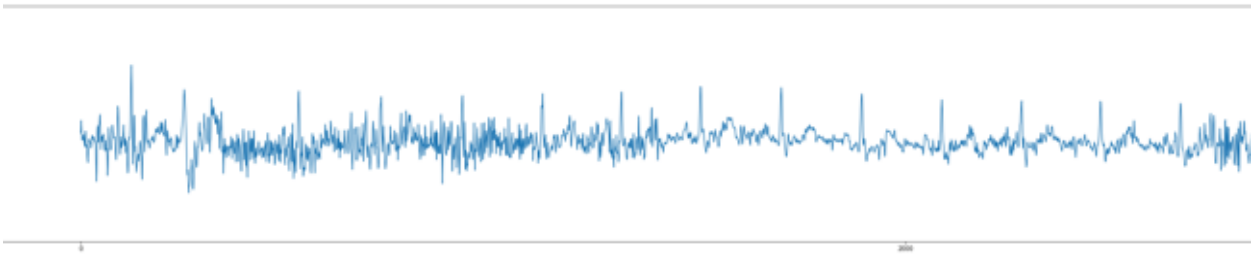


Figure 35 Zooming in to the first simulation

Zooming into the simulated 12-lead ECG, we could directly see that it resembles the other data sets in various ways although some other leads have abnormal signals. This shows great success as a rapid simulation.

III.3 Simulation using mathematical models

What if we directly simulated the polymorphic VT from scratch, this time it won't be a prediction, a previous data set won't be needed and the simulation would more randomized. With that being said, here are the parameters taken into consideration to give life to the model:

- Sampling frequency of the ECG signal
- N total number of heartbeats to generate the signal
- Amplitude with or without noise addition
- Mean heart rate in bpm (beats per minute)
- Standard deviation heart rate
- Ratio of low-frequency to high-frequency components in the heart rate variability.
- Internal sampling frequency
- List of angles in degrees representing the waveform characteristics (P, Q, R, S, T)
- Amplitude parameters for the waveform components
- Shape parameters for the waveform components

The polymorphic VTs are described also with alternating non-periodically of RR time series, which is the interval time between two QRS complexes, and the angles representing the waveform characteristics are also another description of the polymorphic VT phenomenon. The other components exist for the signal to be realistic.

Finally adding noise is essential, using Gaussian Noise Addition which is a simplified amplitude alteration to add a small noise to the signal.

Following all of these components the simulation is done by a randomized signal which is he 3rd and most important characteristic of polymorphic VTs as shown in previous examples. The function used to create the simulated signal called "ecgsyn" starts then by formulating a random signal, adding exponential components to manage the differences between different parts of the QRS complex of each heartbeat. Then Exponential functions are used for every wave inside the ECG signal (P, Q, R, S, T). The simple equation for that is:

$$amplitude * np.exp(-0.5 * ((t / ti) ** 2))$$

For the realism of the signal, we use Cubic interpolation to unsample the RR interval series. This algorithm consists of:

- Fitting cubic spline curve to the data
- Estimating the values at the intermediate positions

The results after this interpolation are having a continuous and realistic RR interval series.

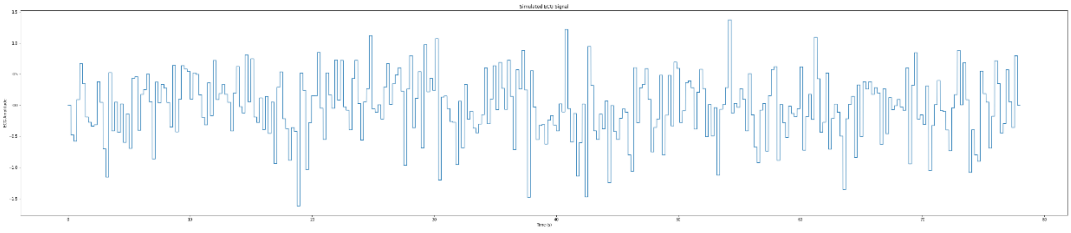


Figure 36 Result of the simulation of the polymorphic VT

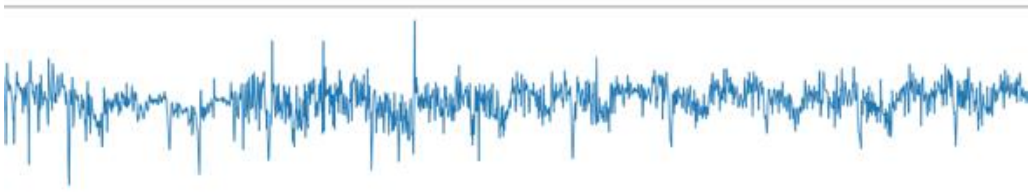


Figure 37 a real polymorphic VT

As we can see in the results the simulated ECG signal resembles the real polymorphic VT in this comparison. The main difference is the great amount of noise affecting the real patient's signal.

III.4 Simulation using a precise QRS complex editor

In this last part of the simulation, we aim to obtain the most precise and accurate QRS complex simulation. Contrarily to previous simulation, we want to simulate a normal heartbeat, not an ECG signal but a single QRS complex with a P and a T wave. For that we dive deeper into the problem and simulate each part of the ECG alone, then add the waves together to obtain the resulting simulation.

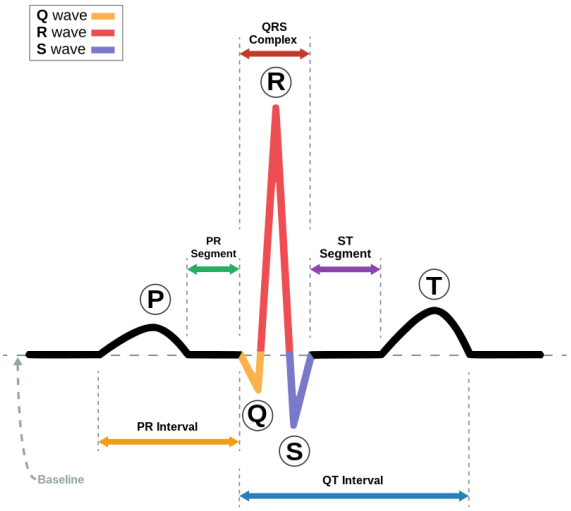


Figure 38 Simplified QRS complex

The equation used for the simulation of each wave with different parameters is:

$$\boldsymbol{a}.\boldsymbol{e}^{(\frac{-(\boldsymbol{x}-\boldsymbol{t})}{\boldsymbol{d}})^{\boldsymbol{2}}}.sin(\boldsymbol{2}.\boldsymbol{\pi}.\boldsymbol{f}.\boldsymbol{x}+\boldsymbol{\varphi})$$

Where a is the amplitude, x is the input timed signal, t is time delay of every wave, f is the frequency, d is the standard deviation, $\boldsymbol{\varphi}$ is the phase of the waveform. Playing with these parameters we obtain a signal very far in precision to the required signal but having the same form.

The segments and especially the S-T segment are not always constant in reality so an exponential function is added to show the inconsistency in these types of segments for that an exponential function is used:

$a * e^{b*x}$, where a and b are changed regarding the segment.

Noise was also added since all heartbeats have noise due to machine error and other source of artifacts, the Gaussian noise addition was used.

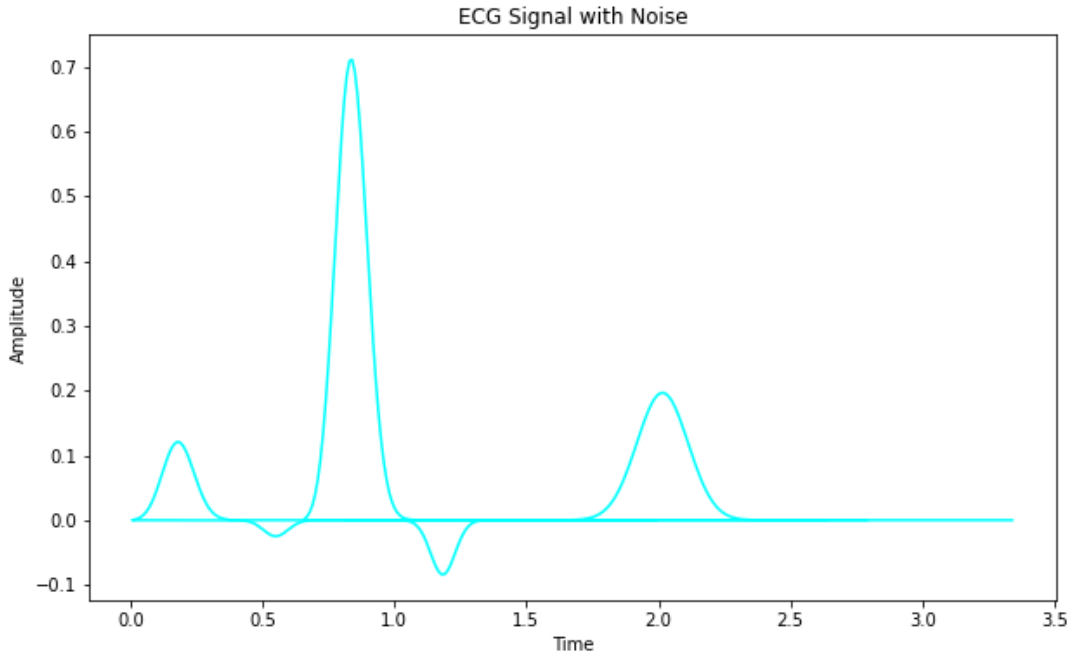


Figure 39 first result for the simulation of the QRS complex

This figure above shows the resulted QRS complex with the P and T waves, it looks very similar to an ECG signal but the realism in this figure is not present, for that, we activate the segment's exponential functions and use processing techniques to smoothen the signal.

Least square method was used to smoothen and alter the signal in a more realistic way. It consists of finding the m and c in (m*t +c) in order to minimize the sum of the squared differences between two waveform values, this replaces the two values with a smoother line and creates realism when two consecutive points are far from each other.

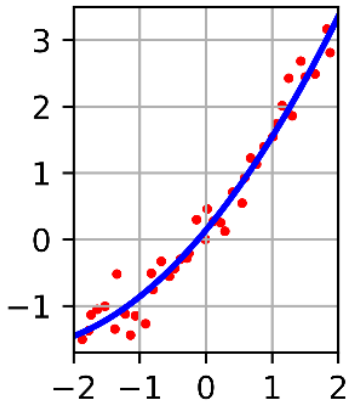


Figure 40 Least square method

To do this method, an algorithm should be put in place to find the least squared values all over the signal, but in python libraries a simple use of the **np.linalg.lstsq()** function calculates that and the method becomes much easier for compiling.

In addition, a **wavelet transform** was also executed to obtain a much more realistic signal. This smoothing method is widely used in the signal processing domain. In the ECG processing world, the most used type of wavelet transform is **Daubechies**. It has many types and decomposition levels. This is the algorithm followed to apply this smoothing technic:

1. Choose a wavelet function: Chosen db4 which is widely used in ECG signal processing
2. Choose a decomposition level, chose a high decomposition level to elevate and enhance the details of the decomposition
3. Use a high-pass filter and a low-pass filter to obtain coefficients at a current level
4. Threshold the coefficients obtained in the previous step by calculating the threshold value by the soft threshold technic
5. Compare the values to the threshold, if the value is below the threshold, it becomes 0, else it stays the same
6. Reconstruct the signal by using the coefficients and applying reverse wavelet transform
7. Repeat the process for every level

Using the pywavelet, every step in this algorithm is a function used in python.

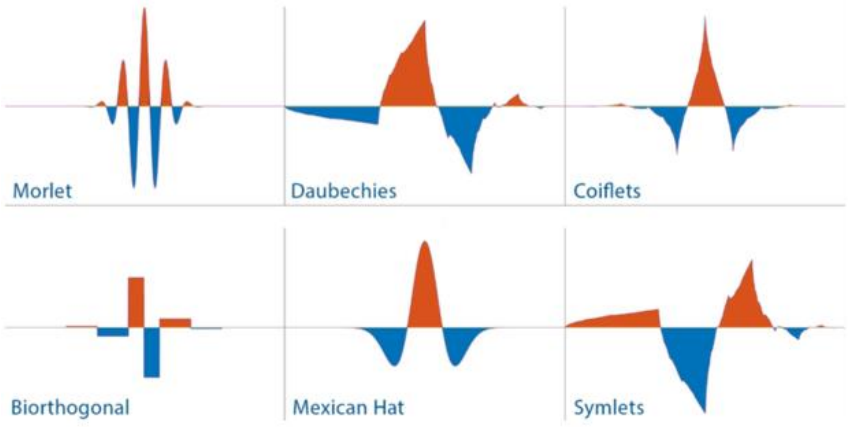


Figure 41 wavelet transform methods

Since we don't want symmetric components but asymmetric ones, we chose the Daubechies wavelets, and since it has different levels, trying to find the best one while compiling this method was easy.

Finally, we chose a heartbeat that had no abnormalities from the data base we already have and changed the parameters to obtain a very similar heartbeat simulation.

To compare the simulated and real signals, a comparison function using the RMSE method was compiled to help us compare the two signals

The RMSE which is the root to mean square error is a method used to compare signals

$$RMSE = sqrt(sum((y\_true - y\_simulated)\text{^}2) / n)$$

After compiling multiple times and changing the values of parameters of all the functions used, and applying the wavelet transform of Daubechies 5 times in a row for maximum smoothing, we obtained the following results

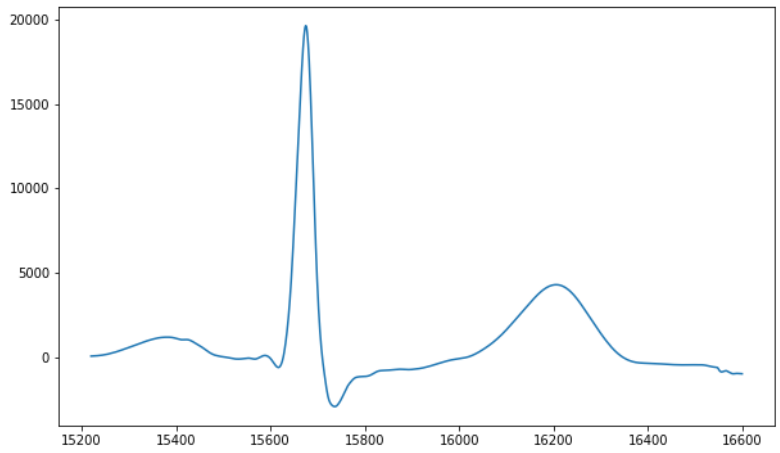


Figure 42 Real ECG signal

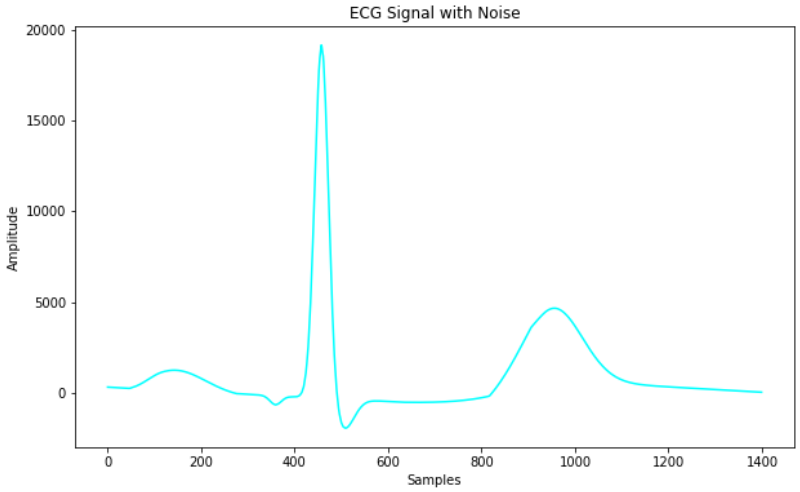


Figure 43 Simulated ECG signal

The output of the error function is 4.300656629943981%.

III.5 Conclusion

The simulation of ECG signals is a wider problem than anticipated, the simulation could be of 12 lead ECG signals, of a certain heart arrythmia like the VT or of a single QRS complex, either way using all of these results could change and affect positively the world of biomedical engineering in cardiac domain. The simulation done by prediction is accurate but is limited by the fact that a perfect data-set should exist. The simulation of polymorphic VT is limited by a very specific simulation and not all heartbeats and every ECG. The only simulation that has close to no limitations is the last simulation, it gave very positive results as it is really close to a normal heartbeat by 95%. This simulation program could then be automated and randomized, a frequency of sampling of many randomized parameters between different simulation of the same type could be done to create a higher number of QRS complexes, this could give us an ECG signal of multiple heartbeats. The limitation that exists in this domain is the number of compiling tasks that should be done to obtain a proper ECG signal.

CONCLUSION

The heart muscle is really complicated. Even through many simplifications and algorithms used to make this study much easier, the results were not perfect. The heart is a complex muscle and the cardiac electricity system shows that the evaluation of its signals made us understand how it works, the parameters that are affecting it. To help cardiac ablation and similar operations, this study was made to make these types of operations much faster and more precise, since operations have less risk, the less time passes during the procedure. For that, the CNN algorithm in the deep learning world was used to create an architectural model and train it to achieve a classification of the signals in an accurate method. Since the problem was simplified to a binary classification, either left or right ventricular outflow tact, the results were promising. Nevertheless, they were not satisfying since a high error of around 20% exists in this classification. Therefore, simulation was the next method that was chosen to try and work on the limitation of the CNN. Knowing that the simulation study is not only great for this particular method but is a recent task that not many scientists and specialists have worked with. If there is a perdition of the biomedical engineering future, we could all agree on is the dominance of the artificial intelligence over the medical field. With that being said three types of simulations were made. A first simulation that was linked to the existence of a data-set of other real heartbeats, it should great success in the simulation of a 12-lead ECG. Another simulation limited by only simulating polymorphic VTs, also showed great potential in the study of this particular arrythmia. Finally, the last simulation made a revolution in the simulation of QRS signals as it was so close the real heartbeat. Altering the parameters in small amounts could make us another signal of a heartbeat, this one is safe to be considered a precise and accurate simulation of a QRS complex.

REFERENCES

Jan-Christoph Edelmann, Dominik Mair, Daniel Ziesel, Martin Burtscher & Thomas Ussmueller (2018) An ECG simulator with a novel ECG profile for physiological signals, Journal of Medical Engineering & Technology, 42:7, 501-509, DOI: 10.1080/03091902.2019.1576788

Ian Goodfellow Yoshua Bengio and Aaron Courville (2017) Deep Learning, LCCN 2016022992 | ISBN 9780262035613

Bob Hood, (2011), Writing a Python Plugin, NewTek, Inc.

Boonstra MJ, Oostendorp TF, Roudijk RW, Kloosterman M, Asselbergs FW, Loh P and Van Dam PM (2022), Incorporating structural abnormalities in equivalent dipole layer based ECG simulations. Front. Physiol. 13:1089343. doi: 10.3389/fphys.2022.1089343

R. KARTHIK, (2008), ECG SIMULATION USING MATLAB Principle of Fourier Series, College of Engineering, Guindy, Anna University, Chennai – 600025

Zachary P. Kilpatrick, (2013), Wilson-Cowan Model, DOI 10.1007/978-1-4614-7320-6_80-1 # Springer Science+Business Media New York 2013

Dössel, O.; Luongo, G.; Nagel, C.; Loewe, A. Computer Modeling of the Heart for ECG Interpretation—A Review. Hearts (2021), 2, 350–368. Retrieved from: https://doi.org/10.3390/hearts2030028

Jianwei Zheng, Guohua Fu, Kyle Anderson1, Huimin Chu & Cyril Rakovski, (2020), A 12-Lead ECG database to identify origins of idiopathic ventricular arrhythmia containing 334 patients, retrieved from: https://doi.org/10.1038/s41597-020-0440-8

Zheng J, Fu G, Abudayyeh I, Yacoub M, Chang A, Feaster WW, Ehwerhemuepha L, El-Askary H, Du X, He B, Feng M, Yu Y, Wang B, Liu J, Yao H, Chu H and Rakovski C (2021) A High-Precision Machine Learning Algorithm to Classify Left and Right Outflow Tract Ventricular Tachycardia. Front. Physiol. 12:641066. doi: 10.3389/fphys.2021.641066

Tomofumi Nakamura, Yasutoshi Nagata, Giichi Nitta, Shinichiro Okata, Masashi Nagase, Kentaro Mitsui, Keita Watanabe, Ryoichi Miyazaki, Masakazu Kaneko, Sho Nagamine, Nobuhiro Hara, Tetsumin Lee, Toshihiro Nozato, Takashi Ashikaga, Masahiko Goya, Tetsuo Sasano, (2020), Prediction of premature ventricular complex origins using artificial intelligence–enabled algorithms, Heart Rhythm Society.

Mayo clinic, (2022), Electrocardiogram (ECG or EKG), retrieved from: https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983

Mayo clinic, (2023), cardiac ablation, retrieved from: https://www.mayoclinic.org/tests-procedures/cardiac-ablation/about/pac-20384993

Wikipedia, (2023), Electrocardiography, retrieved from: https://en.wikipedia.org/wiki/Electrocardiography

Wikipedia, (2023), Premature ventricular contraction, retrieved from: https://en.wikipedia.org/wiki/Premature_ventricular_contraction

Wikipedia, (2023), Cardiac Cycle, retrieved from: https://en.wikipedia.org/wiki/Cardiac_cycle

Wikipedia, (2023), Savitzky-Golay filter, retrieved from: https://en.wikipedia.org/wiki/Savitzky%E2%80%93Golay_filter

Wikipedia, (2023), Least squares, retrieved from: https://en.wikipedia.org/wiki/Least_squares

Wikipedia, (2023), Daubechies wavelet, retrieved from: https://en.wikipedia.org/wiki/Daubechies_wavelet

Vitali, (2023), 12-Lead ECG Placement Guide, retrieved from: https://www.vitalipartners.com/blog/2023/01/12-lead-ecg-placement/

J.A. Groeneweg & J. F. van der Heijden & D. Dooijes &T. A. B. van Veen & J. P. van Tintelen & R. N. Hauer, (2014), Arrhythmogenic cardiomyopathy: diagnosis, genetic background, and risk management, DOI 10.1007/s12471-014-0563-7

Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.

Caner C, Engin M, Engin EZ. The programmable ECG simulator. J Med Syst. 2008 Aug;32(4):355-9. doi: 10.1007/s10916-008-9140-1. PMID: 18619099.

Edelmann JC, Mair D, Ziesel D, Burtscher M, Ussmueller T. An ECG simulator with a novel ECG profile for physiological signals. J Med Eng Technol. 2018 Oct;42(7):501-509. doi: 10.1080/03091902.2019.1576788. Epub 2019 Feb 18. PMID: 30773952.

Diware S, Dash S, Gebregiorgis A, Joshi RV, Strydis C, Hamdioui S, Bishnoi R. Severity-Based Hierarchical ECG Classification Using Neural Networks. IEEE Trans Biomed Circuits Syst. 2023 Feb;17(1):77-91. doi: 10.1109/TBCAS.2023.3242683. PMID: 37015138.

Mincholé A, Camps J, Lyon A, Rodríguez B. Machine learning in the electrocardiogram. J Electrocardiol. 2019 Nov-Dec;57S:S61-S64. doi: 10.1016/j.jelectrocard.2019.08.008. Epub 2019 Aug 8. PMID: 31521378.

ANNEXE

The codes inserted below were written and used in the Spyder IDE and executed using the same program, the required libraries were downloaded to a specified environment that was named “tf”. Please find all of the codes in python format inside this google drive link:

https://drive.google.com/drive/folders/1BJ5JkOV5iPfB7Q5M7Md7rK_7CJs2BxXy

Link that provides visualization of the polymorphic data set:
https://www.physionet.org/lightwave/?db=incartdb/1.0.0

slicing temp.py

```
import os
import wfdb
import matplotlib.pyplot as plt

data_dir= 'E:/5th year/pfe/data/polymorph'
filename = 'I75'

#462600 is max sampto: 30 min * 60 sec * 257 Hz = N of samples

record = wfdb.rdrecord(os.path.join(data_dir, filename))
QRS_SAMPLES = 15000
start_sample = 0
end_sample = 450000

for i in range(start_sample, end_sample, QRS_SAMPLES):

    j = i + QRS_SAMPLES
    if j > end_sample:
        j = end_sample
    slice_filename = f"{filename}_slice_{i}_{j}.png"
    fig, axs = plt.subplots(nrows=12, ncols=1, figsize=(200,100))
    for k, ax in enumerate(axs):
        ax.plot(record.p_signal[i:j, k])
        ax.set_title(f'Lead {k+1}')

    plt.savefig(os.path.join(data_dir, slice_filename))
    plt.close()
```

temp.py

```
import pandas as pd
import matplotlib.pyplot as plt

column_index = 11 # leads 0 to 11
start_sample = 15220  # Specify the starting sample index
```

```
end_sample = 16600  # Specify the ending sample index
figsize = (10, 6)  # Specify the figsize (width, height)

df = pd.read_csv(r'E:\5th year\pfe\data\monomorph\PVCVTECGData\1067472.csv')

df_col = df.iloc[start_sample:end_sample, column_index]
plt.figure(figsize=figsize)
plt.plot(df_col)
plt.show()
```

temp2.py

```
import os
import wfdb
import matplotlib.pyplot as plt

data_dir= 'E:/5th year/pfe/data/polymorph'
data_dir_good= 'E:/5th year/pfe/data/polymorph/1-data'
filename = 'I01'

#462600 is max sampto
temp=0

record = wfdb.rdrecord(os.path.join(data_dir, filename), sampfrom=temp,
sampto=15000 + temp)

lead_names = ['I', 'II', 'III', 'AVR', 'AVL', 'AVF', 'V1', 'V2', 'V3', 'V4',
'V5', 'V6']

fig, axs = plt.subplots(nrows=12, ncols=1, figsize=(200,100))
for i, ax in enumerate(axs):
    ax.plot(record.p_signal[:, i])
    ax.set_title(f'{lead_names[i]} - Lead {i+1}')
#plt.savefig(os.path.join(data_dir, f"{filename}_sliced-on_{temp}" +
'_plot.png'))
plt.savefig(os.path.join(data_dir_good, f"{filename}_sliced-on_{temp}" +
'_plot.png'))
```

jpeg temp.py

```
import os
import wfdb
import matplotlib.pyplot as plt

data_dir= 'E:/5th year/pfe/data/polymorph'
data_dir_good= 'E:/5th year/pfe/data/polymorph/good data'
filename = 'I01'

#462600 is max sampto
temp=5900
```

```
record = wfdb.rdrecord(os.path.join(data_dir, filename), sampfrom=temp,
sampto=15000 + temp)

lead_names = ['I', 'II', 'III', 'AVR', 'AVL', 'AVF', 'V1', 'V2', 'V3', 'V4',
'V5', 'V6']

fig, axs = plt.subplots(nrows=12, ncols=1, figsize=(200,100))
for i, ax in enumerate(axs):
    ax.plot(record.p_signal[:, i])
    ax.set_title(f'{lead_names[i]} - Lead {i+1}')
plt.savefig(os.path.join(data_dir, f"{filename}_sliced-on_{temp}" +
'_plot.jpg'))
```

npz temp.py

```
import os
import wfdb
import numpy as np

data_dir = 'E:/5th year/pfe/data/polymorph'
data_dir_npz = 'E:/5th year/pfe/data/polymorph/SIMX'
filename = 'I19'

# 462600 is max sampto
temp = 240000
record = wfdb.rdrecord(os.path.join(data_dir, filename), sampfrom=temp,
sampto=15000 + temp)

lead_names = ['I', 'II', 'III', 'AVR', 'AVL', 'AVF', 'V1', 'V2', 'V3', 'V4',
'V5', 'V6']

data = {}
for i in range(12):
    lead_data = record.p_signal[:, i]
    lead_label = record.sig_name[i]
    data[f'{lead_names[i]}'] = lead_data
    data[f'{lead_names[i]}_label'] = lead_label

np.savez(os.path.join(data_dir_npz, f"{filename}_sliced-on_{temp}" + '.npz'),
**data)
```

combine npz.py

```
import os
import numpy as np

def combine_npz_files(folder_path, output_file):
    npz_files = [f for f in os.listdir(folder_path) if f.endswith('.npz')]
    combined_arrays = {}

    for file in npz_files:
```

```
        data = np.load(os.path.join(folder_path, file))
        for key in data.files:
            if key in combined_arrays:
                combined_arrays[key] = np.concatenate((combined_arrays[key],
data[key]))
            else:
                combined_arrays[key] = data[key]

    np.savez(output_file, **combined_arrays)
    print(f"Combined NPZ files saved as {output_file}")

folder_path = "E:/5th year/pfe/data/polymorph/npz-data"
output_file = "E:/5th year/pfe/data/polymorph/combined-npz"

combine_npz_files(folder_path, output_file)
```

sim1.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
#from scipy import signal
from scipy.signal import savgol_filter

input_dir = 'E:/5th year/pfe/data/polymorph/npz-data'
output_dir = 'E:/5th year/pfe/data/polymorph/preprocessed-data'

def preprocess_data(data):
    # Savitzky-Golay smoothing filter
    smoothed_data = {}
    for lead, signal_data in data.items():
        smoothed_signal = savgol_filter(signal_data, window_length=51,
polyorder=6)
        smoothed_data[lead] = smoothed_signal

    # Normalize the signals
    preprocessed_data = {}
    for lead, signal_data in smoothed_data.items():
        normalized_signal = (signal_data - np.mean(signal_data)) /
np.std(signal_data)
        preprocessed_data[lead] = normalized_signal

    return preprocessed_data

# Iterate over all NPZ files in the input directory
for filename in os.listdir(input_dir):
    if filename.endswith('.npz'):
        # Load the ECG signals from the NPZ file
        data = np.load(os.path.join(input_dir, filename))

        # Preprocess the ECG data
        preprocessed_data = preprocess_data(data)
```

```
        # Save the preprocessed data to the same NPZ file in the output
directory
        output_filename = os.path.join(output_dir, filename)
        np.savez(output_filename, **preprocessed_data)

# Plot the preprocessed data of the first file
data = np.load(os.path.join(input_dir, os.listdir(input_dir)[0]))
preprocessed_data = preprocess_data(data)
fig, axs = plt.subplots(nrows=12, ncols=1, figsize=(200, 100))
for i, lead in enumerate(preprocessed_data.keys()):
    axs[i].plot(preprocessed_data[lead][:15000])
    axs[i].set_title(f'{lead} - Lead {i+1}')
    axs[i].set_xlim([0, 15000])
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'preprocessed_data_plot.png'))
plt.show()

print("Preprocessing completed successfully.")
```

testing in general.py

```
import tensorflow as tf
print(tf.__version__)
#import sys
#print(sys.version)
```

Working on 5 files only-sim.py

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers

# Set the path for the input directory
input_dir = 'E:/5th year/pfe/data/polymorph/preprocessed-data'

# Step 1: Load the preprocessed data
data = []
labels = []

lead_names = ['I', 'II', 'III', 'AVR', 'AVL', 'AVF', 'V1', 'V2', 'V3', 'V4',
'V5', 'V6']

for filename in os.listdir(input_dir):
    if filename.endswith('.npz'):
        npz_data = np.load(os.path.join(input_dir, filename))
        for lead_name in lead_names:
            lead_data = npz_data[lead_name]
            num_samples = lead_data.shape[0]
            data.append(lead_data)
```

```
            labels.append(lead_names.index(lead_name))

# Convert the data and labels to arrays
data = np.array(data)
labels = np.array(labels)

# Reshape the data to have 3 dimensions
data = np.reshape(data, (data.shape[0], data.shape[1], 1))

# Get the shape of the input data
num_samples = data.shape[1]
num_channels = data.shape[2]

num_classes = len(lead_names)

model = tf.keras.Sequential([
    layers.Conv1D(32, kernel_size=3, activation='relu',
input_shape=(num_samples, num_channels)),
    layers.MaxPooling1D(pool_size=2),
    layers.Conv1D(64, kernel_size=3, activation='relu'),
    layers.MaxPooling1D(pool_size=2),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
batch_size = 32
epochs = 20

model.fit(data, labels, batch_size=batch_size, epochs=epochs,
validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(data, labels)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

simulation-1st.py

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt

os.chdir('E:/5th year/pfe/data/polymorph')
```

```
# Set the path for the input directory
input_dir = 'E:/5th year/pfe/data/polymorph/data-5'
replacable_file_path = 'E:/5th year/pfe/data/polymorph/sim-x'

# Step 1: Load the preprocessed data
data = []
labels = []

lead_names = ['I', 'II', 'III', 'AVR', 'AVL', 'AVF', 'V1', 'V2', 'V3', 'V4',
'V5', 'V6']

for filename in os.listdir(input_dir):
    if filename.endswith('.npz'):
        npz_data = np.load(os.path.join(input_dir, filename))
        for lead_name in lead_names:
            lead_data = npz_data[lead_name]
            num_samples = lead_data.shape[0]
            data.append(lead_data)
            labels.append(lead_names.index(lead_name))

# Convert the data and labels to arrays
data = np.array(data)
labels = np.array(labels)

# Reshape the data to have 4 dimensions
data = np.reshape(data, (data.shape[0], data.shape[1], 1, 1))

# Get the shape of the input data
num_samples = data.shape[1]
num_channels = data.shape[2]

num_classes = len(lead_names)

model = tf.keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 1), activation='relu',
input_shape=(num_samples, num_channels, 1)),
    layers.MaxPooling2D(pool_size=(2, 1)),
    layers.Conv2D(64, kernel_size=(3, 1), activation='relu'),
    layers.MaxPooling2D(pool_size=(2, 1)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
batch_size = 32
epochs = 10
```

```
model.fit(data, labels, batch_size=batch_size, epochs=epochs)

# Load the replacable ECG data
replacable_data = np.load(replacable_file_path)

# Generate simulated ECG data based on the replacable data
num_leads = len(lead_names)
num_samples = replacable_data[lead_names[0]].shape[0]

simulated_data = np.zeros((num_samples, num_leads, num_channels, 1))
prediction = model.predict(simulated_data.reshape(num_samples, num_leads,
num_channels, 1))
lead_values = np.argmax(prediction, axis=-1)
simulated_data[:, :, 0, 0] = lead_values

print("Simulation completed.")

# Overwrite the replacable ECG data with the simulated data
for lead_name in lead_names:
    replacable_data[lead_name] = simulated_data[:,
lead_names.index(lead_name), 0, 0]

# Save the modified replacable ECG data
np.savez(replacable_file_path, **replacable_data)

# Reshape the simulated data
simulated_data = np.transpose(simulated_data, (1, 0, 2, 3))

# Plot the simulated ECG 12-lead data
fig, axs = plt.subplots(nrows=num_leads, ncols=1, figsize=(200, 100))

for i, ax in enumerate(axs):
    ax.plot(simulated_data[:, i, 0, 0])
    ax.set_title(f'{lead_names[i]} - Lead {i + 1}')

plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.suptitle('Simulated ECG 12-Lead Data', fontsize=36)
plt.tight_layout()
plt.subplots_adjust(top=0.95)
plt.show()
```

pre-processing synchronized with model.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
#from scipy import signal
from scipy.signal import savgol_filter

input_dir = 'E:/5th year/pfe/data/polymorph/npz-data'
```

```
output_dir = 'E:/5th year/pfe/data/polymorph/preprocessed-data-synchro'

def preprocess_data(data):
    # Savitzky-Golay smoothing filter
    smoothed_data = {}
    for lead, signal_data in data.items():
        smoothed_signal = savgol_filter(signal_data, window_length=51,
polyorder=6)
        smoothed_data[lead] = smoothed_signal

    # Normalize the signals
    preprocessed_data = {}
    for lead, signal_data in smoothed_data.items():
        normalized_signal = (signal_data - np.mean(signal_data)) /
np.std(signal_data)
        preprocessed_data[lead] = normalized_signal

    return preprocessed_data

# Iterate over all NPZ files in the input directory
for filename in os.listdir(input_dir):
    if filename.endswith('.npz'):
        # Load the ECG signals from the NPZ file
        data = np.load(os.path.join(input_dir, filename))

        # Preprocess the ECG data
        preprocessed_data = preprocess_data(data)

        # Save the preprocessed data to the same NPZ file in the output
directory
        output_filename = os.path.join(output_dir, filename)
        np.savez(output_filename, **preprocessed_data)

# Plot the preprocessed data of the first file
data = np.load(os.path.join(input_dir, os.listdir(input_dir)[0]))
preprocessed_data = preprocess_data(data)
fig, axs = plt.subplots(nrows=12, ncols=1, figsize=(200, 100))
for i, lead in enumerate(preprocessed_data.keys()):
    axs[i].plot(preprocessed_data[lead][:15000])
    axs[i].set_title(f'{lead} - Lead {i+1}')
    axs[i].set_xlim([0, 15000])
plt.tight_layout()
plt.savefig(os.path.join(output_dir, 'preprocessed_data_plot.png'))
plt.show()

print("Preprocessing completed successfully.")
```

simulation using mathematical method.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
```

```
from scipy.interpolate import interp1d

output_directory = 'E:/5th year/pfe/simulation task'

def ecgsyn(sfecg=None, N=None, Anoise=None, hrmean=None, hrstd=None,
lfhfratio=None, sfint=None, ti=None, ai=None, bi=None):
    if sfecg is None:
        sfecg = 256
    if N is None:
        N = 10000
    if Anoise is None:
        Anoise = 0.1
    if hrmean is None:
        hrmean = 60
    if hrstd is None:
        hrstd = 1
    if lfhfratio is None:
        lfhfratio = 0.5
    if sfint is None:
        sfint = 512
    if ti is None:
        ti = np.deg2rad([-70, -15, 0, 15, 100])

    # Convert angles to radians
    ti = np.deg2rad(ti)

    if ai is None:
        ai = np.array([1.2, -5, 30, -7.5, 0.75])
    if bi is None:
        bi = np.array([0.25, 0.1, 0.1, 0.1, 0.4])

    # Adjust extrema parameters for mean heart rate
    hrfact = np.sqrt(hrmean / 60)
    hrfact2 = np.sqrt(hrfact)
    bi = hrfact * np.array(bi)
    ti = np.multiply([hrfact2, hrfact, 1, hrfact, hrfact2], ti)

    # Check that sfint is an integer multiple of sfecg
    q = int(sfint / sfecg)
    qd = sfint / sfecg
    if q != qd:
        raise ValueError("Internal sampling frequency (sfint) must be an
integer multiple of the ECG sampling frequency (sfecg).")

    # Define frequency parameters for rr process
    flo = 0.1
    fhi = 0.25
    flostd = 0.01
    fhistd = 0.01

    fid = 1
    print(f"ECG sampled at {sfecg} Hz")
    print(f"Approximate number of heart beats: {N}")
```

```
    print(f"Measurement noise amplitude: {Anoise}")
    print(f"Heart rate mean: {hrmean} bpm")
    print(f"Heart rate std: {hrstd} bpm")
    print(f"LF/HF ratio: {lfhfratio}")
    print(f"Internal sampling frequency: {sfint}")
    print("      P  Q  R  S  T")
    print(f"ti = {ti} radians")
    print(f"ai = {ai}")
    print(f"bi = {bi}")

    # Calculate time scales for rr and total output
    sampfreqrr = 1
    trr = 1 / sampfreqrr
    tstep = 1 / sfecg
    rrmean = 60 / hrmean

    # Calculate number of rr samples to be generated
    Nrr = int(np.ceil(N * tstep / trr))

    # Calculate total number of samples to be generated
    Ntot = int(np.ceil(N * tstep / trr * sfint / sfecg))

    # Calculate rr samples
    rr0 = rrprocess(flo, fhi, flostd, fhistd, lfhfratio, hrmean, hrstd,
sampfreqrr, Nrr)

    # Upsample rr
    t = np.arange(0, Nrr)
    f = interp1d(t, rr0, kind='cubic')
    tnew = np.arange(0, Nrr - 1, 1 / sfint)
    rr = f(tnew)

    # Calculate total number of samples to be generated
    Ntot = len(rr)

    # Initialize ecg
    ecg = np.zeros(Ntot)

    # Generate noise
    Anoise = Anoise * np.std(ecg)
    noise = Anoise * np.random.randn(Ntot)

    # Generate heartbeats
    ind = np.cumsum(rr)
    ti = np.concatenate(([0], np.cumsum(ti)))
    t = np.arange(1, Ntot + 1) / sfint
    for i in range(0, len(rr) - 1):
        t0 = np.arange(ind[i], ind[i + 1], 1)
        if t0[-1] >= len(ecg):  # Skip if t0 is out of bounds
            continue
        ti_i = ti[i % len(ti)] if ti[i % len(ti)] != 0 else 1e-6  # Handle
zero values in ti
        ai_i = ai[i % len(ai)]  # Fix index out of range error
```

```
        P = bi[0] * np.random.randn() + ai_i * np.random.randn() * np.exp(-
0.5 * ((t0 / ti_i) ** 2))
        Q = bi[1] * np.random.randn() + ai_i * np.random.randn() * np.exp(-
0.5 * ((t0 / ti_i) ** 2))
        R = bi[2] * np.random.randn() + ai_i * np.random.randn() * np.exp(-
0.5 * ((t0 / ti_i) ** 2))
        S = bi[3] * np.random.randn() + ai_i * np.random.randn() * np.exp(-
0.5 * ((t0 / ti_i) ** 2))
        T = bi[4] * np.random.randn() + ai_i * np.random.randn() * np.exp(-
0.5 * ((t0 / ti_i) ** 2))
        ecg[t0.astype(int)] = P + Q + R + S + T

    # Add noise
    ecg = ecg + noise

    return ecg

def rrprocess(flo=None, fhi=None, flostd=None, fhistd=None, lfhfratio=None,
hrmean=None, hrstd=None, sampfreq=None, N=None):
    if flo is None:
        flo = 0.1
    if fhi is None:
        fhi = 0.25
    if flostd is None:
        flostd = 0.01
    if fhistd is None:
        fhistd = 0.01
    if lfhfratio is None:
        lfhfratio = 0.5
    if hrmean is None:
        hrmean = 60
    if hrstd is None:
        hrstd = 1
    if sampfreq is None:
        sampfreq = 1
    if N is None:
        N = 10000

    # Calculate number of low frequency components and number of high
frequency components
    nlf = int(np.ceil(lfhfratio / (1 + lfhfratio) * N))
    nhf = N - nlf

    # Generate uniformly distributed random numbers
    U = np.random.rand(nlf) * (fhi - flo) + flo
    V = np.random.randn(nlf) * flostd
    X = U + V

    # Generate normal distributed random numbers
    Y = np.random.randn(nhf) * fhistd

    # Concatenate and shuffle low and high frequency components
    Z = np.concatenate((X, Y))
```

```
    np.random.shuffle(Z)

    # Adjust extrema parameters for mean heart rate
    hrfact = np.sqrt(hrmean / 60)
    Z = Z * hrfact

    # Add mean heart rate to the signal
    M = np.mean(Z)
    Z = Z - M + hrmean

    # Add Gaussian distributed noise
    Z = Z + np.random.randn(N) * hrstd

    return Z

# Example usage
sfecg = 256
N = 10000
Anoise = 0.1
hrmean = 60
hrstd = 1
lfhfratio = 0.5
sfint = 512
ti = [-70, -15, 0, 15, 100]
ai = [1.2, -5, 30, -7.5, 0.75]
bi = [0.25, 0.1, 0.1, 0.1, 0.4]

ecg_signal = ecgsyn(sfecg, N, Anoise, hrmean, hrstd, lfhfratio, sfint, ti,
ai, bi)

# Add noise to the signal
noise_amplitude = 1  # Adjust the noise amplitude as desired
noise = noise_amplitude * np.random.randn(len(ecg_signal))
ecg_signal_with_noise = ecg_signal + noise

# Plot the ECG signal
figsize = (50, 10)
plt.figure(figsize=figsize)
t = np.arange(len(ecg_signal)) / sfecg
plt.plot(t, ecg_signal)
plt.xlabel('Time (s)')
plt.ylabel('ECG Amplitude')
plt.title('Simulated ECG Signal')

save_path = os.path.join(output_directory, 'ecg_plymorphe_sim.png')
plt.savefig(save_path)
plt.close()
```

simulation-ecg-function.py

```
import os
import glob
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
import pywt

# Real data:
column_index = 11 # leads 0 to 11
start_sample = 15220  # Specify the starting sample index
end_sample = 16600  # Specify the ending sample index
figsize = (10, 6)

df = pd.read_csv(r'E:\5th year\pfe\data\monomorph\PVCVTECGData\1067472.csv')
output_directory = 'E:/5th year/pfe/simulation task/123'

df_col = df.iloc[start_sample:end_sample, column_index]
plt.figure(figsize=figsize)
plt.plot(df_col)

#save_path = os.path.join(output_directory, 'ecg_plot_real_V6_1067472.png')
#plt.savefig(save_path)
#plt.close()

plt.show()

def p_wav(x, a, d1, d2, t, li, start, finish, p):
    wave = np.zeros_like(x)

    x1 = x[(x >= start) & (x <= t)]
    x2 = x[(x > t) & (x <= finish)]

    wave[(x >= start) & (x <= finish)] = np.concatenate((
        a * np.exp(-((x1 - t) / d1) ** 2) * np.sin(2 * np.pi * li * x1 + p),
        a * np.exp(-((x2 - t) / d2) ** 2) * np.sin(2 * np.pi * li * x2 + p)
    ))

    return wave

def q_wav(x, a, d, t, li, start, finish, p):
    wave = np.zeros_like(x)

    wave[(x >= start) & (x <= finish)] = -a * np.exp(-((x[(x >= start) & (x
<= finish)] - t) / d) ** 2) * np.sin(2 * np.pi * li * x[(x >= start) & (x <=
finish)] + p)

    return wave

def qrs_wav(x, a, d, li, start, finish, p):
    wave = np.zeros_like(x)
```

```
    wave[(x >= start) & (x <= finish)] = a * np.exp(-((x[(x >= start) & (x <=
finish)] - 0.166) / d) ** 2) * np.sin(2 * np.pi * li * x[(x >= start) & (x <=
finish)] + p)

    return wave

def s_wav(x, a, d, t, li, start, finish, p):
    wave = np.zeros_like(x)

    wave[(x >= start) & (x <= finish)] = -a * np.exp(-((x[(x >= start) & (x
<= finish)] - t) / d) ** 2) * np.sin(2 * np.pi * li * x[(x >= start) & (x <=
finish)] + p)

    return wave

def t_wav(x, a, d1, d2, t, li, start, finish, p):
    wave = np.zeros_like(x)

    x1 = x[(x >= start) & (x <= t)]
    x2 = x[(x > t) & (x <= finish)]

    wave[(x >= start) & (x <= finish)] = np.concatenate((
        a * np.exp(-((x1 - t) / d1) ** 2) * np.sin(2 * np.pi * li * x1 + p),
        a * np.exp(-((x2 - t) / d2) ** 2) * np.sin(2 * np.pi * li * x2 + p)
    ))

    return wave

def u_wav(x, a, d, t, li, start, finish, p):
    wave = np.zeros_like(x)

    wave[(x >= start) & (x <= finish)] = a * np.exp(-((x[(x >= start) & (x <=
finish)] - t) / d) ** 2) * np.sin(2 * np.pi * li * x[(x >= start) & (x <=
finish)] + p)

    return wave

def add_noise(signal, mean=0, std=0.01):
    noise = np.random.normal(mean, std, len(signal))

    return signal + noise

def zero_wave():

    return 0

def exponential_wave(x, a, b, start, finish):
    wave = np.zeros_like(x)

    wave[(x >= start) & (x <= finish)] = a * np.exp(b * (x[(x >= start) & (x
<= finish)]))
```

```
    return wave

def calculate_signal_error(directory):
    # Get all CSV files in the directory
    csv_files = glob.glob(os.path.join(directory, '*.csv'))

    # Ensure at least two CSV files are present
    if len(csv_files) < 2:
        print("Error: At least two CSV files are required in the directory.")
        return

    # Read the first two CSV files with specified encoding
    df1 = pd.read_csv(csv_files[0], encoding='utf-8')
    df2 = pd.read_csv(csv_files[1], encoding='utf-8')

    # Extract the signal values from each dataframe
    signal1 = df1['Signal'].values
    signal2 = df2['Signal'].values

    # Check if signal arrays have different lengths
    if len(signal1) != len(signal2):
        # Resample or interpolate the signal with shorter length
        if len(signal1) < len(signal2):
            f = interp1d(np.arange(len(signal1)), signal1)
            signal1 = f(np.linspace(0, len(signal1)-1, len(signal2)))
        else:
            f = interp1d(np.arange(len(signal2)), signal2)
            signal2 = f(np.linspace(0, len(signal2)-1, len(signal1)))

    # Calculate the root-mean-square error (RMSE)
    mse = np.mean((signal1 - signal2) ** 2)
    rmse = np.sqrt(mse)

    # Calculate the error percentage
    max_value = np.max([np.max(signal1), np.max(signal2)])
    error_percentage = (rmse / max_value) * 100

    # Print the error percentage
    print(f"Error Percentage: {error_percentage}%")

def least_squares(wave):
    n = len(wave)
    t = np.arange(n)

    # Calculate the coefficients using the method of least squares
    A = np.vstack([t, np.ones(n)]).T
    m, c = np.linalg.lstsq(A, wave, rcond=None)[0]

    # Generate the modified wave using the obtained coefficients
    modified_wave = wave - (m * t + c)

    return modified_wave
```

```
def smooth_ecg_signal(signal):
    # Choose the wavelet function
    wavelet = 'db4'

    # Set the number of decomposition levels
    num_levels = 6

    # Perform wavelet decomposition
    coeffs = pywt.wavedec(signal, wavelet, level=num_levels)

    # Apply thresholding to the detail coefficients
    threshold = np.std(coeffs[-1]) * np.sqrt(2 * np.log2(len(signal)))
    coeffs = [pywt.threshold(detail, value=threshold, mode='soft') if idx ==
num_levels else detail
              for idx, detail in enumerate(coeffs)]

    # Reconstruct the signal using the modified coefficients
    reconstructed_signal = pywt.waverec(coeffs, wavelet)

    return reconstructed_signal

x_pwav = np.arange(0.1, 0.6, 0.01)
x_qwav = np.arange(0.2, 0.67, 0.01)
x_qrswav = np.arange(0.01, 0.415, 0.01)
x_swav = np.arange(0.26, 0.7405, 0.01)
x_twav = np.arange(0.001, 1, 0.01)
x_uwav = np.arange(0.01, 3, 0.01)
x_e = np.arange(0, 0.6, 0.01)

li = 30 / 72

a_pwav = 2000
d1_pwav = 0.14
d2_pwav = 0.22
t_pwav = 0.25
phase_p =-0.05

a_qwav = 500
d_qwav = 0.036
t_qwav = 0.65
phase_q =0

a_qrswav = 46000
d_qrswav = 0.045
phase_r = 0

a_swav = 2000
d_swav = 0.066
t_swav = 0.2
phase_s = 0

a_twav = 6000
d1_twav = 0.4
```

```
d2_twav = 0.2
t_twav = 0.25
phase_t = 0

a_uwav = 0.035
d_uwav = 0.1
t_uwav = 0.433
phase_u = 0

e_a = 100
e_b = 4.9

start = 0
finish = 3

pwav = p_wav(x_pwav, a_pwav + 0.05, d1_pwav, d2_pwav, t_pwav, li, start,
finish, phase_p)
qwav = q_wav(x_qwav, a_qwav, d_qwav, t_qwav - 0.1, li, 0.3, 0.75, phase_q)
qrswav = qrs_wav(x_qrswav, a_qrswav, d_qrswav, li, start, finish,phase_r)
swav = s_wav(x_swav, a_swav - 0.1, d_swav, t_qwav - 0.27, li, start,
finish,phase_s)
twav = t_wav(x_twav, a_twav - 0.1, d1_twav, d2_twav, t_twav, li, start,
finish,phase_t)
uwav = u_wav(x_uwav, a_uwav, d_uwav, t_qwav + 0.6, li, start, finish,
phase_u)  # Increase the distance between T and U wave
ewav = exponential_wave(x_e, e_a, e_b, start, finish)

# Calculate the common time axis
common_time = np.linspace(0, 3, 1400)

# Interpolate or resample each wave onto the common time axis
pwav_resampled = np.interp(common_time, x_pwav, pwav)
qwav_resampled = np.interp(common_time, x_qwav + 0.22, qwav)
qrswav_resampled = np.interp(common_time, x_qrswav + t_qwav + 0.16, qrswav)
swav_resampled = np.interp(common_time, x_swav + t_qwav + 0.05, swav)
twav_resampled = np.interp(common_time, x_twav + t_qwav + 1.1, twav)
uwav_resampled = np.interp(common_time, x_uwav + t_qwav + 1.4, uwav)
ewav_resampled = np.interp(common_time, x_e + t_qwav + 0.7, ewav)

# Calculate the Total signal
Total_1 = pwav_resampled + qwav_resampled + qrswav_resampled + swav_resampled
+ twav_resampled + uwav_resampled + ewav_resampled

# Least mean squares
Total_no_noise = least_squares(Total_1)

# work on an elevation
Total_no_noise += 1000

# Apply wavelet transform 5 times
Temp1 = smooth_ecg_signal(Total_no_noise)
Temp2 = smooth_ecg_signal(Temp1)
Temp3 = smooth_ecg_signal(Temp2)
```

```
Temp4 = smooth_ecg_signal(Temp3)
Temp5 = smooth_ecg_signal(Temp4)

# Add noise to the signals
Total = add_noise(Temp5)

# Plot the ECG signal with noise
plt.figure(figsize=(10, 6))
#plt.plot(x_pwav, pwav, color='cyan')
#plt.plot(x_qwav + 0.2, qwav, color='cyan')
#plt.plot(x_qrswav + t_qwav + 0.2 , qrswav, color='cyan')
#plt.plot(x_swav + t_qwav + 0.35, swav, color='cyan')
#plt.plot(x_twav + t_qwav + 1.1, twav, color='cyan')
#plt.plot(range(len(ewav_resampled)), ewav_resampled, color ='red')

plt.plot(range(len(Total)), Total, color='cyan')

plt.xlabel('Samples')
plt.ylabel('Amplitude')
plt.title('ECG Signal with Noise')

#plt.show()

# Save the plot as a PNG file
save_path = os.path.join(output_directory, 'ecg_plot_sim.png')
plt.savefig(save_path)
plt.close()

# Save the simulated ECG signals as CSV files
simulated_signals = {'Samples': range(len(Total)), 'Signal': Total}
df_simulated = pd.DataFrame(simulated_signals)
csv_path = os.path.join(output_directory, 'ecg_signal_sim.csv')
df_simulated.to_csv(csv_path, index=False)

# Save the real ECG signal as a CSV file
real_signal = {'Samples': range(len(df_col)), 'Signal': df_col}
df_real = pd.DataFrame(real_signal)
csv_path = os.path.join(output_directory, 'ecg_signal_real_V6_1067472.csv')
df_real.to_csv(csv_path, index=False)


calculate_signal_error(output_directory)
```