Analysing Electrocardiogram signals to Extract the PQRST Waves

CS39440 Major Project Report  
  
Author: Jake Newall ([jan21@aber.ac.uk](mailto:jan21@aber.ac.uk))  
  
Supervisor: Dr Otar Akanyeti (ota1@aber.ac.uk)

3rd April 2019  
Version 1.8 (Complete)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Science (GH7P)

Department of Computer Science   
Aberystwyth University   
Aberystwyth   
Ceredigion  
SY23 3DB  
Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name: Jake Newall

Date: 03/05/2019

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Jake Newall

Date: 03/05/2019

Acknowledgements

I am grateful to Dr Federico Villagra Povina for his help in recording the electrocardiogram data and for giving guidance on research throughout the project.

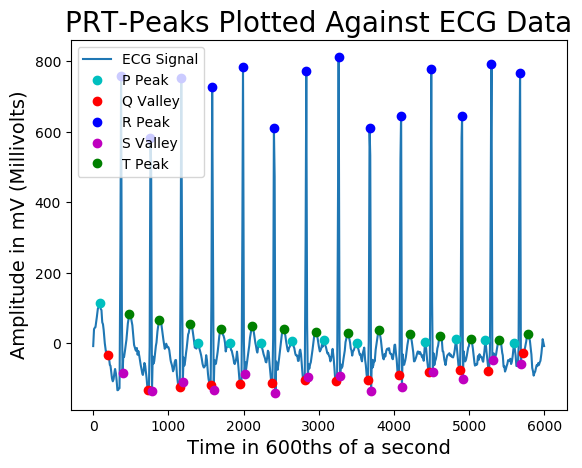
I’d like to thank Otar Akanyeti for providing support and offering guidance through group and mentor meetings during the project.

Abstract

Signal processing is an important skill which allows for the collection, filtering and manipulation of different time-series datasets which consist of change over time. This skill allows for the manipulation of audio, speech, image and video files along with many others.   
With the advancement of technology including Neural Networks, we can use key features taken from an electrocardiogram (ECG) dataset to automatically classify heart disorders [1].

The aim of this project is to record ECG signals using a clinical electrocardiograph and to manipulate these signals so that the main sections known as the PQRST sequence can be derived from the data.   
  
The PQRST sequence is important and is used by doctors and physicians inside of hospitals to diagnose certain cardiac diseases which in turn saves people’s lives.  
   
To do this, the noisy signal data is filtered using fast fourier transform (FFT) and cubic spline interpolation of the moving average to remove the biological and electromagnetic frequencies which were added during the recording process.  
   
The entire signal is then split into individual beats which are easier to manipulate.  
Peak detection will be utilized in order to detect the P, R and T sections whilst a valley finding algorithm is used to detect the Q and S sections.  
  
Graphs are created throughout using Matplotlib and saved into different folders to illustrate how the signal changes as more processing is carried out.  
  
No tests are written for the code provided.  
  
Finally, the PQRST data is saved to CSV files for further use.

During the project, the Agile Scrum methodology was used to keep a clear and concise goal for the following week and GitHub branches were created from the Scrum stories.



Contents

1. **Background ………………………………………………………………………………………………………….. 6**
   1. **The Heart ………………………………………………………………………………………………… 6**
      1. **Sinoatrial Node and Arrythmias …………………………………………………. 6**
   2. **The Electrocardiogram ……………………………………………………………………………. 7**
   3. **A Heart Beat and the PQRST Sequence ……………………………………………………. 7**
   4. **Current Methods for Detecting the PQRST Sequence ………………………………. 8**
   5. **Electrocardiographs and Alternative Sensors ………………………………………….. 8**
2. **Project Design and Initial Documentation …………………………………………………………….. 9**
   1. **Python 3 …………………………………………………………………………………………………. 9**
   2. **Real-Time Data vs Pre-recorded Data ……………………………………………………… 9**
   3. **Scrum ……………………………………………………………………………………………………… 9**
      1. **Trello ………………………………………………………………………………………… 10**
      2. **GitHub ………………………………………………………………………………………. 10**
   4. **Initial UML Diagram ………………………………………………………………………………… 11**
3. **Implementation ……………………………………………………………………………………………………. 13**
   1. **Recording the Electrocardiogram …………………………………………………………….. 13**
      1. **Data Formatting Issues ………………………………………………………………. 14**
   2. **Simple User Interface ………………………………………………………………………………. 14**
   3. **Importing and Exporting CSV Files ……………………………………………………………. 15**
   4. **Averaging the Signals ………………………………………………………………………………. 15**
   5. **Calculating the Mean for the Entire Averaged Dataset …………………………….. 15**
   6. **Calculating the Running Mean Datapoints ……………………………………………….. 16**
   7. **Calculating the Cubic Spline Interpolation of the Running mean ………………. 16**
   8. **Removing Baseline Drift …………………………………………………………………………… 17**
   9. **Removing Noise Using Fast Fourier Transform …………………………………………. 18**
   10. **Smoothing the Signal ……………………………………………………………………………….. 19**
   11. **Finding the R-Wave Peak …………………………………………………………………………. 20**
   12. **Splitting the Signal into Separate Beats ……………………………………………………. 21**
   13. **Finding QS Valleys ……………………………………………………………………………………. 22**
   14. **Finding PRS Peaks and Saving the Values …………………………………………………. 23**
   15. **Final UML Diagram ………………………………………………………………………………….. 25**
4. **Results ………………………………………………………………………………………………………………….. 26**
   1. **ECG Toolkit 2.4 CSV Dataset vs Sanitised ECG Dataset ……………………………… 26**
   2. **FFT Vs Non-FFT PQRST Sections ……………………………………………………………….. 26**
   3. **FFT Filtered PQRST Time Intervals ……………………………………………………………. 28**
   4. **Heartrates and Zones ………………………………………………………………………………. 29**
5. **Critical Evaluation …………………………………………………………………………………………………. 29**
   1. **Reliability ………………………………………………………………………………………………… 29**
   2. **Speed ………………………………………………………………………………………………………. 30**
   3. **Usability ………………………………………………………………………………………………….. 30**
6. **Conclusion …………………………………………………………………………………………………………….. 31**
7. **Project Deliverables**
8. **Annotated Bibliography**
9. **Appendices** 
   1. **Appendix A – Third Party Code and Libraries**
   2. **Appendix B – Ethics Submission**
   3. **Appendix C –**

**1 Background**

This chapter addresses the related biological research which was undertook for the project before explaining the process of converting a biological electrical signal into an ECG.   
The signal which has been created is finally explained before having a brief outlook at the different devices capable of recording a signal and how the signal is normally broken-down using software.

**1.a The Heart**

The heart is a muscular organ which contains 4 chambers. These 4 chambers and additional valves allow blood to flow around the body unidirectionally. Muscles and tissue require the oxygen which is carried in the blood in order to create energy through a chemical process called aerobic respiration. To allow this process to happen, the heart “pumps oxygenated blood throughout the body and deoxygenated blood to the lungs” [2].   
If the heart is not functioning properly, the body may not be getting the optimum level of blood in order to sustain itself which could lead to a multitude of issues.

**1.a.i Sinoatrial Node and Arrhythmias**

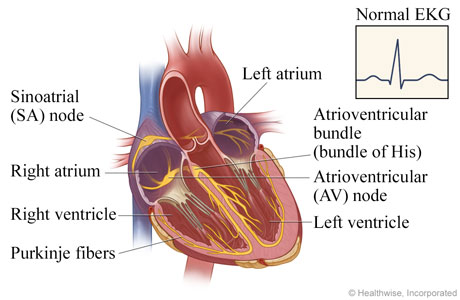
The sinoatrial node (SA node) ensures that the heart beats in a regular rhythm to ensure optimum blood flow which is one of the reasons it is also known as the body’s natural pacemaker. This is done with an electrical signal which “spreads through the walls of the atria and causes them to contract” [3].  
  
**1.** The signal initially starts at the SA node (located in the right atrium) and propagates to the left and right atria. This in turn causes them to depolarize and pump blood into the ventricles below.  
  
**2.** The Atrioventricular node (AV node) acts as a gate which slows the signal, allowing the ventricles to fill with blood.  
  
**3.** The AV node forwards the signal across the ventricles causing them to depolarize which pumps blood around the body. Whilst this is happening, the atria re-polarize.

Figure – Cross Section of the Heart [4]

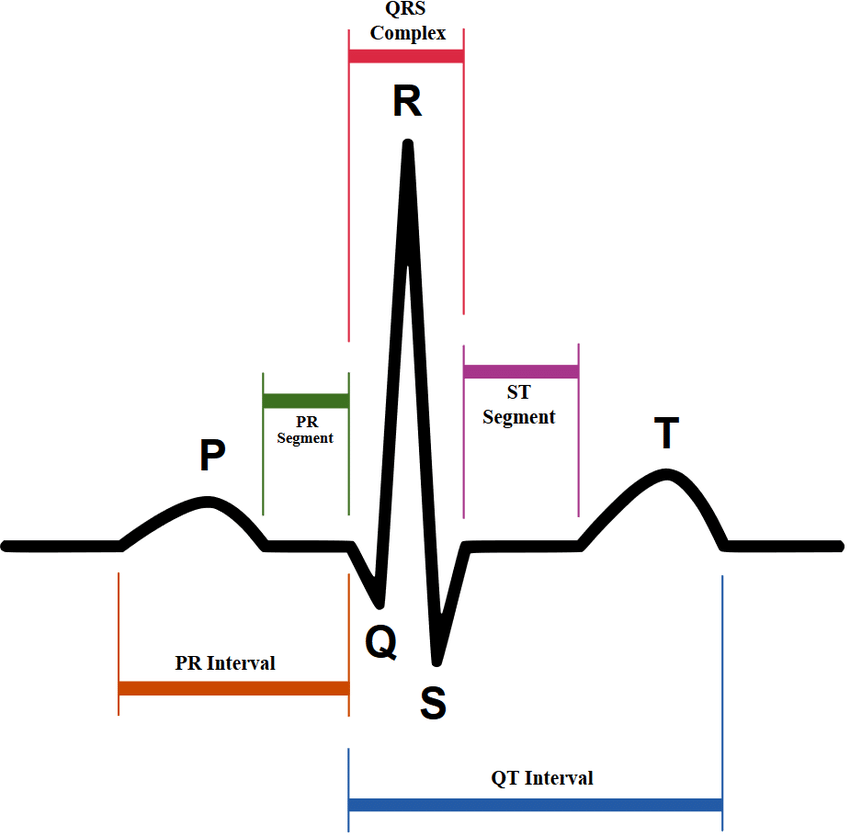
**4.** The ventricles finally re-polarize after all the blood has been pushed out of them. [4][5]

If the signal from either the SA node or the AV node is timed incorrectly, the heart can develop an arrhythmia. If the electrical activity of the heart is healthy, the rhythm will be diagnosed as a “normal sinus rhythm”. Sinus arrhythmia is directly related to an incorrect timing of the SA node [6].  
Atrial fibrillation (AFib) occurs when the SA node fires multiple signals causing a spasm in the atria making them less effective. This confuses the AV node which de-polarises the ventricles in an unorganised manner leading to an irregular heartbeat [3].  
  
In certain circumstances, it may be necessary to fit a patient with an artificial pacemaker that replaces the SA node.

**1.b The Electrocardiogram**

An electrocardiograph is a device which has multiple electrodes which are placed on the skin. These sensors detect the electrical signals which are produced by the SA node and AV node. The signals transmit up a wire into a computer which attempts to filter out any biological noise that might happen during the reading.   
  
The readings that the electrocardiograph takes are displayed as an ECG. ECG’s are commonly displayed as a 2-dimensional line graph with the x-axis resembling time and the y-axis resembling amplitude in millivolts (mV).  
  
There are multiple electrocardiographs on the market and each device allows for a different function. Certain electrocardiographs have been developed to remove as much signal noise as possible because a long-term study may need to be conducted on a mobile patient outside of a hospital. These machines are called Holter monitors.   
Clinical electrocardiographs such as the one which was used in this project utilize 12 leads with 10 electrodes and typically take readings for a 10 second period. Different filters can be implemented when setting up the recording software dependant on the study.

**1.c A Heart Beat and the PQRST Sequence**

After an ECG has been recorded using an electrocardiograph, singular beats can be broken down into their corresponding PQRST sections. These sections are easily characterizable by a medically trained professional if a high-quality signal has been recorded.  
Utilizing the points from 2.a.i:

1. The P wave is associated with point 1 – The left and right atria depolarising and pumping blood into the ventricles.
2. The Q wave relates to point 3 and early ventricular depolarisation.
3. The R wave relates to point 3 and the main ventricular depolarisation. The reason for its height (as seen in figure 2) is due to the larger size of the ventricles compared to the atria.
4. The S wave relates to point 3 and the depolarisation of the Purkinje fibres.
5. The T wave relates to point 4 and the repolarisation of the ventricles. [5]

Figure - PQRST Sequence [7]

Noise can be an issue when recording using an electrocardiograph and this can lead to three common issues:  
  
Baseline wander can be distinguished by seeing the overall signal change its amplitude over time causing a wave. This can make it difficult to distinguish certain parts of the PQRST sequence. It is often caused by body movements due to respiration but can also be caused by perspiration. Baseline wander was an issue in this study for my personal readings due to the perspiration from the stress of a clinical setting. Because of the biological nature of this issue, filtering is needed post recording [8][9].  
  
Power line interference is caused by an electromagnetic field that occurs in the powerlines which run from the electrodes to the electrocardiograph. This can add 60 Hz frequencies into the ECG making it hard to distinguish certain sections such as the P and T waves which have low amplitudes. Shielded electrode leads and keeping leads apart can help to stop the frequencies, but they can still commonly occur [8][9].

Electromyographic noise occurs due to electrical stimulation within the muscles where the electrodes are placed that can add interference. The noise can manifest as small changes of amplitude across the signal or inverted P, R and T waves. This can be filtered out in part by the electrocardiograph dependant on the model, however the noise can be reduced by using post filtering [8][9].

**1.d Current Methods for Detecting the PQRST Sequence**

As mentioned in 2.c, baseline wander, powerline interference and electromyographic noise cause issues when attempting to find feature in an ECG signal.  
  
One way of removing baseline wander is to calculate the moving average of the signal. Once this moving average has been calculated, cubic spline interpolation can be used to create a curve which follows the average as it changes over the signal. This average can then be used for future calculations instead of the normal mean of the data [8][9].

Powerline interference and electromyographic noise cannot be removed in the same way as baseline wander. These two issues make the ECG signal less smooth in appearance and can even “hide” peaks below the noise.  
One common way of dealing with the issue is to use Fast Fourier Transform (FFT). FFT calculates the frequency domain for a signal. The frequencies can then be filtered to remove noise [8][9][10]. The filtered frequencies can then be transformed back to the time-series domain.

**1.e Electrocardiographs and Alternative Sensors**

An alternative technology that is used for cardiovascular health and detecting blood flow throughout the body is called photoplethysmography (PPG). Fitbit pioneered this technology in their health watches and there has been a recent trend to also embed these sensors inside of mobile phones such as Samsung’s recent devices.  
  
PPG works by shining a light onto the skins surface (normally on the wrist or tip of a finger) and detecting the amount of light that has bounced back. A low-intensity infrared green light is normally used but this is not always the case. Changes in the diameter of capillaries under the skin which occur due to the blood volume passing through them affect the amount of light which is bounced back to the detector.   
Light is absorbed more by the blood than the tissue of the skin and capillaries, so this allows for a good estimation of volume flowing through the body which changes the voltage output of the PPG sensor [11].

PPG is seen as an alternative to ECG due to its reliability whilst a person is going about their daily lives. Whilst exercising, sweat does not interfere with the light reflected from the capillaries making the signal which is retrieved less noisy. Movement is also less of an issue than in electrocardiographs meaning which also improves the reliability.

PPG sensors do gather less information relating to cardiovascular health as the light reflected can only show Systolic and Diastolic blood pressures as the blood volume in the capillaries change compared to the PQRST sections which can be seen in an ECG [12].  
 **2 Project Design and Initial Documentation**

This chapter discusses the design choice for the software, the programming language and the ECG data collection method. A UML diagram is supplied to illustrate the initial design and the implementation of the Agile Scrum methodology is explained.

**2.a Python 3**

The Python programming language was used primarily for its extensive libraries to allow for data manipulation and mathematical calculation. Some of these libraries include SciPy to allow for peak detection, NumPy to allow for the implementation of FFT and Matplotlib which is a powerful tool used for the creation of graphs.  
Python tends to run faster than other programming languages when doing a lot of calculations. One of the reasons for this is because certain libraries used by Python are written in languages like C. NumPy is an example of a library that is mostly written in C.

**2.b Real-Time Data vs Pre-recorded Data**

Vital sign monitors in general practitioners and hospitals use a real-time implementation for ECG data which allows the medical staff to diagnose conditions instantaneously. Whilst this method is useful for a trained medical eye that can quickly decipher if there are any issue with the PQRST sequence, computers can take a while to get information from these signals depending on the amount of processing that needs to be done.

One of the aims of this project was to create graphs to show how the signal is processed over time which gives a better view of how signal processing algorithms work. Creating multiple graphs whilst iterating through live data is processor heavy and eventually the software would fall behind the live data.  
  
Initially, the idea was to use data that can be found and accessed online but the format of the data would change depending on the source of the data. If the data was in CSV form, some data would occasionally have been saved with tabs rather than commas meaning that the data was not saved as a proper CSV file but would have all the data inside of a single column instead.   
It was decided to use the University of Aberystwyth Sport Science departments new Welch Allyn Electrocardiograph instead.  
Furthermore it is out of the scope of this project to develop the relevant software to retrieve a live data signal from a system such as the Welch Allyn Electrocardiograph which was used.

**2.c Scrum**

A modified version of the scrum agile methodology was used throughout this project to keep track of a backlog of features that needed to be implemented. Instead of daily scrum meetings, there was a weekly supervisor meeting to check on the progress of the project.  
The project was broken down into week long stories (otherwise known as Sprints) on a Trello board and GitHub was used to save code for individual features on different branches.

**2.c.i Trello**

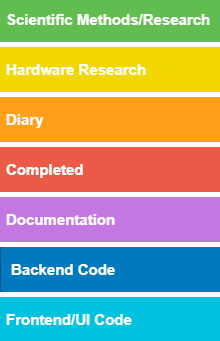
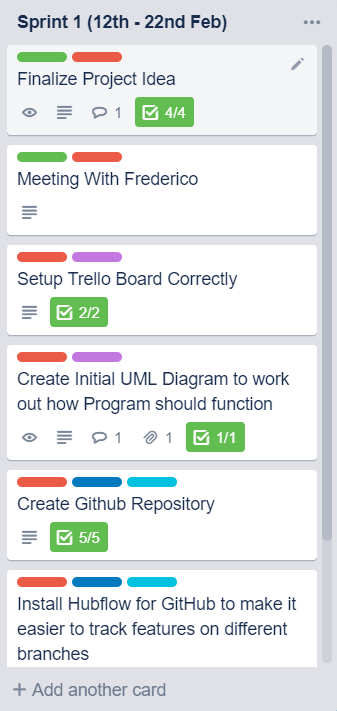
Trello is a web-based application that allows users to make different boards and add lists to the boards. Trello is not explicitly for project management but is a free, simplistic alternative to other services such as Jira.  
  
The project was broken down into 7 different categories which represent different parts. These categories were added to the features in the backlog so that they were visible as they had different colours associated with them [Figure 3].   
A few examples of categories would be Documentation, Hardware Research and Completed. This makes it easier to see what features relate to whilst quickly glancing over them. It was especially useful to be able to tell if a feature was completed by seeing a colour attached to a   
story [Figure 4].  
The known individual features were created at the start of the project and assigned a category. As the project   
progressed and more features were needed, they were   
created and added into the backlog.  
  
The backlog is another list of features that have not yet been implemented and have not been added to a story. This backlog was checked every week and updated.  
At the start of every week, the features that were needed next for the project to progress were placed from the backlog, into the story for that week.

Figure - A Sprint

Figure - The Projects Trello Categories

**2.c.ii GitHub**

The GitHub repository is composed of a Master branch and a Develop branch. When a new feature in the story is started in Trello, a new branch is created from develop with the feature name from Trello. An example of a feature name is “Feature 6 - Get Average of 8 ECG Leads”.  
  
Commits to the current branch have comments associated with them such as “Averaged the 8 ECG leads to create a single signal”. This helps to keep track of the updates that have happened throughout the project which may help if a bug occurs and a previous version of code is needed again.  
  
A pull request was created for each individual branch that explained what was added to the codebase. This code was not peer reviewed externally but by me. The changes to the code was checked thoroughly to make sure that no errors would get merged into Develop.  
  
When the feature was finally finished, the pull request was reviewed and after all conflicts were resolved, the code was merged back into the develop branch and the feature branch was deleted.  
Multiple feature branches were not used consecutively which tends to be the case if multiple people are working on a project.

GitHub enabled bugs to be found faster and a backup of the codebase in case of a hardware issue or bug.

**2.d Initial UML Diagram**

It is normally best to start a project with an idea of the direction that it will be heading in. A UML diagram was created in the first week of the project as a guideline. This UML diagram was created before the further stages of research had begun so was a guideline. It is also important to note that this diagram was created before it was known that the Welch Allyn electrocardiograph would be used.  
  
The diagram below [Figure 5] shows a main class and the relevant functions which are needed to access ECG data, iterate through and process the data to calculate the heart rate and finally show the data to the user.  
The initial idea is listed as 7 main steps in the program:

1. detectFileType() - Checks if the user has selected a JSON file or a CSV file by detecting if the file ended with .CSV or .JSON. The file is then automatically sent to the correct function to parse the file. Note – This idea did was attempted successfully with ECG data found online but was not included in the project due to the data formats that the Welch Allyn electrocardiograph can produce.
2. iterateThroughData() – This would take all of the data which has been parsed into an array and pushes this data on towards the detectSingleBeat() function.
3. detectSingleBeat() – This shall take the data and segments the data into a 2D array containing the different heart beats and their corresponding data points.
4. findPqrstSections() – This utilizes the separated individual heart beat data and calculates the PQRST sections which are needed for further analysis.
5. calculateHeartrate() – This would take the R Wave data for each individual beat and calculates how many of these there are inside of a minute which gives the heart rate as beats per minute.
6. classifyHeartrate() - This would use the PQRST sections which were detected in findPqrstSections() and calculate the timings in-between them which would give an idea of health.
7. Heart Rate Display – A display to show the heart rate and classification found from the PQRST sections.

The design altered as the project progressed from this early prototype UML diagram and changed in scope due to better ideas. Most notably, the user interface does not display any data and is used as a file explorer in the final version of the software.

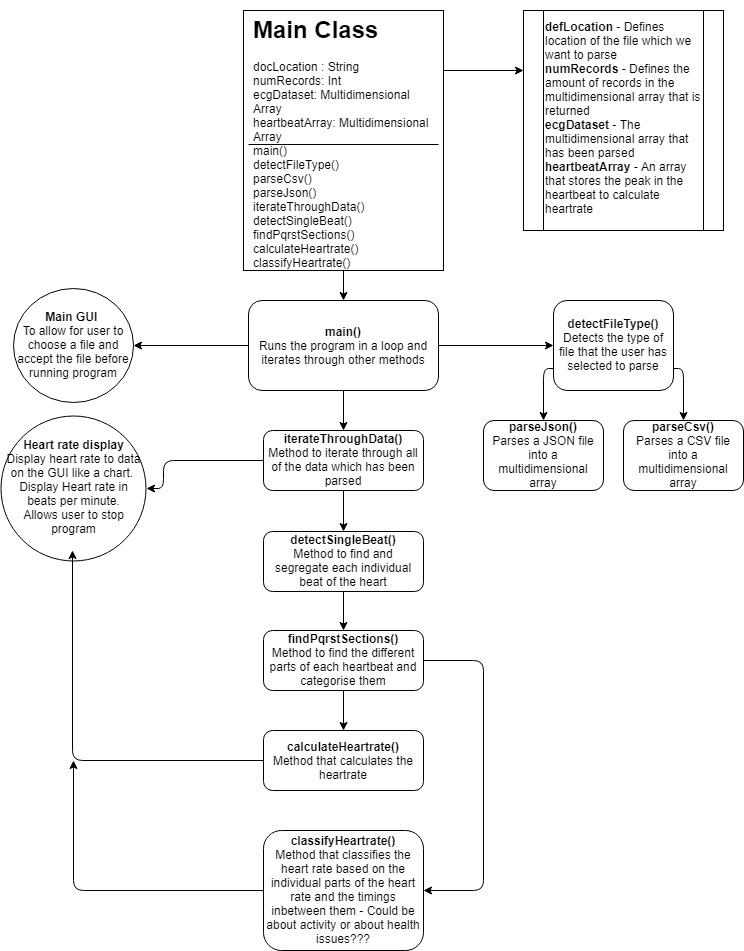


Figure - Initial UML Diagram

**3 Implementation**

This chapter discusses the actual implementation of the project. This includes discussion about data acquisition, algorithm’s, code, issues that were found during the process, changes that were made to the original plan and the final UML diagram. TODO

**3.a Recording the Electrocardiogram**

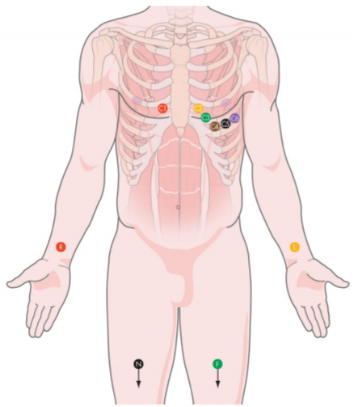
As mentioned previously in chapter 3, data was initially sourced from webpages such as Kaggle and CSV representations saved from the MIT-BIH arrhythmia datasets [13].  
The datasets mentioned above have no uniformed order to them so in order to acquire the data the new Welch Allyn CardioPerfect electrocardiograph was used to get readings for my personal heartbeat.   
Unfortunately, there was no technician available to get more data from other participants to validate the software.   
  
The CardioPerfect WorkStation software by Welch Allyn showed the ECG to the user after it was recorded and gave some information that it could calculate from the reading. This information includes heartrate and an idea of cardiac health and arrhythmias.  
This software is not free to download but and the University of Aberystwyth owns a license. This meant that the software could not be used from home.  
  
The 10 individual readings that were taken using the electrocardiograph took an hour and a half to obtain. In total there are 6 readings of me lying down and 5 readings of me sitting up. Initially it was believed that we would be able to take a reading whilst exercising, however it was quickly discovered that a slight movement introduces a lot of noise into the signal. Unfortunately, the system does not have the available filters to remove this noise.  
  
The electrocardiograph has 10 leads which were placed on my body after applying a coat of electrical gel [Figure 6]. These leads prohibited any movement of the arms, legs and chest whilst readings were taken. The leads on the legs are used as grounds for the system whilst the chest leads get measurements from the heart.  
  
The 10 second readings took so long to obtain due to the amount of times that we needed to restart the readings due to noise. If I breathed too heavily, if a cable was moved, or if pressure was accidently put onto an electrode, the signal showing in the software would change dramatically manifesting one or all the main issues obtaining data from an electrocardiograph – Baseline wander, powerline interference and electromyographic noise.

Figure - ECG Lead Placement [14]

Perspiration was an issue throughout due to the stress of a clinical environment which lead to a baseline wander on a few of the readings. These readings are included in the dataset submitted for this project and named “Jake Newall Lying Down – 3” and “Jake Newall Sitting Up – 1” through to “Jake Newall Sitting Up – 5”.   
There were initially supposed to be 6 readings of me sitting up but sitting completely still for an hour and a half made me stressed and fidgety making the final reading impossible to take.

**3.a.i Data Formatting Issues**

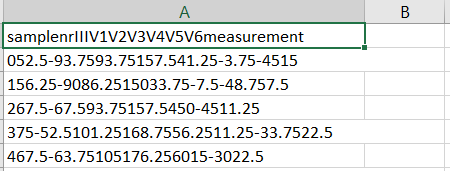
Unfortunately, it was discovered that the CardioPerfect WorkStation software could only export files in the 3 following formats: .SCP, .SDF and .MDW.   
After attempting a multitude of different ways to open the file types, the decision was made to load the .SCP files into a second program which allowed for .CSV files to be exported. This piece of software is free to download and is called ECG Toolkit 2.4 [15].   
  
ECG Toolkit 2.4 is the same as the CardioPerfect Workstation in terms of design and what it can do. The only reason that this software is used is for retrieving CSV data from the SCP files.   
  
The software also displays the heartrate which is calculated which was useful for testing the finished software against.

Figure - Exported CSV Data from ECG Toolkit 2.4

Like some of the testing data which was found on Kaggle, the ECG Toolkit 2.4 software did not separate the signal correctly and used tabs instead of commas [Figure 7]. This required some further processing which is discussed in 4.c.

**3.b Simple User Interface**

PyQt5 is a Python binding of the graphical user interface toolkit known as Qt. PyQt5 acts as a Python library that enables users to create simplistic user interfaces with buttons and text boxes.

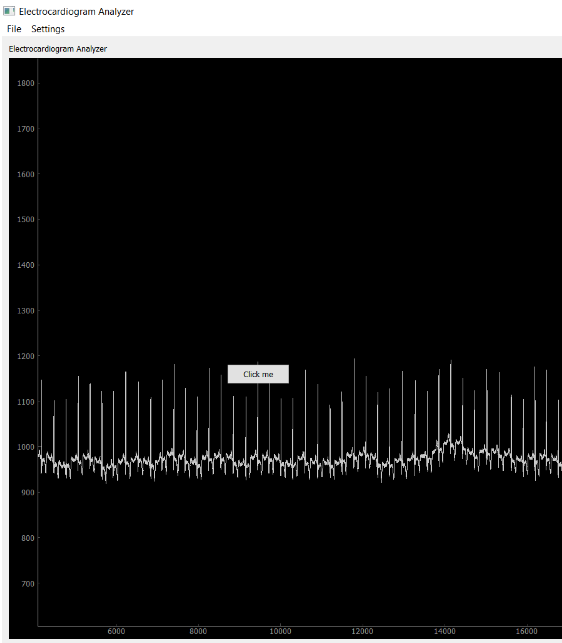
After attempting to use PyQt5 to make a user interface that would allow for graphs to be incorporated using Matplotlib in an earlier build [Figure 8], it was quickly decided that the user interface would be simplistic in design.   
  
It was found that embedding a plot into the window made it incredibly difficult to update the graph. This issue took a lot of time to attempt to fix, sadly to no avail.   
Attempting to process multiple files and show all of the graphs created inside of a user interface would be difficult.  
Because this project is about signal manipulation, the decision was made to save graphs to folders on the computers local hard drive instead.

Figure - Early Prototype to Display ECG Signal in PyQt

The scaled back approach meant allowing the user to only open folders with the user interface which leaves the interface with simple instructions only. The user interface simply functions as a file explorer that allows the user to select a folder which contains CSV files.  
  
 **3.c Importing and Exporting CSV Files**

After the user has chosen the folder that contains the CSV ECG data, the software iterates through all the files contained within and checks if these files end with .CSV. It was expected at the start of this project that this would need to be implemented (Shown in Figure 5 in section 3.d).  
   
For each individual file, the software calls the next function processCsvData() and passes the folder location and filename as a parameter. If the file does not end with .CSV, the file is ignored.

ProcessCsvData() (Line 121 of ECG Analyzer Main.py) is the main function in the program. It imports individual files and calls the necessary functions which process and return the data to help find the PQRST sequence.  
The data must first be imported so the inbuilt open() function is called with the folder location and filename variables as parameters.

The data is iterated through and the tabs (\t) are removed from the data and replaced with commas for exporting the data in a better format. Whilst the tabs are being removed, the rows are being added up to make the averaged signal which is made up out of the other 8 signals which are found inside of the CSV files.

If a newline (\n) is found, the previously found signals for that row are averaged using the createAverageSignal() method found in signalProcessing.py. If the character ‘s’ is found however, the line is ignored because the first line in the CSV data relates to the naming of the leads and not actual data.

Finally, the data has been correctly formatted for easier use and the averaged signals are added to the multidimensional array of data. This data is saved to a file in the correct CSV format to make up for the shortcomings of the ECG Toolkit 2.4 software and to make it easier to read. In order to save the data, a saveToCsv() method was created inside of the signalProcessing.py file under the Util class.

**3.d Averaging the Signals**

As mentioned in 4.c, processCsvData() calls the createAverageSignal() method and passes the 8 signals in that row. This method simply takes the data and finds the mean before returning a float back to the processCsvData() function where it is appended to an array.   
Eventually, all 8 leads and the 6,000 readings for each of them are saved as averaged signals ready to be processed further.

It is important to average the 8 ECG leads into an averaged signal. After testing the signals individually, I found that certain electrode leads had severe noise caused by powerline interference and muscular electricity making them almost impossible to process. The leads that caused the noise changed for each individual reading which also made it harder to filter out an individual lead.  
  
When the average of the 8 ECG leads was taken, the singular output was compared to the individual leads and the noise was greatly decreased giving a cleaner signal to process.

**3.e Calculating the Mean for the Entire Averaged Dataset**

To illustrate the difference between the mean of the data and the running mean calculated every 400 samples due to baseline drift, the average for all the averaged ECG signals is calculated. This mean is used throughout the software when creating graphs to show the progression of the data due to each step of processing.   
  
ProcessCsvData() calls calculateMeanOfData() which is located in signalProcessing.py and passes the array containing the averaged data through.  
The calculation is simple and after it has been done, an array is formed containing 6,000 copies of the mean so that it can be plotted in Matplotlib before returning the data back to processCsvData().

**3.f Calculating the Running Mean Datapoints**

Baseline drift is an issue which needs to be addressed and filtered when recording ECG’s. One means of doing this is to calculate the cubic spline interpolation of the running mean which gives us a mean curve that changes over time, effectively showing the baseline drift [Figure 9].

ProcessCsvData() takes the averaged ECG data and passes it to the calculateRunningMean() method found on line 46 in averageLeads.py.   
A number is also passed which is used as the number of samples to calculate the mean for before moving on to the next ‘n’ amount of sample to process.  
This is due to the fact that this process is later repeated in order to remove noise from the signal but with a much smaller number of samples.  
  
In this instance, it was found that 400 samples gives optimal results for calculating the running mean. The Welch Allyn electrocardiograph records 6,000 samples every 10 seconds which equates to 36,000 samples every minute.  
If a person has a heartrate of 60 beats per minute (which is what is considered a healthy resting rate), then the ECG would typically pick up 10 of the beats. This would put the sample to heartrate ratio at 600 samples per heartbeat. The lower the samples (to a point), the more reliable the curve which we can find.   
  
Ideally, the R peaks could be detected and the distances between them calculated in time so that the sample variable is based on individual beats rather than the lowest beat. This would scale, allowing data that shows a heartrate of 200 beats per minute to have the running mean be calculated on 180 samples instead.  
Unfortunately, it is unreliable to try to find the R-Peaks without the removal of baseline drift due to the change in amplitude of the signal. One way to resolve this issue would be to make a second pass of the original data after the drift has been removed in order to get a more reliable results.

**3.g Calculating the Cubic Spline Interpolation of the Running Mean**Cubic spline interpolation takes a set of datapoints and fits a curve to the data. In order to do this, it fills in gaps between the datapoints with more points whilst adjusting to create a curve, rather than a linear fit [Figure 9][10].  
  
After processCsvData() has obtained the running mean datapoints, this array is passed to the calculateInterpolation() method inside of averageLeads.py along with the averaged ECG data.  
The SciPy interpolate library is used for the cubic spline interpolation and returns the interpolation function. The interpolation function is needed to create the new set of datapoints within the range of the original datapoints.  
  
It is noticeable in figure 9 that there is a baseline drift in the data.

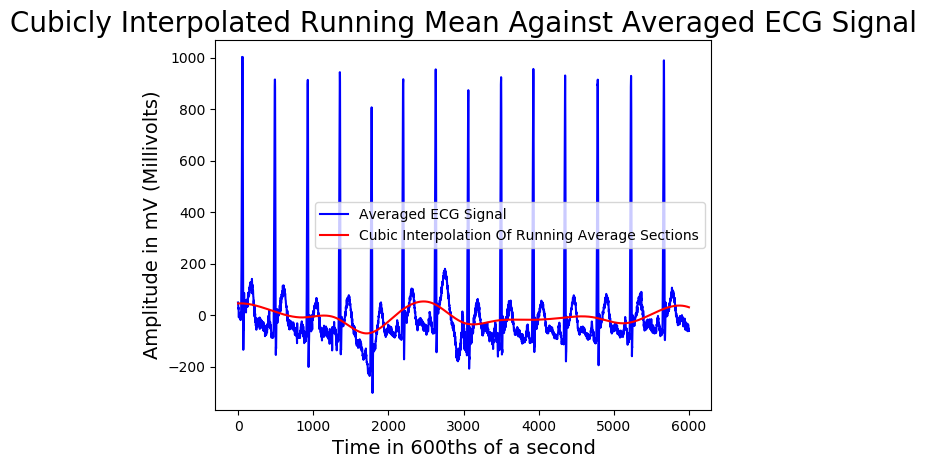


Figure - Averaged ECG Signal and the Cubically Interpolated Running Mean

Using NumPy’s arrange function, the x values are created for use with the interpolation function. The variables which are passed to the arrange function are dependent on the sample rate which was used to calculate the running mean.   
If 400 datapoints were used, then there will be 15 variables in the running mean array. In this instance, the values passed to the arrange function will be 1, 15 and 0.0023. This will create evenly spaced numbers that increment by 0.0023 between 1 and 15. This is important as if we wish to make the cubic spline interpolation as accurate as possible, we want the end array which is calculated to have the same amount of datapoints as the sample rate (in this case, 400).  
To calculate this we simply take the length of the array -1 and divide by the sample rate which is 6000.

**3.h Removing Baseline Drift**

The method which was chosen to remove the baseline drift is to calculate the difference between each variable in the newly processed running mean dataset and 0. The values in the averaged ECG data array which corresponds to each index of the running mean dataset then have these differences negated from them.  
This processes essentially flattens both the running mean dataset and the averaged ECG dataset which attempts to bring the mean of the data closer to 0.  
  
The removeDrift() method inside averageLeads.py is called from the processCsvData() function. The data is iterated through and the above process is carried out. Finally, the new mean of data is calculated to show how the mean has changed from the unprocessed data to the processed data.  
  
This process works well for some of the CSV files. Other files unfortunately have a lot of drift due to certain leads having too much noise which affects the averaged signal. These files still have most of the drift negated and all the files which were tested produced results which were good enough to allow for further processing.   
  
A graph is created at the end of the function which plots the ECG data without drift removed, the ECG data with drift removed, the mean of the data before drift removal and the mean of the data after drift removal.

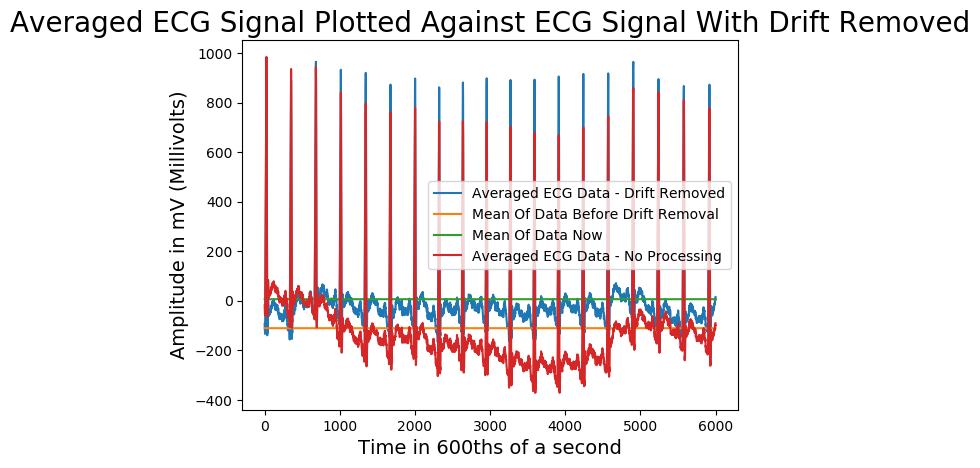
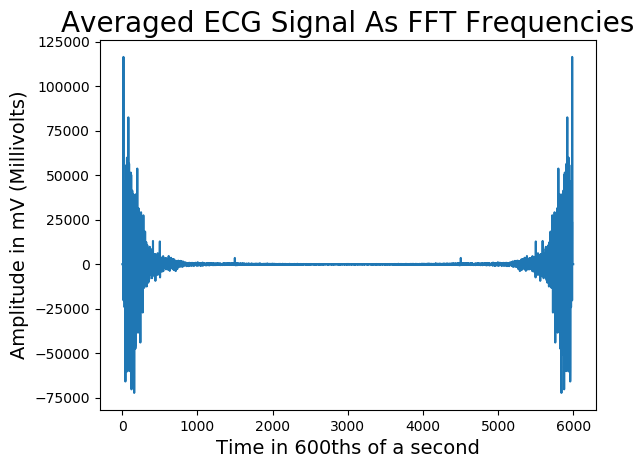


Figure - ECG Data Drift Removed Plotted Against ECG Data

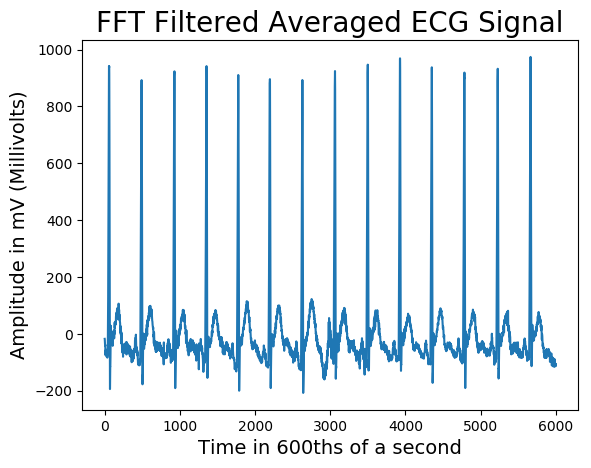
It is clear from the data displayed in Figure 10, that the proposed method to remove baseline drift is a success. This was the worst signal recorded and the change is dramatic.  
The initial signal has a mean of data that lies around the -120 mark whereas the processed data has a mean of 0.  
   
The mean sits roughly around the P and T sections throughout the signal apart from the last 1,400 datapoints. This is due to the curve which was fit during the calculation of the cubic spline interpolation.  
  
One way to avoid this issue (as mentioned before) is to attempt to eliminate baseline drift when recording the signal. There is a possibility that if the signal was processed again using the same method, the data may have a better fit around the mean.

**3.i Removing Noise using Fast Fourier Transform**

FFT allows for a signal to be converted into the frequency domain. The frequencies can then be filtered and changed back to the time-series domain in order to remove noise [10].

There is a block of code in signalProcessing.py from lines 305-316. These lines were found on the internet [16].  
Initially, this block was used for debugging and testing the library. After attempting to change this block with different filtration algorithms, the signal which was produced got worse.  
It was because of this that the lines of code which are in place are not my own work.  
  
The filtering method takes the signal and converts it into the frequency domain [Figure 11].

Figure - ECG Signal Converted To Frequency Domain

The frequencies then undergo a process which removes the negative values from the array. The array is then converted back into time-series data [Figure 12].

This process tends to make valleys deeper and peaks taller. This also means that the features with are found at the peaks (PRT-Waves) and the features which are found in the valleys (QS-Waves) tend to be easier to find.  
  
The amount of noise in the signal is not reduced dramatically.   
This means that further processing must be done to further separate the PQRST sections from the noise.

Figure - ECG Signal With FFT Filtering

**3.j Smoothing the Signal**

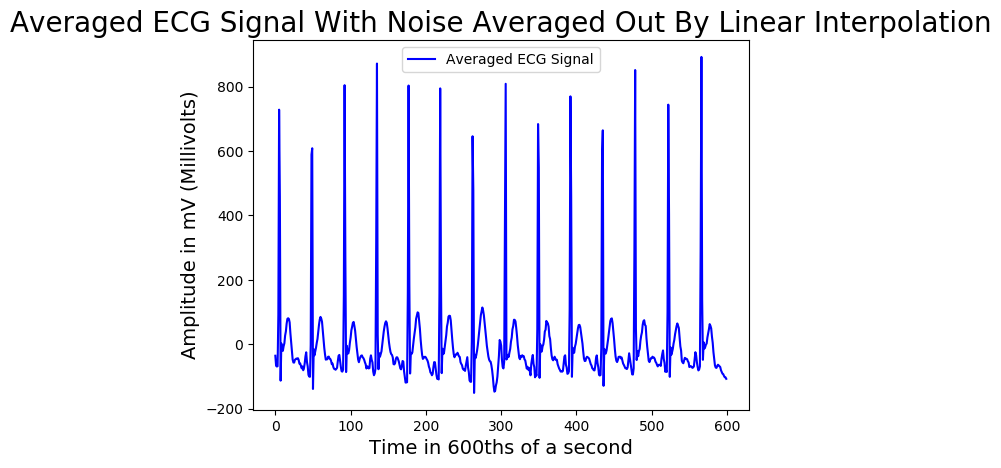
Peak detection can be difficult if there is a lot of noise in the signal. FFT filtration has made the peaks and valleys more prominent and reduced some of the noise. To reduce the rest, the calculateRunningMean() and calculateInterpolation() functions mentioned in 4.f and 4.g are used again. If the moving average is calculated on a small number of signals (such as 8 or 10) compared to the much larger number used to remove the baseline drift, it is possible to average out noise which in turn makes the signal smoother.  
  
ProcessCsvData takes the ECG data with the baseline drift removed and repeats the steps mentioned in 4.f and 4.g but with the algorithm set up so that the average is calculated on 10 datapoints instead.   
  
Initially, 2 datapoints were used in this process but unfortunately, the data still contained a lot of noise.  
10 was found to be the most reliable number of datapoints to average whilst saving the PQRST sections found within the signal.   
If any more than 14 datapoints are used, the signal starts to lose important information. Most notably the P wave becomes less pronounced and as more and more datapoints are used, the T wave disappears, followed by the Q and S waves. The signal becomes less and less noticeable until the point where it start to look like the moving average curve shown in 4.g, Figure 9.  
  
  
If the data shown in Figure 13 is compared to Figure 12, you can see how this process has changed the data. The overall signal has stayed roughly the same, but the noise caused by the electricity of the muscles picked up by the electrodes has been averaged out.  
  
  
  
  
The PQRST sections are still in the same position as they were originally and are more defined. The amplitude of the signals has changed as a result of the smoothing process, but this is only noticeable on the QRS wave of the ECG.

Figure - ECG Data Noise Removed

**3.k Finding the R-Wave Peak**

With most of the noise eliminated from the averaged ECG data, features can finally be extracted from the signal. The first feature that the software looks for is the R Wave. This wave typically stands at an amplitude between 600-1000 mV making it the most prominent and easiest to find.

The SciPy library has a function inside of the signal class called “find\_peaks” which “takes a one-dimensional array and finds all local maxima by simple comparison of neighbouring values” [17].

After ProcessCsvData() has received the smoothed signal, the ECG data is passed to findHeartrate()inside of the PeakValleyDetection class from signalProcessing.py.  
The Scipy.signal.find\_peaks() function is utilised with the ECG data array passed as the first parameter and the minimum height required for peaks and the distance between peaks passed as the second and third parameters.  
  
The minimum height of an R Peak in the 10 datasets provided has an amplitude of approximately 520 mV and can be found in the “Jake Newall Sitting Up 2” CSV file at beat 6 when the software is ran.   
This is caused due to the smoothing mentioned in 4.i which decreased the R Peak amplitude and subsequently made the S wave deeper allowing for it to drop to -200 mV compared to some of the other beats which had an S wave depth of around -100 mV.  
   
After looking at the different CSV files and corresponding graphs, it was decided that it would be best to set the minimum height required for R Peak detection to be 400.  
This is 80 mV less than the lowest R-Peak mentioned above and 230 mV higher than the next highest data point which is not related to the R Peak. This datapoint can be found in the “Jake Newall Lying Down” CSV file at beat 7 in the peak of the T Wave in the graphs included.

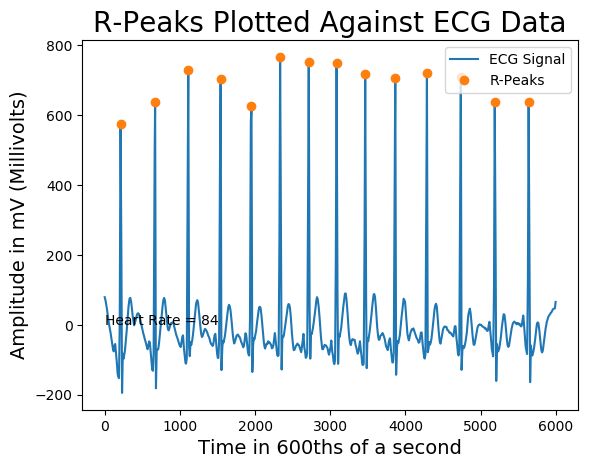
The distance between peaks is set to 200. This acts as a filter which will remove any extra peaks which could have been created by noise if they are closer than 200 datapoints.  
Ideally this value would be set by the number of beats are in the dataset, but this value has not yet been found.   
  
The scipy.signal.find\_peaks function returns an array containing the indices of peaks found and a dictionary containing the amplitude of the peaks.  
The heartrate of the individual is calculated by taking the number of peaks found and multiplying this figure by 6 which gives us beats per minute. This heartrate corresponds to different zones which identify the type of exercise that the individual is doing.   
  
A function called “calculateHeartRateZones()” is called from the Util class and takes the heartrate as a parameter. After researching different zones, the image shown in Figure 14 was found. The ECG data was collected from myself (a 25-year-old) so the zones are calculated from that age group.   
The corresponding zone is returned back to findHeartrate().  
The heartrate and associated zone are saved to a separate CSV file by passing an array with these values to the saveToCsv() method inside of the Util class.   
This method attempts to make a new folder called “Results” if the folder does not already exist and then uses the Numpy.savetxt() function with a comma as a delimiter to save the file.  
The graph shown in Figure 15 is then created to show the R Peaks that were detected against the ECG signal.

Figure - Heartrate Zones [18]

Figure - R-Peaks Plotted Against the Averaged ECG Data

**3.l Splitting the Signal into Separate Beats**

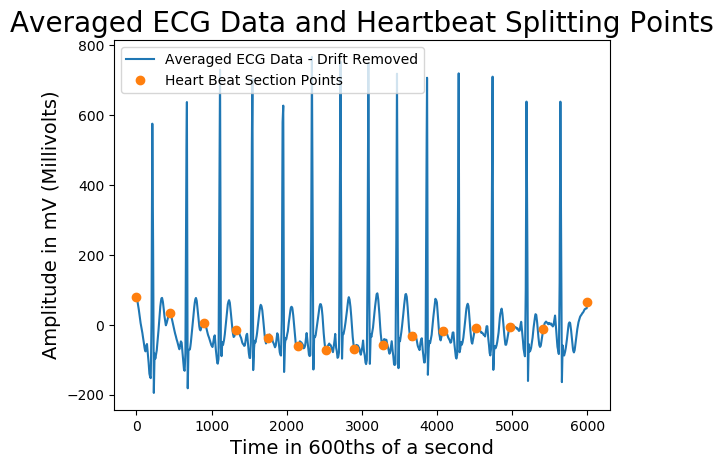
Now that the R Peaks have been found, the signal can be split into individual beats.  
The findIndividualBeats() inside of the class Util in signalProcessing.py is called from processCsvData(). The ECG data and R-Peaks are passed as parameters.  
  
The function finds the midway point between R-Peaks and uses that as a basis of where the beats start and where the beats end.  
If the function is looking for the first beats splitting point, the start of the array is used. If the function is looking for the last beats splitting point, the end of the array is used.  
This creates the splits that can be found in Figure 16.

Figure - ECG Split into Individual Beats

The individual beat cuts are saved to an array which is returned back to processCsvData().

**3.m Finding QS Valleys**

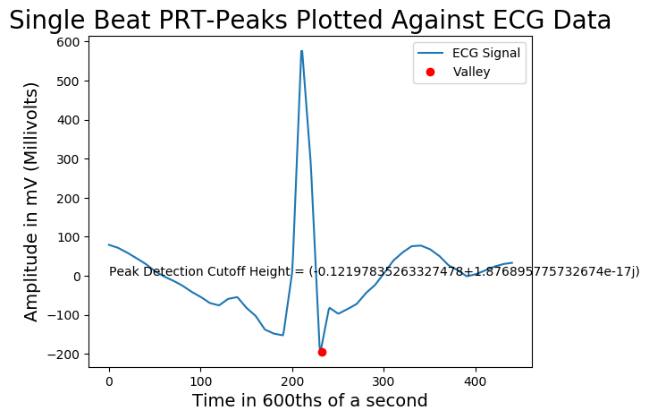
To find the QS Valleys, processCsvData() calls the findPeaks method in the class PeakValleyDetection in signalProcessing.py The ECG data is passed along with the individual beat cuts mentioned in 4.l.  
  
This method splits the ECG signal into the individual beats and iterates through the data. First, signal.find\_peaks() is called to calculate as many peaks as possible.  
When the R-Peak has been found, we can calculate the QS Valleys which lie underneath.  
  
The S Valley is calculated first. The array containing the data for the individual heartbeat is cut again, this time from the R-Peak until the end of the data. This array is then sent to PeakValleyDetection.findValleys().  
  
FindValleys() takes the mean of the data as a checker variable to see if the data is below and not above. The data is then iterated through from the top of the R-Peak until it gets to the bottom.  
Whilst this iteration is happening, if the datapoint is below the checker variable then this variable becomes the datapoint.   
Eventually, when a datapoint is above the checker variable, the S Valley has been found and is returned to findPeaks(). The individual heartbeat is then saved as a graph with the S Valley marked on top of it [Figure 17].

Figure - Single Heartbeat and S-Valley

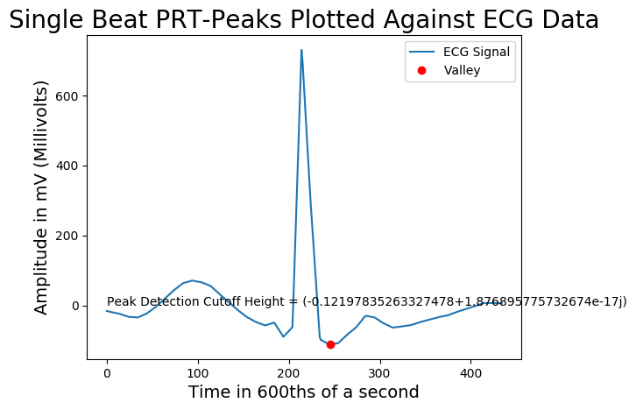
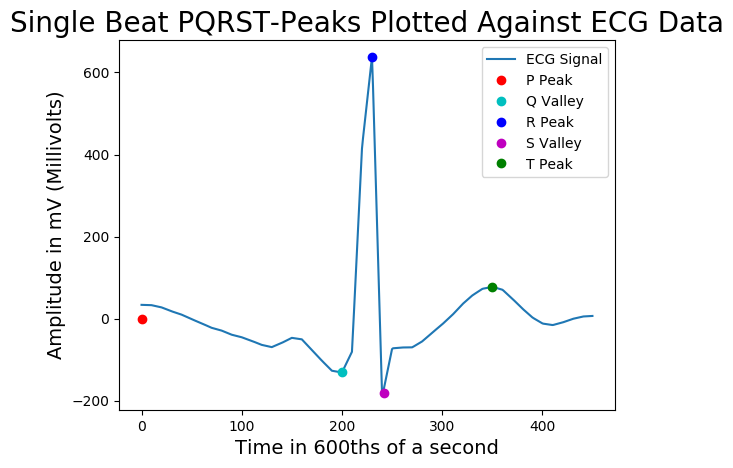
The same is done for the Q Valley with a slight difference. To make sure that the same function can be utilized again, the array is reversed because the function finds valleys in front of the R-Peak.   
To ensure that the index of the Q Valley is correct, the “found index” is removed from the length of the array containing the heartbeat giving the correct index. This can be seen in Figure 18.

Figure - Single Heartbeat and Q-Valley

**3.n Finding PRS Peaks and Saving the Values**

The findPeaks() method mentioned in 4.m also looks for the PT Peaks. The method iterates through each heartbeat and its corresponding data and does the calculations mentioned in here and 4.m in each loop.  
  
The method has rules for what can be a P-Peak and what can be a T-Peak. P-Peaks need to be before R-Peaks and can only be found if the P-Peak for that beat has not been found. T-Peaks need to be after the R-Peaks and can only be found if the T-Peak for that beat has not been found.  
There is also a limit of height set on both the P and T Peaks mentioned in 4.k. The closest P or T Peak to the height limit of 400 is 230mV. The R-Peak must be above this limit and the P and T Peaks must be below this limit.  
  
With these simple rules in place for the P and T Peaks, they are found inside of the array returned by scipy.signal.find\_peaks. There are exceptions to this which are explained in (TODO) Results.

A graph is saved for each iteration of findPeaks() that shows the PQRST segments that have been found plotted against the ECG data [Figure 19]. The P-Peak at 0,0 has not been found.

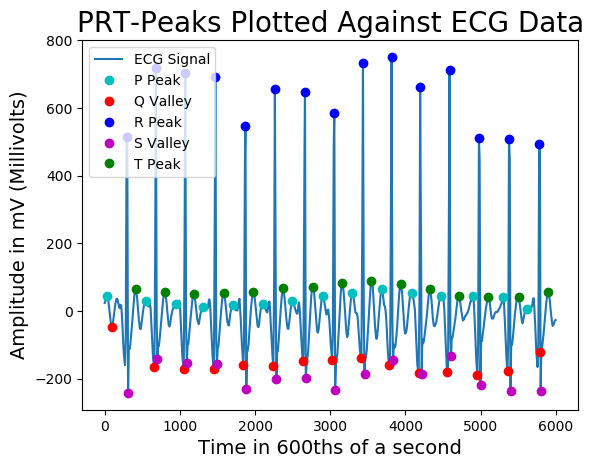
  
  
After all the iterations through the heartbeats have finished, we are left with 5 arrays containing the PQRST Sections and 5 arrays containing their index’s ready to be plotted to a graph and saved as a CSV file. One final graph is created showing the output of the software [Figure 20].

Figure - PQRST Sections Plotted on ECG Data

Figure - PQRST Sections Plotted On Complete ECG Signal

Finally, the data is saved to a CSV file. Util.pqrstToSeconds() is called from the bottom of findPeaks(), passing the ECG data, PQRST index arrays and PQRST datapoints.

PqrstToSeconds() initializes a 2D array and iterates through the data that was passed through. The data is converted from 600ths of a second to seconds. The data is then added to the 2D array in a specific format so that it is easier to read when opening the CSV file in Excel.  
The number of each of the PQRST sections that have been found for that dataset are tallied and added underneath the individual readings in seconds.

The Util.saveToCsv() method is then called which saves the PQRST data to the “Results” folder so that it can be accessed.

**3.o Final UML diagram**

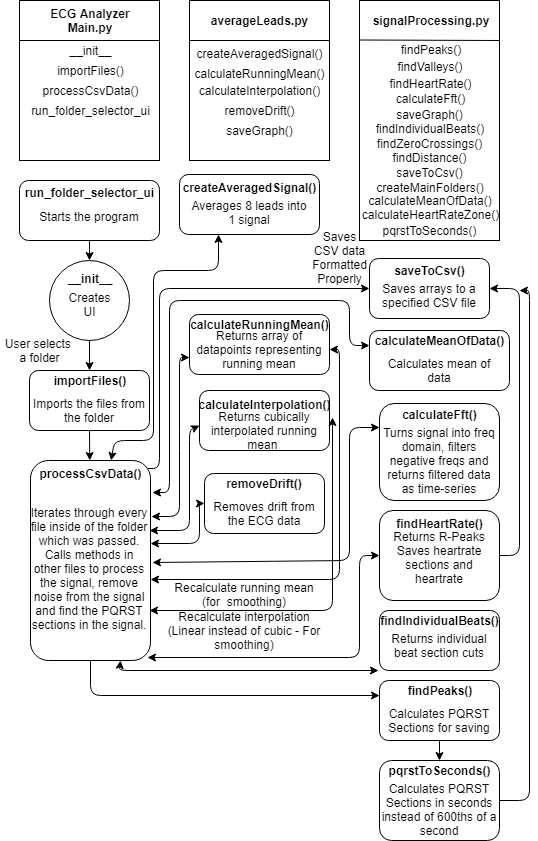
The diagram shown in Figure 21 is noticeably different from the original UML diagram shown in Figure 5.   
As mentioned when mentioning Trello in 3.c.i, unknowns get added to the backlog.  
  
Certain parts from the diagram have been left out to save space and stop confusion such as saveGraph().  
  
The flow for processCsvData() follows the arrows down. The arrows closer to the top are called first.  
  
The functions depicted are directly below the file that they belong to.  
  
The first function called is the box below ECG Analyzer Main.py and is called run\_folder\_selector\_ui.  
  
The last function called is on the bottom right and is called pqrstToSeconds()

Figure - Final UML Diagram

**4. Results**

**4.a ECG Toolkit 2.4 CSV Dataset vs Sanitised ECG Dataset**

Due to ECG Toolkit 2.4 being the only freeware that was available, the CSV data that is used for this program is initially bad. The data is separated by tabs rather than commas when exported leading to the data seen in Figure 22.  
As the data needs to be sanitized to be used in this software, the output is saved in a better format [Figure 23]. The averaged signal is also appended onto the end of the new dataset. This is useful if another program needs to use the PQRST sections saved in another file with original ECG data.

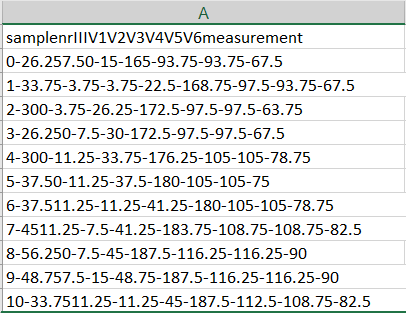
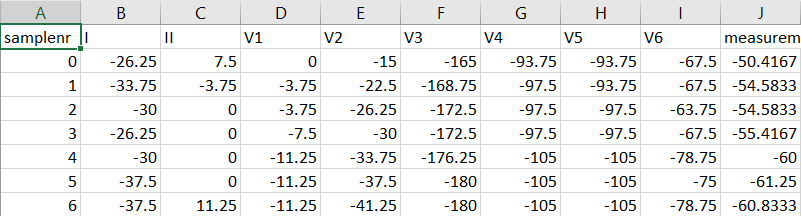


Figure 23 - Sanitised ECG Data

Figure 22 - ECG Toolkit 2.4 CSV Data

**4.b FFT vs Non FFT PQRST Sections**

**Number of PQRST sections found for each file – No FFT**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | LD1 | LD2 | LD3 | LD4 | LD5 | LD6 | SU1 | SU2 | SU3 | SU4 | SU5 |
| P | 1 | 1 | 1 | 1 | 0 | 2 | 4 | 0 | 0 | 0 | 0 |
| Q | 14 | 15 | 14 | 14 | 15 | 13 | 19 | 15 | 15 | 15 | 15 |
| R | 14 | 15 | 14 | 14 | 15 | 13 | 19 | 15 | 15 | 15 | 15 |
| S | 14 | 15 | 14 | 14 | 14 | 13 | 19 | 15 | 15 | 15 | 15 |
| T | 14 | 15 | 14 | 14 | 14 | 13 | 11 | 15 | 15 | 15 | 15 |

Where LD = Laying Down and SU = Sitting Up.  
*Table 1 – PQRST Sections Found in each File (No FFT)*

**Number of PQRST sections found for each file – FFT**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | LD1 | LD2 | LD3 | LD4 | LD5 | LD6 | SU1 | SU2 | SU3 | SU4 | SU5 |
| P | 3 | 15 | 1 | 11 | 0 | 13 | 16 | 0 | 4 | 1 | 4 |
| Q | 14 | 15 | 14 | 14 | 15 | 13 | 19 | 15 | 15 | 15 | 15 |
| R | 14 | 15 | 14 | 14 | 15 | 13 | 19 | 15 | 15 | 15 | 15 |
| S | 14 | 15 | 14 | 14 | 14 | 13 | 19 | 15 | 15 | 15 | 15 |
| T | 14 | 15 | 14 | 14 | 14 | 13 | 17 | 15 | 15 | 15 | 15 |

Where LD = Laying Down and SU = Sitting Up.  
*Table 2 – PQRST Sections Found in each File (FFT Filtered)*

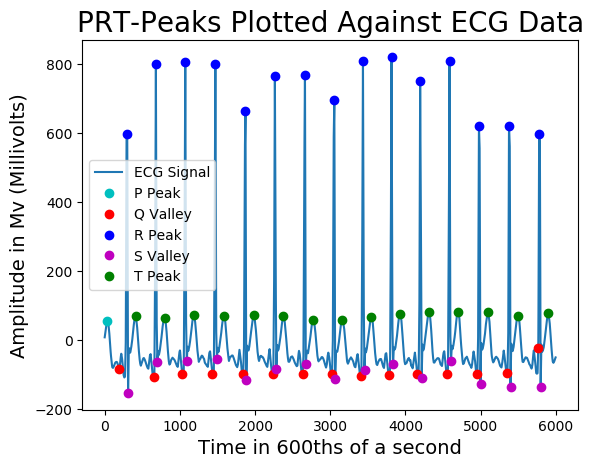
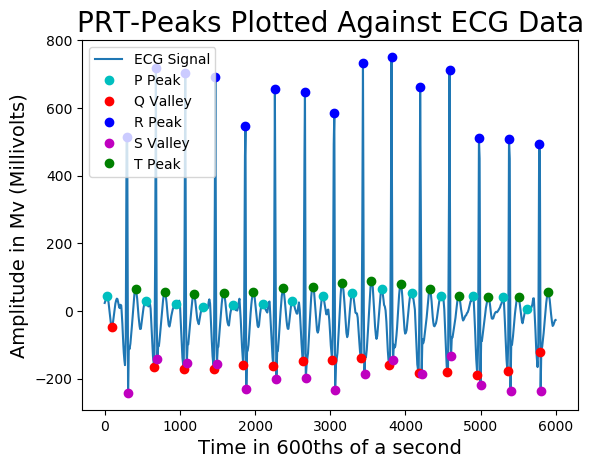
The calculateFft() function is allowed as default inside of ECG Analyzer Main.py. To change this back to validate the above results, line 202 - calculateRunningMean() needs to have averaged\_ecg\_drift\_removed passed as a parameter instead of filtered\_ecg\_signal. The calculateFft() function can then be commented out.  
  
The results for the code ran with FFT showed an improvement when calculating the P-Wave, however caused the overall quality of the data to be reduced for most instances of the files which were processed.   
This led to a small number of PQRST sections being diagnosed incorrectly because of the degradation of the signal.   
One of the reasons that this may be is due to the removal of negative frequencies from the signal. This removes additional frequency noise from the signal.  
There is the possibility of changing how the frequencies are filtered to create better results, but unfortunately any changes attempted caused further data loss from the signal.  
  
Valleys are also noticeable deeper for most of the signals when using an FFT filtering method compared to when the filtering method is not used. This can be seen in Figure 25 where the valleys are much deeper compared to Figure 24. This shows that FFT filtering changes the signal in a significant way.  
The definition of the peaks and valleys when using FFT filtering makes it easier to find the PQRST sections but may also slightly change the location of these sections. This tends to happen more on the P and T waves. Again, this correlates to the idea that using FFT filtering can degrade the overall signal even though more sections can be found.  


Figure 25 - Laying Down 2 Final Data (FFT Filtered)

Figure 24 - Laying Down 2 Final Data (No FFT)

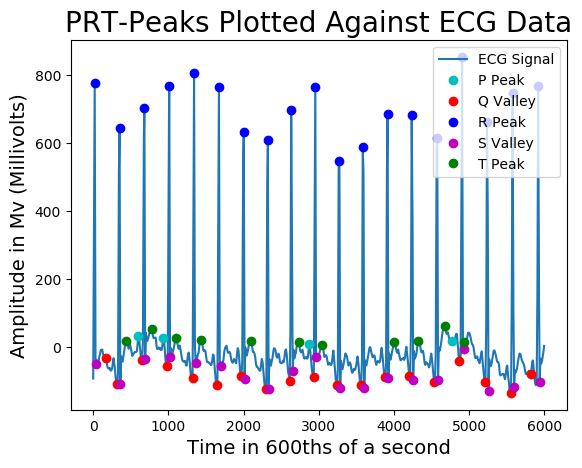
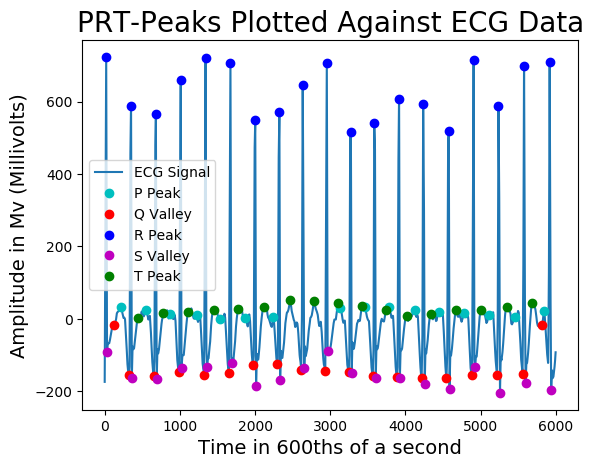
For the datasets that were tested, using FFT does not lose any of the QRST sections of the signal and captures them all. In the results for SU1 [Table 1] [Table 2], it can be noted that 6 extra T-Waves are found. This can be seen in Figure 27 (on the right) and Figure 26 (on the left) below.  
SU1 is the worst signal that is processed due to the amount of baseline drift of the input data.

Figure 27 - Sitting Up 1 Final Data (FFT Filtered)

Figure 26 - Sitting Up 1 Final Data (No FFT)

Looking at Figure 27, it is also noticeable that additional baseline drift has been removed from the signal compared to Figure 26.  
  
The findings and final graphs for the code ran with FFT can be found inside of the attached folder “FFT Results” whilst the results for the code ran without FFT can be found inside of the “Results – No FFT” folder.

**4.c FFT Filtered PQRST Time Intervals**

The PQRST sections are saved in seconds to excel spreadsheets in the “Results” folder. These readings have great significance when studying ECG data as the timing between these signals can give an insight into the health of a person.   
  
Looking at Figure 28 below, it is noticeable that there are fewer P-Wave readings compared to the QRST waves. In this instance, the peak finding algorithm did not detect the P-Waves due to noise in the signal which flattened the peaks. Occasionally, hearts can skip certain Waves completely due to cardiac health issues.  
  
As the list descends, the time since the start of the ECG dataset increases. For a healthy heartbeat, we would expect the number that these values increment by to be within a certain range. If the incrementation strays from what is considered normal, it could be possible to give an estimation regarding the patient’s cardiac health.  
  
We expect that the signals should fit within certain “norms”. The P-Wave should come first, followed by the Q-Wave, R-Wave, S-Wave and T-Wave. The results show this to be true for most of the data however, there is an issue with the first and last row for the Q and R waves.  
The Q-Wave is in-front of the R-Wave in terms of time which should not be possible. This is caused by the way that individual heartbeats are found in the software. The R Peak is found initially and then the data is split on both sides of that peak between the closest R Peaks.  
The start and end of the signal do not have R Peaks on which to split, so the start of the dataset and end of the dataset are used instead. This data is then processed, and the Q Waves are found leading to the issue mentioned above.

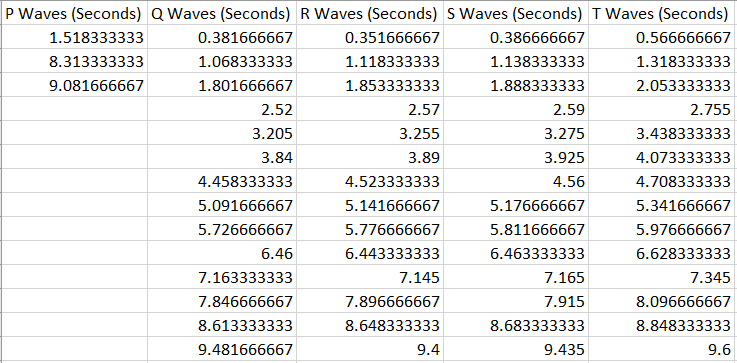


Figure 28 - Lying Down 1 PQRST Wave Timings (FFT Filtered)

This software does not give an estimation of a person’s health based on these values because this is out of the scope of the project. However, it was important to save this data to show that the values had been extracted from the signal ready for classification of health.

**4.d Heartrates and Zones**

The heartrate and heartrate zones are saved in seconds to an excel spreadsheets in the “Results” folder. These readings are calculated based off the R-Peaks found within the ECG signal. By multiplying the number of R-Peaks found in the dataset (which lasts for 10 seconds) by 6, we can calculate the approximate heartrate of the individual.  
  
Heartrate zones are calculated by taking influence from different diagrams because the zones change dependant on which source is used [18].  
  
Looking at Table 1 and Table 2 in 5.b called “Number of PQRST sections found for each file – (FFT/No FFT)”, the R-Peaks found are the same number. This means that both the FFT filtered data and non-filtered data will show the same heartrate and zone. If all of the R-Peaks are detected, the heartrate will be accurate using this method.

**5 Critical Evaluation**

**5.a Reliability**

The reliability of the software is hard to judge. The software is created for the use of data which has gone through a certain process. The data comes from the Welch Allyn electrocardiograph and is captured at 600hz. The data is then converted to CSV using ECG Toolkit 2.4.  
There were no datasets that were found which fit directly into this category so no external data was tested. No technician was on hard to get additional data from the Welch Allyn electrocardiograph. This software is known to work for my personal heartrate only.   
The data that is produced by the software is on the files which were tested is reliable. The PQRST sections match up to their correct location in the final graphs that are produced and this is the same as the data which is saved to the CSV file.  
  
Certain checks have been put in place to be able to deal with other data where the heartrate is significantly different than my own.  
No additional checks have been put in place for people that have certain sections from the PQRST sequence that are missing due to a cardiac issue. This would have been hard to test for as I do not believe that I have a cardiac issue.   
Currently, certain sections are not found during the software which is due to noise. Attempting to separate actual missing sections from sections missed due to noise would be an ideal improvement to the software.  
  
More of the PQRST sections can be found if the signal is of a better quality. If the signal quality is bad to begin with due to noise when recording the signal, the data will be harder to read. There is a certain baseline which the software needs in terms of quality. This is true for all software which attempts to analyse signals as eventually, removing the noise removes too many features from the signal.  
  
If the user was able to add their age as an input to the UI, the heartrate zones could be calculated from this data instead of using a 25-year-old’s values which would lead to unreliability if the person was not 25.  
  
To increase the quality of the data, additional filters could be set up on the electrocardiograph when data is being processed. This would help to deal with electromyographic noise and baseline drift.

**5.b Speed**

The software currently iterates through 11 different files, each containing 6,000 datapoints. In total that amounts to 66,000 datapoints which must be read and manipulated multiple times throughout the software. Graphs are also produced which take time and are processor heavy.  
  
The software takes a long time to run but this will depend on the specifications of the computer that are running it. From testing, I calculate that it takes an average of 1 Minute, 40 seconds to process all 11 files on my laptop.  
To keep processing time down, the number of files which are processed can be reduced. If the software was to be used for solely finding PQRST sections without showing the process behind the software, all the graphs produced except for the last graph could be commented out which would increase the speed.  
  
Certain functions could also be changed to make the calculations which are performed to be faster. This was not done in part since Python 3 is not the main language that I program in. There was also considerably more time spent attempting to change certain variables in order to get a better signal than attempting to speed up the program.  
  
I would state that the speed of the software is acceptable considering the graphs and data produced although it would have been beneficial for testing purposes to allow for buttons on the UI which stops certain graphs from being created.

**5.c Usability**

The software is simple to use. A UI opens when the program is run that shows a list of instructions for the user. The user selects a folder that contains CSV files which have been processed in the correct way and the program creates graphs and CSV files in that folder.  
  
The program does not notify the user when the calculations have been completed which could be changed for better usability. This was not implemented due to time constrains and issues with understanding the PyQt5.  
  
The UI does not show graphs or the extracted data to the user. This would have been difficult to implement since the number of files which could be processed at any one time can be extremely large. 11 files is already a large number and produces a lot of graphs to show the process.  
  
For the purpose of this project, I believe that the usability is simple and meets the requirements needed in order to process data.

**6 Conclusion**

This project was interesting and informative in nature and required a lot of research. I have learned a lot about the nature of biological signals and the heart. It was surprising to me that I was able to understand the biological aspects of the project and use this knowledge to write the software and support claims as well as I have.  
  
There were difficulties when initially starting the project as it originally started as a project based on PPG sensors and smartwatches. This quickly changed after meetings with my supervisor and discussions having discussions about ECG’s. The original project outline can be found in Automatic\_Heartrate\_monitoring.docx.  
The revised project outline including the description is contained within this document.  
  
The initial prototype that was created attempted to plot the graphs using Matplotlib but most of this code was removed and changed for the final piece of software. Through this I learned that it is sometimes better to change the direction that software is moving and change the code rather than sink more time into code that is not working.  
  
Python is not a first-language to me so it was an unusual experience to get used to some of the features (or lack-thereof) such as the importance of indentation and no need for parenthesis and semi colons. The different mathematical libraries used were foreign to me at the start of the project and required time to wrap my head around which paid off and led to me being more informed as I read the documentation  
FFT is a concept which I previously knew about but had not implemented. I now understand how useful FFT Filtering can be to remove noise from signals.  
Spline interpolation is one of the favourite things that I have learned about. It is extremely useful to remove baseline drift and when I saw the first graph that was created using this method, I was astounded.  
  
Even though certain P-Waves (in particular) are not found, the software provides the function that it was created for – to find PQRST sections. I am proud of this fact and that the software works as well as it does.

TODO – Remember to add fft results as an attachment

TODO – Remember to add normal results as an attachment

Put github abnd trell9 links

Clean trello and readme

What is included and foldter structure

Findqrscomplex is now findheartrate

Only takes into account my heartrate due to lack of samples – machine should give same rates for most people

Is baseline drift explained anywhere? If so maybe add a photo to show what I mean

The development process

Zeros in results/pwrst sections in seconds are caused by array initialisation – Fixed this so ignore

To check if these results are good however you can check the graph to see which points are off

Talk about results containing a folder that holds pqrst sections

Q wave has sometimes come after r wave (weird) – So you can filter this stating if time (seconds) is after r wave – ignore

The noisier the data gets, the harder it will be to determine signal

Talk about passing through the first cubic spline interpolation after finding r sections and calculating heart rate. This would help eliminate more of the drift….

The code was taken from a website as WELL - http://scipy-lectures.org/intro/scipy/auto\_examples/plot\_fftpack.html

Talk about how getting fft from the internet was the best thing that I could find to make it work

Talk about find zero crossings

Make it obvious when talking about FFT that it is not used (Put it in the correct place when talking about code as well)

TODO – I think I said there are 4 of me sitting up somewhere. It is actually 5

Change lying to laying

Without the fft frequency removal – There were issues finding the p peaks – After adding fft frequency removal, p peaks were found

Issue with separating heartbeat (start of file and end of file)

It was eventually decided that if the program used pre-recorded ECG data then multiple files could be accessed at once by the user selecting a folder containing all of the files which they wanted to be processed.

Spent a lot of time cleaning up code – Making functions useable multiple times and getting rid of similar lines

Not always perfect but….

Add in 1 reading and its process through the whole thing as graphs

# Check for TODOS and swears

# Change readme to explain how to use this program compared to the other programs

Program takes a while to run because of all of the graphs

# Write tests for the code

Should work on any computer and create the files for you – Check this

TODO – Add code lines somewhere!!!

Change default folder for another persons computer?

TODO – allow for the CSV data to be saved again in a better format.

TODO – add somewhere that the signals must be added together (Explain why – certain leads give rubbish results)

TODO – talk about 10 second ecgs

TODO – talk about 600hz

Clean trello and add pictures of it

Show github repo as well

Will code run on another computer?

Talk about zero crossings

Talk about wanting to do more ui

Talk about saving out the csv data again in the proper format…

Neural network stuff

# Talk about library needed to be downloaded in PyQt to make the code work!!!!!

Talk about it taking a long time to get variables just right – How a lot of variables depend upon the heart rate itself

# Could not get fedeicos data. We are here for long term. Will be working on project over summer. - Tsted this for fun sort of stuff – explain this

Write guide for running program

TODO – If variable names are changed to s\_gfsgd\_fdsfsa from SGfsgFdsfsa – Sort out in report as well

Sort out folder with all relevant material needed

Talk about what I would have done if I had more time

TODO – Talk about naming conventions – Classes are capital letters etc

TODO – Go more in depth about welch allyn system

Which python version is being used?

# State what graphs folder does and what results folder does in report

Talk about sticking to correct naming conventions for variables: l\_l and classes MyClass

**Chapter 9  
Annotated Bibliography**

[1] Hela Lassoued and Raouf Ketata, IEEE, ECG multi-class classification using neural network as machine learning model.  
<https://ieeexplore.ieee.org/document/8379901>, 2018.  
Text: Used as research.  
Accessed March 2019.  
  
*A paper published by IEEE. IEEE is a reliable company that sets plenty of standards in the industry.*

[2] Inc Argosy Publishing. At the Heart of It All: Anatomy and Function of the Heart.  
<https://www.visiblebody.com/learn/circulatory/circulatory-the-heart>, 2008.  
Text: Direct Quote.  
Accessed April 2019.   
  
*A webpage as part of a broader website by Argosy Publishing that is used as a 3D human anatomy visualisation and learning tool. This webpage allows for a detailed look at the structure of the heart and gives a thorough breakdown of how the heart works.*

[3] Cleveland Clinic. Heart Beat.  
<https://my.clevelandclinic.org/health/articles/17064-heart-beat>, 2018.  
Text: Direct Quote regarding SA node.  
<https://my.clevelandclinic.org/health/diseases/16765-atrial-fibrillation-afib>, 2019  
Text: Research regarding Atrial Fibrillation  
Accessed March 2019.  
  
*A webpage operated by an American academic medical centre that is based in Cleveland, Ohio. The webpage discusses the heart’s electrical system and the normal electrical activity that is provided via the SA node in order to allow for blood to pump to and from the heart.*

[4] My Health, AV node ablation for atrial fibrillation.  
<https://myhealth.alberta.ca/Health/pages/conditions.aspx?hwid=zm6205>, 2018  
Image: Cross section of the heart.  
Text: Used for research.  
Accessed March 2019.  
  
*A website operated by the Alberta government and Alberta health services to allow for access to health related information. Used as research to illustrate how the SA node and AV node work.*

[5] AliveCor, What is an ECG?  
<https://www.alivecor.com/education/ecg.html>, Unknown  
Text: Used for research in regards to the SA node, AV node and the PQRST Sequence.  
Accessed March 2019.  
  
*A webpage ran by AliveCor which is a medical device/Artificial Intelligence company which sells ECG hardware. Used for research detailing how the SA node and AV node work inside of the heart.*

[6] Medical News Today, Sinus arrhythmia: Definition, signs and diagnosis  
<https://www.medicalnewstoday.com/articles/319987.php>, 2017  
Text: Used for research  
Accessed March 2019.  
  
*A webpage ran by Medical News Today – a news outlet that reports the latest medical news. Used to research and support the link between Sinus Arrythmia and the SA node.*

[7] Ramos, Guilherme & Alfaras, Miquel & Gamboa, Hugo. (2018), Real-Time Approach to HRV Analysis  
Available from: <https://www.researchgate.net/figure/ECG-PQRST-points_fig1_322879235>, 2018  
Image: PQRST Sequence as shown as an ECG.  
Accessed April 2019.  
  
*A paper that was written which can be found on the link above. The image has been used for illustration and the image contained in this paper is widely known to be correct.*

[8] Author: N/a. University of Washington, ECG Filtering.  
Available from: <https://courses.cs.washington.edu/courses/cse466/13au/pdfs/lectures/ECG%20filtering.pdf>, 2013.  
Text: Used for research regarding Baseline Wander, Power Line Interference and Muscle Noise.  
Accessed March 2019.  
  
*A lecture that is from the University of Washington. This source does not have a lecturer’s name assigned to it. It can be deemed reputable considering that it is on the University of Washington’s website. Additionally, there is another reference [9] that collaborates the findings in this lecture. This lecture talks about different methods to filter ECG signals*

[9] Cecilia Vinzio Maggio, Ana & Bonomini, María & Laciar, Eric & Arini, Pedro. (2012), Quantification of Ventricular Repolarization Dispersion Using Digital Processing of the Surface ECG.  
Available from: <https://www.researchgate.net/publication/221923119_Quantification_of_Ventricular_Repolarization_Dispersion_Using_Digital_Processing_of_the_Surface_ECG>, 2012.  
Text: Used for research to collaborate findings with reference [7] about Baseline Wander, Power Line Interference and Electromyographic Noise.  
Accessed March 2019.  
  
*A study conducted to quantify the QRS Complex by processing an ECG. This study discusses the changes that are needed to be made to raw data in order to extract information.   
The information provided here is backed up by many other sources*

[10][Weisstein, Eric W.](http://mathworld.wolfram.com/about/author.html) "Fast Fourier Transform." From [*MathWorld*](http://mathworld.wolfram.com/)--A Wolfram Web Resource.   
FFT Link: <http://mathworld.wolfram.com/FastFourierTransform.html>, 2016.  
Text: Used for researching fourier transform.  
Accessed March 2019.  
Cubic Spline Interpolation Link: <http://mathworld.wolfram.com/CubicSpline.html>, 2016  
Text: Used for research cubic spline interpolation.  
  
*A website that is dedicated solely to teaching. The information contained within is backed up by multiple sources*

[11] Healthy Doc, How Does PPG Technology Work?  
<https://soulfit.io/blog/how-does-ppg-technology-works/>, 2018  
Text: Used for researching how photoplethysmography works.  
Accessed March 2019.  
  
*Soulfit is a wellness institute in England. The writer of the article (Healthy Doc) is unknown. The findings in this article are backed up by multiple other sources on the internet however.*

[12] Daniel McDuff\*, Student Member, IEEE, Sarah Gontarek, and Rosalind W. Picard, Fellow, IEEE.  
<https://affect.media.mit.edu/pdfs/14.McDuff_et_al_Remote.pdf>, Date: Unknown.  
Text: Used for researching how PPG sensors can calculate systolic and diastolic pressure.  
Accessed April 2019.  
  
*A paper written by members of IEEE. This paper describes the way that PPG sensors can be used to ascertain data relating to a person’s health. IEEE Is a reputable source that is well trusted*

[13] Kaggle, ECG Datasets.  
<https://www.kaggle.com/shayanfazeli/heartbeat>, Date: Variable.  
Data: ECG datasets were downloaded from this link at the start of the project. These datasets were not used after week two and were only for testing ideas.  
Accessed February 2019.  
  
*Kaggle is a website that acts as a community for data scientists and computer scientists. The website shares information and datasets. The dataset that was used was of (or at least highly-resembled) ECG data when plotted onto a graph*

[14] Welch Allyn, CardioPerfect Resting ECG Quick Start Guide.  
<https://www.welchallyn.com/content/dam/welchallyn/documents/sap-documents/LIT/80012/80012274LITPDF.pdf>, 2018  
Image: The image of where the leads are placed is used to illustrate how the ECG data readings took place.  
Accessed April 2019  
  
*Welch Allyn is a medical diagnostic device manufacturer based in America. Considering that the image is used by professionals to set up their equipment which was provided by Welch Allyn, this is the best source.*

[15] mvanettinger, C# ECG Toolkit.  
Downloaded from: https://sourceforge.net/projects/ecgtoolkit-cs/files/ecgtoolkit-cs/ecgtoolkit-cs-2\_4/  
Used for: Converting SCP data into CSV data so that it was accessible.  
Downloaded March 2019  
  
*This software was found online after attempting to use similar freeware. There is not much information on the software, but a vigorous virus scan has shown no issues and the computer that was used for this project does not have a virus. The software does the job that it was intended for.*

[16] SciPy Lectures, 1.5.12.18. Plotting and manipulating FFTs for filtering.  
<http://scipy-lectures.org/intro/scipy/auto_examples/plot_fftpack.html>, Date: Unknown.  
Code: Code block taken from lines 305 – 316 in signalProcessing.py.  
Accessed March 2019  
  
*SciPy-Lectures is a website that allows for people to learn how to use SciPy. The source is reputable and the code which was copied worked fine.*

[17] SciPy, scipy.signal.find\_peaks.  
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html>, Date: Unknown.  
Text: Direct quote from the above link.  
Accessed April 2019  
  
*SciPy is a python library that allows for certain mathematical and scientific functions. The library is frequently used and widespread around the Python community lending to its reputability.*

[18] Wikimedia, Exercise Zones.  
<https://commons.wikimedia.org/wiki/File:Exercise_zones.png>, Date: Unknown.  
Image: Image is used to show different exercise zones.  
Accessed April 2019  
  
*Although Wikimedia’s reputation is not the best, the image serves as a template for what my zones were calculated from. The calculation vary dependant on age, sex and in some cases even weight*