Charlie Maclean

Synthesis of Heart-Rate Detection Methods

Computer Science Tripos – Part II

King's College

March 6, 2020

Proforma

Name:	**Charlie Maclean**
College:	**King's College**
Project Title:	**Synthesis of Heart-Rate Detection Methods**
Examination:	**Computer Science Tripos – Part II, July 2020**
Word Count:	**TODO**[1]
Project Originator:	Dr Robert Harle
Supervisor:	Dr Robert Harle

Original Aims of the Project

To research and implement the detection of heart rate from smartwatch sensors. To investigate the effectiveness of a selection of filters and peak finding algorithms. To use accelerometer data to find motion artifacts within the data, and compare methods of removing these artifacts.

Work Completed

All that has been completed appears in this dissertation.

Special Difficulties

Learning how to incorporate encapulated postscript into a LaTeX document on both Ubuntu Linux and OS X.

[1]This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

Declaration

I, Charlie Maclean of King's College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

List of Figures

Acknowledgements

This document owes much to an earlier version written by Simon Moore [?]. His help, encouragement and advice was greatly appreciated.

Chapter 1

Introduction

Elite runners have historically used heart rate to provide an accurate measure of fitness, and allow them to train more effectively. [Expand on uses of heart rate]. Previously, Electrocardiography (ECG) chest straps have been used to measure heart rate, by detecting the electrical signals controlling the expansion and contraction of the heart. They are accurate devices however often prohibitively expensive, and hence inaccessible to casual runners.

In recent years, a new technology has emerged - Photoplethysmogram (PPG) - light is directed at the skin, and sensors measure how much blood vessels scatter it. PPG sensors are cheaper than ECG sensors, and hence are available in a variety of products, particularly smartwatches. This innovation has bought a new wave of advanced training and monitoring onto the wrists of any runner.

Switching from ECG to PPG is not without flaws though - ECG sensors return a clean signal, as opposed to PPG signals which are contaminated with noise. [More details about noise].

In particular when running, the motion can cause blood velocity to change, and the sensor can slip across the skin [9], [10]. These result in a distortion to the PPG signal, known as a motion artifact (MA). Fortunately, smartwatches contain other sensors, such as accelerometers and gyroscopes which can be used to predict the presence of MAs, and hence compensate for them.

My task is to research and develop a heart rate detection algorithm for smartwatches worn during running.

Chapter 2

Preparation

This chapter details the steps I took to determine how to develop my implementation. It first describes photoplethysmography, the method with which heart-beats are detected on watches. Next, I give an overview of the algorithms I will use to extract the heart-rate from the PPG signal: filters, peak-detectors and motion artefact removers. Finally, I outline my development methodology.

2.1 Photoplethysmography (PPG)

To develop an algorithm to track heart-rate on wrist-watches it is first important to understand how watches track heart activity. PPG is a technique where light is used to detect the volume of blood in veins. In hospitals, this is used in finger pulse oximeters (Figure 2.1) to record the heart-beat of patients. These work by transmitting light on one side of the finger, and then measuring how much light is received on the other side of the finger. The amount of light which permeates through the finger is related to how much blood is currently in the veins.

Figure 2.1: Finger pulse oximeter. Image source [3].

With watches, we cannot transmit light on one side, and receive light on the other side, as the wrist is far too large. Hence, instead of monitoring the absorption of light by the skin, we monitor the reflection of light by the skin. A light is shone into the wrist, and sensors nearby monitor how much is reflected back. When the wrist is full of blood, more light is reflected back as blood scatters lights.

We know now how the signals are recorded, but we have not yet explored the signal that is actually received from this technique. In Figure 2.2 we see a clean PPG recording. There are two peaks in the data - a sytolic peak and a diastolic peak. The systolic peak represents the point at which the heart has beat - and hence pushed blood through the body. The challenge is to find this peak.

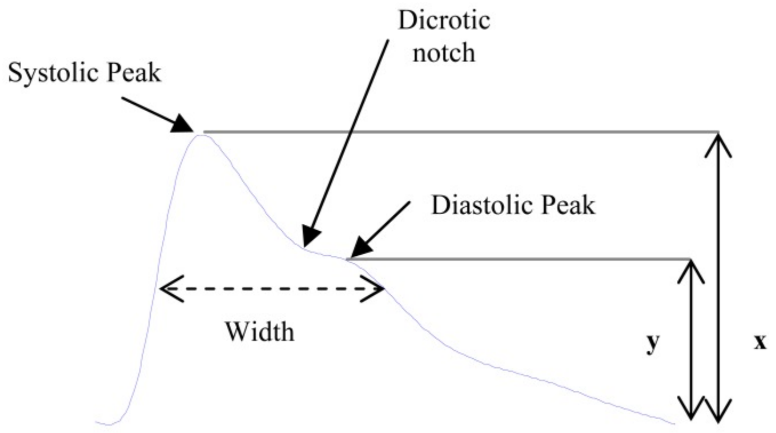


Figure 2.2: Example PPG signal. Image and annotations from [4].

Chapter 3

Implementation

3.1 Gathering Data

In this section I detail the process of developing an application to record PPG data on a Wear OS watch. This turned out to be more complicated than I presumed due to interesting issues inherent with developing for a small wearable with limited power.

3.1.1 Wear OS Development

Language Choices The watch application was developed using the official IDE for Android development - Android Studio. I programmed in a language I'd never used before - Kotlin. Kotlin is an open source language based on Java, which aims to reduce boilerplate code, adds null-safety, and remains interoperable with Java, allowing libraries written for Java to be used in Kotlin.

Structure The program was split into classes as in Figure **??**. The behaviour of each class is as follows:

- *MainActivity* - central class responsible for starting, stopping and saving the recordings.
- *SensorListener* - interface which overrides the Android *SensorEventListener*. Contains methods which are called by the WearOS system whenever new sensor data comes in. Additionally contain methods which save the received data. We need one child for each sensor as each different sensor provides different data, and must be saved in a different format. So we have the following children:
  - *PpgListener*
  - *AccelerometerListener*
  - *RotationListener*

When the user presses start recording, *MainActivity* registers each *SensorListner* with the OS so they receive any sensor changes.

When the user presses stop recording, *MainActivity* unregisters the listners and gets each one to save their recording.

Storing Recordings It was critical that I could easily import the recordings into any program, and hence I chose to export the data into a CSV text file. Each *Listener* class uses a *CSVWriter* object from library *opencsv* to write to a unique file for each sensor, within a directory chosen by *MainActivity*. For example, at the end of the recording in the directory *YYYY-MM-DD/HH.MM.SS/* we have files *ppg.csv*, *accelerometer.csv* and *rotation.csv*.

Interface I chose a simple yet functional interface to allow the user to start and stop a recording. The interface changes colour when recording, which makes it very easy to verify the recording is happening. Pictures of interface in figure **??**.

Problems After I'd developed the application and was testing it out I found two key issues which I will explain here.

Power Saving Power is an enormous issue on wearable devices, as the form factor drastically constrains the size of the battery. Hence developers have explicitly designed Wear OS to limit power usage of any application as much as possible. As part of these optimisations Wear OS automatically suspends any application if it thinks they are not being used. While this is useful for most users, my application needs to run continuously without interruption.

Initially in testing there were long periods of time where no sensor values were being recorded, due to the application being suspended.

To fix this issue, a wake lock was included. Once a wake lock is acquired, the OS does not suspend the application. I added one which is acquired every time a recording is started, and released when the recording ends.

Sampling Rate When I subscribe to values from the PPG sensor and accelerometer sensors, the API asks for a parameter - sampling period, which is the amount of delay we want between sensor readings being delivered. In practise, Android only uses this number as a hint - values may be provided at a higher or lower rate.

Additionally, Android does not expose the actual rate at which sensor values are being provided to the OS. The sensors which are placed within devices always have a constant sampling rate. Sensors write to a very short buffer, which Android then reads from at a rate it decides. This means although we have a reliable sensor, Android occasionally misses readings.

I record the data with a sampling period of 0 - which signifies that I want updates as soon as they are available. However, Android is not guaranteed to provide this. I mitigate this by noting the time each recordings is provided to the application. Then, I define sampling rate to be equal to the time difference between the first and last sample, divided by the number of samples.

So, I have created an application which allows the user to record accelerometer and PPG values at a constant sampling rate, then save them to the watch.

3.1.2 Uploading Recordings

Now data has been recorded, the values must be uploaded somewhere to enable later analysis. I created a server which would run on a Raspberry Pi that the watch could connect to over the local network. The files would be passed to the server, and from there the files are uploaded to Google Drive where they can be accessed anywhere. In this section I go over the details of this implementation. See figure 3.1 for the overall uploading process.

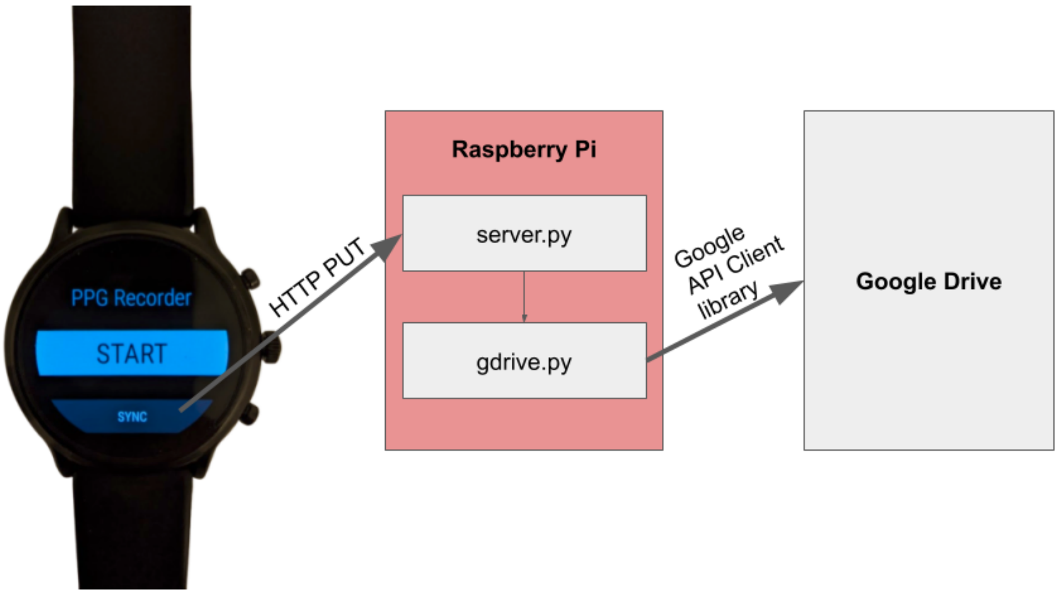


Figure 3.1: The overall process of uploading files.

Changes to Wear OS Application

First a 'sync' button was included in the interface, which users can press to trigger the sending of files to the server.

Then I added the logic to be triggered when the button is pressed. This logic iterates through every file in the recordings directory and performs a HTTP PUT request with them. I used...

3.1.3 Raspberry Pi Access Point

To connect the watch to the server I needed to create a local access point on the Raspberry Pi. The packages I used to do this were *hostapd* and *dnsmasq*.

The package *hostapd* is used to setup the access point, so I used it to configure the name you can find the network from, the password of the network and the channel it can be accessed on.

The package *dnsmasq* is used to setup the DHCP service, assigning IP addresses to devices. I set it up to assign the range of adresses from 192.168.0.1 to 192.168.0.255 on the same interface as *hostapd* is broadcasting over.

Flask Web Application on Raspberry Pi

Now, to setup the server I used Flask, a Python Web App framework. I chose this over other frameworks like Django as it provides a flexible framework which is easy to understand and deploy.

I created an application *server.py* which accepts PUT requests containing a file from URL */upload/path* where *path* is the location we want to place the file. Then, the server makes the directories up to *path* if they don't exist, placing the file in this directory. Unlike in a normal server implementation, I do not secure this file upload process. I am aware there are a number of security issues concerned with leaving a file upload with arbitrary path options, for example a user might be able to upload a file in the parent direction by placing a PUT request for */upload/../../example*. However, I do not worry about these attacks as the access point is secured by *hostapd* and only the watch will be connected.

Once the file has been saved, I upload the new file to google drive, by calling *gdrive.py*, which I detail below.

Python Scripts to Upload to Google Drive

To manage the process of uploading files to Google Drive, I created a new application *gdrive.py*. To upload to Google Drive using Python, I used the library *googleapiclient*, which was a surprisingly complicated API, as I will explain here.

First, to connect to Google servers, the API must authenticate to ensure that it has permission to access a user's Drive. The service requires a credential token which expires periodically. Hence, I store the credentials in a binary file using library *pickle*, and when I need to access them, I refresh them if they have expired.

Then, I must actually upload a file. This process is complicated as Google Drive does not use a path system to store files. Instead, it is a set of files each with a unique IDs. A folder is simply a file with a *folder* file type, and a collection of files associated with it. Furthermore, as files are not referenced by filename, it is possible to have multiple files in the same folder with exactly the same filename.

The Google Drive API for Python only contains two methods which are useful for my application:

- *list(parent)* which lists the files within folder *parent*.
- *create(file, metadata)* which creates a *file* with *metadata*. Note that as folders are just files, this is also how folders are created, the fact a file is a folder is specified within *metadata*.

Using these methods, I define three methods within *gdrive.py*, as follows:

- *createFolder(name, parent)* creates folder by creating a file with metadata specifying the type as a folder.
- *getOrCreatePath(path)* is used to obtain a 'path' from Google Drive. It works by iterating through each level of the path, creating the necessary folders if they don't already exist, then entering them.
- *uploadFile(filepath)* created the path up to the file given by filepath, and then upload the file to the folder.

Now, I have created a system which allows the user to upload recordings from the watch to Google Drive via my server. And thus, I have completed the data recording process. The data is recorded on the Wear OS app, and then with a click transmitted to a Google Drive so it can be analysed.

3.1.4 Synchronising Signals

Once I have collected the PPG signal from the watch, and have recorded the ECG from the chest, I need a method to synchronise the signals, such that they start at the same time and we can compare them accurately.

I set out to produce a solution with the following properties:

- Able to synchronize signals with ± 0.3s accuracy. With a heartbeat of 200 bpm this represents being within a heartbeat. This is an acceptable level of delay, as heart rate is averaged over several beats anyway.
- Can synchronize signals given the two recordings are started within two minutes of each other. This constraint is helpful as it prevents wasted time searching through the signal.
- Does not use the clocks built into the device. We must assume the clock within the ECG is unreliable. Additionally, the two devices may be synchronized to different time, and hence could be out by any amount of time.

Given I know both devices have an accelerometer, I realised I could ask the wearer to move in some motion which is picked up by both devices. Then, I could compare the two accelerometer signals in order to discover the movement. Then comparing the starting times of the two signals would enable calculation of the time difference between them.

The type of motion I chose was important, as it had to be easy to explain to the wearer, but also provide sufficient motion for it to be identifiable against normal motion. I decided jumping was appropriate - I would ask the wearer to hold the watch close to their chest and jump three to five times.

The cross correlation of two signals f and g is a measure of similarity as a function of the displacement between them. It is simple to calculate, as a sum of the products between the samples of f and of g displaced by n.

To synchronise the accelerometer, I developed the following three step algorithm:

1. Normalize signals, by moving to zero mean,
2. Crop PPG signals to two minutes, crop ECG signals to four minutes, compute cross correlation with f = PPG acceleration, g = ECG acceleration.
3. Compute the other way - crop ECG to two minutes, PPG to four minutes, compute cross correlation with f = ECG acceleration, g = PPG acceleration. I compute both ways, to ensure that we can synchronise regardless of which device started recording first.
4. Find the maximum cross correlation across each of the calculations.

3.1.5 Informed Synchronisation

Running cross correlation on two signals is an effective way to compare their similarity, however it is not efficient. Cross correlation has complexity $O(n^3)$. In this section I develop a new algorithm which uses knowledge of the problem to speed up the cross correlation calculation.

By graphing the cross correlation of the acceleration signals, we can draw some useful features that help us to work out a more optimal solution. See figure 3.2 for an example of absolute cross correlation plotted between the two accelerometer signals over time. It is clear that there is one peak which is above all the others, where the jumps line up perfectly. However, in addition we can see that there is a significant rise in cross correlation around the peak, where we are seeing peaks due to one or two of the jumps aligning. This is useful as we hence do not need to scan at each possible time difference, instead we can to a lower resolution initial scan. After this initial scan is done, we find the peak, and check each possible time difference around the lower resolution peak to find the actual peak.

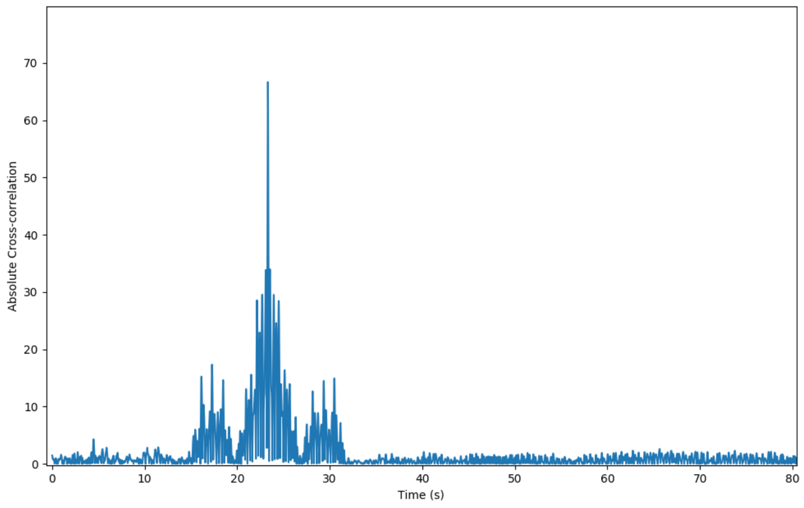


Figure 3.2: Absolute Cross Correlation over time.

The algorithm I designed takes two parameters: step size (g) and correct range (r). Step size is the resolution of the initial scan, so the higher it is the more likely we are to find the peak but the longer the algorithm takes. Correct range gives the range of the time period we want to scan once we have found an initial peak, in order to find the actual peak.

3.2 Filtering

PPGs produce a large amount of noise from various sources:

- Light pollution. Both DC and AC lights in the environment can infiltrate the sensor, affecting the PPG signal [5].
- Temperature. As temperature increases, so does the volume of blood, which will be picked up by the PPG sensor [7].
- Breathing. The change in pressure accosiated with respiration causes variations in the flow of blood, and hence can be seen by the PPG sensor [1].

In order to remove much of this noise, we can use filters to remove frequencies we know are irrelevant. We know the heartbeat can vary from 30 to 220 beats per minute, and hence we would like to disregard any noise outside of this range. In this section, I will introduce the concept of filters, and describe two filters useful for PPG signals.

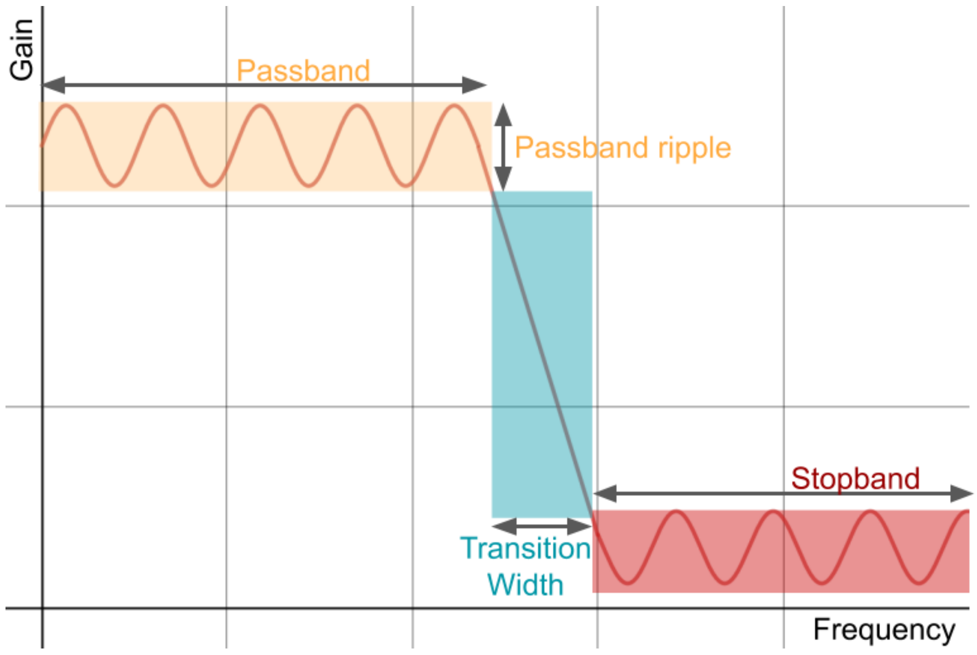


Figure 3.3: Diagram showing filter characteristics

Figure 3.3 displays an example filter magnitude response diagram. This plots the amount of gain applied to each frequency. A gain of zero means the signal is removed, a gain of one means the signal is unchanged. The characteristics displayed are as follows.

The passband is the range of frequencies we would like to remain. In an ideal filter, there is no loss within the passband.

Passband ripple describes the variation in amplitude *within* the passband. An ideal filter will have no passband filter, such that all frequencies within the passband are permitted equally.

The transition width is the frequency range between the start and stop band. Ideally, this is zero, such that frequencies outside of the passband are instantly reduced.

The stopband is the range of frequencies we wish to remove. An ideal filter completely removes all frequencies within the stopband.

The ideal filter, as described above, is impossible - and hence we have a variety of filters which compromise between the desirable characteristics. Following this, I detail two of these compromises - the Butterworth filter, and the Chebyshev filter.

3.2.1 Butterworth Filter

The Butterworth filter aims to minimize passband ripple, at the expense of a larger transition width. To define a Butterworth filter, multiple parameters are used, which I will describe further here.

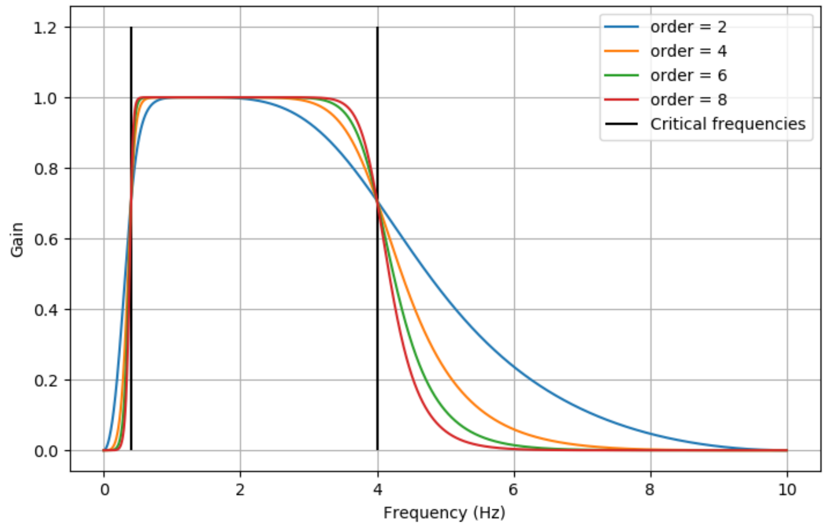


Figure 3.4: Comparison of Butterworth filter orders

Order (N) controls the transition width - the higher the order, the lower the transition width. This is desirable - we want to keep the passband as large as possible, and transition quickly to the stopband. However, it comes at the cost of computation complexity, as higher order filters take longer to apply. See figure 3.4 for a plot displaying the effect of different orders on a filter's response.

Critical frequency gives the frequency at which we want gain to drop below $1/\sqrt{2}$ from the passband. Hence we use this parameter to define the point from which we wish to cut out noise. Figure 3.4 uses critical frequencies 0.4 and 4 hz, which correspond to 24 and 240 bpm, a reasonable heart rate range.

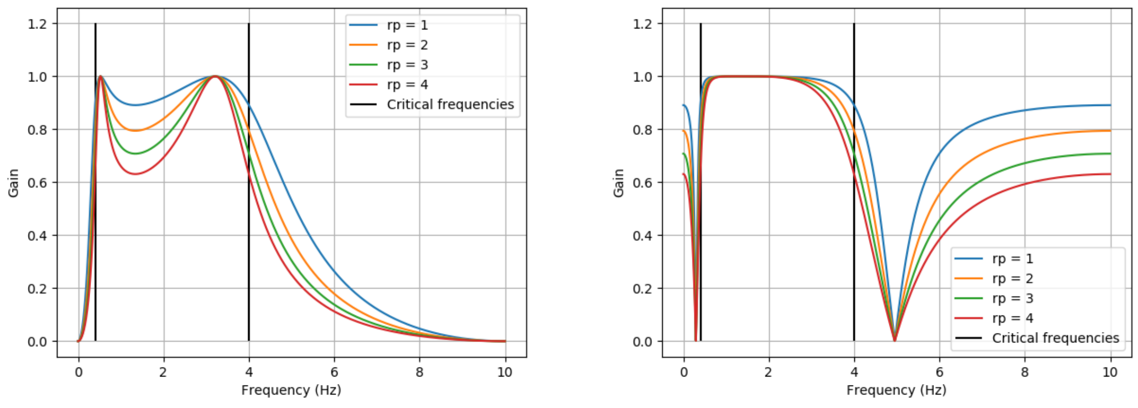
Type describes which frequencies we cut out. There are four different types of Butterworth filters, as follows:

- Lowpass - allow frequencies below the critical frequency.
- Highpass - allow frequencies above the critical frequency.
- Bandpass - combination of lowpass and highpass - given two frequencies, we want the passband to be between those frequencies.
- Bandstop - given two frequencies, put the stopband between them, leaving the passband outside those frequencies.

We will be using a bandpass filter, as there is low and high frequency noise we wish to remove.

3.2.2 Chebyshev Filter

The Chebyshev filter aims to reduce the transition width as much as possible, but to do this it introduces increased ripple.



(a) Chebyshev Type 1 (b) Chebyshev Type 2

Figure 3.5: Comparison of Chebyshev filter rs values

The filter is defined with similar parameters to a Butterworth filter - order, critical frequencies, and type all have very similar meanings. However, there are type 1 and 2 Chebyshev filters, with different aims. Additionally, there is a new parameter *rs*. I will explain these differences here.

Type 1 Chebyshev filters introduce passband ripple, whereas type 2 Chebyshev filters introduce stopband ripple. Figure 3.5 shows how these differences manifest in the response of a second order Chebyshev filter.

The parameter *rs* is defined as the maximum ripple permitted below unity gain. Higher *rs* results in more ripple for a type 1 filter, but less ripple for a part 2 filter, as is demonstrated in figure 3.5. Introducing more ripple is undesirable as leads to more signal distortion, however can be advantageous as it reduces the transition width.

3.3 Motion Artefact Reduction

In this section I explore a method which can be used to reduce the effect of Motion Artefacts (MAs) on the PPG Signal.

3.3.1 Adaptive Noise Cancellation

Adaptive Noise Cancellation (ANC), as described by Widrow et al. [8], is a technique which allows us to remove noise from a signal, given we have another signal correlated to the noise in some way. In our situation, we know MAs are correlated to motion, so we can use the accelerometer to remove MAs. This technique is extremely useful, as it can still produce good results when the MAs are at the same frequency as the heart beat, even if the MAs have a larger amplitude.

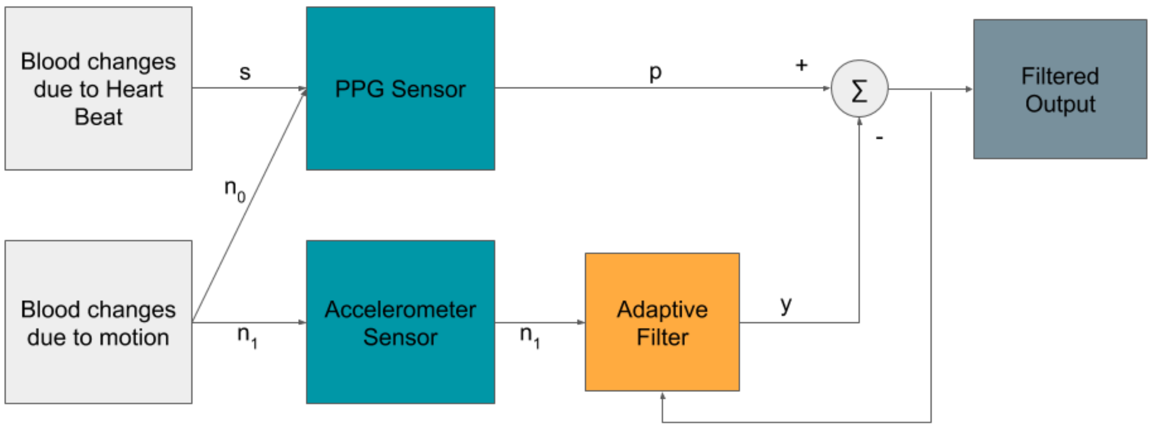


Figure 3.6: ANC overall flow

An overview of the algorithm follows. We have signal from heartbeat s that we want to figure out, but this is contaminated by noise from MAs n_0. We assume the PPG sensor reading p is such that $p = s + n_0$. Additionally, we have accelerometer sensor readings n_1. We filter n_1, producing signal y and subtract this from p, to produce $z = p - y$. Our aim is to adjust the filter, such that y is as close to n_0 as possible.

The type of filter we adjust is known as a Finite Impulse Response (FIR) filter. An Nth order filter is defined by N weights. These weights are convolved with the signal, hence filtering it.

The adaptive filter aims to choose a filter such that the power of the output $E[z^2]$ is minimized. Given s is uncorrelated with n_0 and n_1, and n_0 is correlated with n_1, it can be proven that minimizing the output power is equivalent to finding $z = s$.

There are multiple algorithms which exist in order to adaptively filter a signal. I'll look at two, first describing the least mean squares (LMS) algorithm from Widrow and Hoff. LMS first initializes all of the filter weights to 0. At each step, it finds the gradient of the mean square output $E(z^2)$. Each weight's gradient describes what would happen to the signal if we kept using that weight value. So if the gradient of a weight is positive we know that using that weight again would increase the mean squared output. Hence, at the end of each step we shift the weights down if the gradient is positive, and visa versa if the gradient is negative.

LMS takes a parameter - step size μ, which scales how much we move the weights by. The equation used to update weights is

$$w_{n+1} = w_n - \mu \nabla E(y^2)[n]$$

where W_n is the nth filter weight, and $\nabla E(y^2)[n]$ is the nth weight gradient.

Step size must be chosen very carefully, as if it is too large there is a risk of LMS missing the solution as it overcompensates for the gradient. If it is too small, then the algorithm becomes too expensive as we need many more iterations. The situation is made worse as the ideal step size changes with respect to the power of the input, making it very difficult to choose a step size which gives stability.

To solve these stability issues, NLMS was introduced. It is the same as LMS, except you normalize the input power each iteration. This makes choosing a step size which gives stability a lot easier.

So now we have a stable adaptive filter, which will allow us to tune a filter that will remove motion noise from our PPG signal.

3.4 Heart Rate Calculation

Now we have removed noise from the signal, we should have a signal representing the volume of blood in veins within the arm. As the heart beats, it sends a wave of blood through the body, forcing more blood through these veins. Hence, peaks in the PPG signal correspond to heart beats.

There are multiple approaches to finding heart-rate from our filtered signals, and I will detail a few here.

3.4.1 Local Maxima

If we can find the peaks in the signal, we can count them up and divide by the time period we count those peaks within, giving us a heart rate. One way we can define peaks is as local maxima.

The local maxima in a signal are defined as samples which are larger than both neighbouring samples. Finding local maxima is an easy problem to solve - intuitively, we can create an algorithm which iterates through the samples, comparing each sample to it's neighbours, which has complexity $O(n)$.

3.4.2 Peak-Peak Standard Deviation Minimization

Finding the local maxima is a naive approach that doesn't adapt to any problem specific knowledge. For example, we know heart-rate will not climb above 220 bpm, however the prior solution could report 300 bpm. Additionally, it might find multiple local peaks around a single heart-beat.

Ideally, we want to include knowledge about what a normal heart-beat looks like in our detection algorithm. One thing we can do is look at peak-peak intervals - the time that passes between heart beats. Surprisingly, this is normally not consistent - a healthy heart actually beats with some variation. However, when running it has been found that the heart beats with very little variation [6], and hence in our case we can assume that the heart beats regularly. The aim of this method is to use this assumption, along with minimum and maximum heart-rate to better detect heart-beats.

The method begins by calculating an initial set of peaks, by calculating the rolling mean of the signal, and marking points above that mean as peaks. Then, we compute the standard deviation of the time between these peaks (peak-peak interval). Then, we iteratively increase the rolling mean and again calculate the peaks, along with the standard deviation of their intervals. The idea is every time we increase the rolling mean we exclude more peaks. Then at the end, the peaks we actually select are the peaks which give a reasonable heart-rate (in the range 20-240 bpm), and have the smallest standard deviation.

So, we have developed an algorithm that attempts to find a heart-beat with regular peak-peak intervals, and is within a reasonable range.

3.5 Post Processing

3.5.1 Kalman Filter

So far, we have not considered previous heart-rate readings when considering the current calculation. This is naive, as heart-rate over the course of a run follows a very fixed pattern.

Figure shows heart-rate over a steady run. It is clear initially heart-rate ramps up until it reaches a 'steady-state'. From here on the heart-rate remains pretty constant, not responding to slight changes in exertion. Finally, the heart-rate decreases rapidly once the run has completed.

The Kalman filter is an algorithm which can compensate for noisy and inaccurate measurements such as what we get from the PPG signal. It consists of two phases:

1. Prediction step - predict the current heart rate based on the previous heart rate, and some model. In this step we can incorporate the accelerometer as a reasonable estimator for the change in running effort.

2. Update step - take the actual reading from the PPG signal and combine with the output of the prediction step.

The Kalman filter relies on modelling current heart-rate as a Gaussian distribution with it's standard deviation representing the certainty of the current estimate.

Kalman gain is the parameter used to weight the measurements against the prediction. A higher gain assigns more trust to the measurements, meaning the filter will respond quicker, a low gain produces more smoothing.

3.6 Earbuds

As an extension, I wanted to collect data from a new kind of PPG-enabled device: earbuds. In theory, these devices are less effected by motion noise as they are not affected by arm movement. Additionally, the ear has a rich set of veins from which we can gather PPG. For the purposes of this project I used a set of Jabra elite sport earbuds, which have a PPG sensor under the right bud - see figure 3.7.

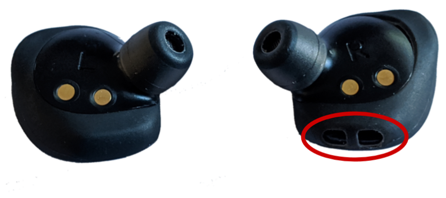


Figure 3.7: Positioning of the PPG sensor on Jabra elite sport earbuds.

In this section, I discuss how I obtained data from the earbuds, and then how I processed the data I received.

3.6.1 Gathering data

To collect data, I connected the earbuds to the phone, and then data is sent through the Bluetooth Light Energy (LE) protocol. There are a few key pieces of terminology, which I explain here:

Generic Attribute Profile (GATT) describes the overall methods with which data values are transported by Bluetooth LE.

GATT Server is a device which stores and distributes information. In our case the earbuds are a GATT client.

GATT Client is a device which requests information from a GATT server, in our case a mobile phone.

Characteristics are single data points, for example in our case a heart-rate integer. Characteristics also have a number of descriptors describing the value.

Services are logical group of characteristics, for example the heart-rate service contains the heart-rate measurement characteristic along with sensor position characteristic.

Characteristic Value Notifications are sent from client to server to indicate that the client would like to receive updates when a certain characteristic is changed. This is how frequently changing values are passed over Bluetooth LE, as it is more power efficient than having the client periodically request updates.

Figure 3.8 shows the flow I aimed to introduce in my data collection program to run on the phone.

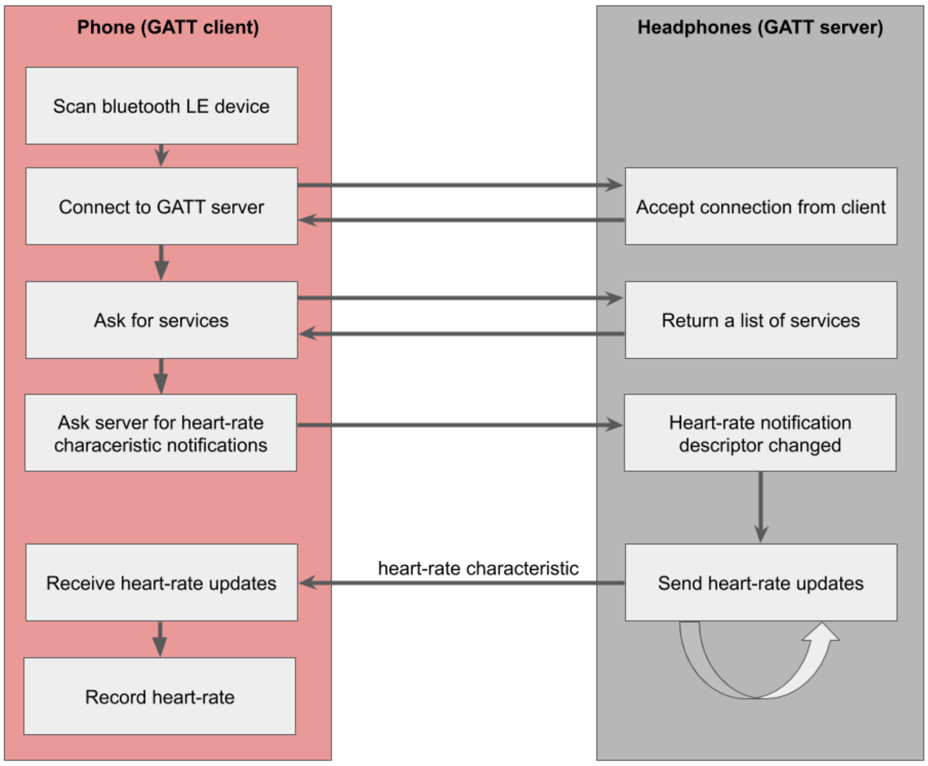

Phone (GATT client)
Headphones (GATT server)
Scan bluetooth LE device
Connect to GATT server
Accept connection from client
Ask for services
Return a list of services
Ask server for heart-rate characeristic notifications
Heart-rate notification descriptor changed
Receive heart-rate updates
heart-rate characteristic
Send heart-rate updates
Record heart-rate

Figure 3.8: Retrieving Data from Headphones

3.6.2 Sensor Fusion

Once I had gathered data from the earbuds, I evaluated it and found a problem, see section 4.6.1 for details. In essence, the issue I found was that the earbuds would lose the heartbeat signal due to the sensor changing position during a run. Particularly annoyingly, the earbuds report a heart-rate of zero whenever this happens.

To rectify this, I will implement a sensor-fusion technique. This technique makes use of a Kalman filter. The idea is it takes measurements from the earbuds, which we know are accurate, and then in periods where the heart-beat signal drops out we guide the output using the less accurate PPG from the wrist-watch.

Chapter 4

Evaluation

In this chapter, I describe the steps I took in order to evaluate both functionality and performance of my project. I go through all the implementation stages - gathering data, filtering data, motion artefact reduction, heart-rate calculation, post-processing and dealing with earbuds - for each commenting on their validity and performance.

4.1 Gathering Data

4.1.1 Synchronising Signals

In section 3.1.4, I detail an algorithm which allows me to calculate the difference in start-times between different recordings from a watch and from an ECG. Initially I use cross-correlation of accelerometer signals to determine the time at which the signals are the most similar, however this is quite ineficient, so I also develop an improved algorithm, referred to as informed cross correlation from now on.

In this section, I will evaulate both the validity and the performance of the solutions. To prove validity of imformed cross correlation, I compare the time difference it predicts to the actual time difference calculated by cross correlation, and then take the average absolute error over all recorded runs, in seconds.

To evaulate performance of informed cross correlation I run the algorithm on a sample set of accelerometer data, 500 times, using the python module timeit. This module avoids some easy mistakes that can be made when measuring execution times in python, for example it uses *time.perf_counter()* to measure time, which only measures execution time, not time spent sleeping.

Informed cross correlation takes an additional parameter, g which describes how many seconds to skip whilst calculating the initial cross correlation. I will test the algorithm with different values of this parameter, so I can determine which gives the best peformance whilst stil being valid.

The results of my test are displayed below

Algorithm	Average Absolute Error (s)	Time for 500 iterations (s)
Cross correlation	0	112.227
Informed g = 0.1s	0	17.935
Informed g = 0.25s	0	11.089
Informed g = 0.5s	0	8.648
Informed g = 1s	0.137	7.069

The first result I draw from this table is that my informed algorithm leads to a massive improvement in performance. Even the slowest informed runtime is 6 times faster than the naive implementation. Next, it can be seen that the runtime of the informed algorithm increases as the size of g decreases. This makes sense as the smaller the size of g, the fewer cross-correlation values we calculate.

In terms of validity, the table shows that the total absolute error of the informed algorithm is 0, for all values of g up to $g = 1$s. As g increases, the likelihood that the initial scan misses the optimal solution increases because the step size is too large.

In conclusion, I have discovered that the informed algorithm is capable of giving a valid solution, although if the value of g rises too high the solution may no longer be valid. In addition, I have discovered that it gives a much quicker solution, which gets quicker as g rises. For safety, I would choose $g = 0.25$s, which ensures the result will be valid, whilst still providing a ten-fold performance increase.

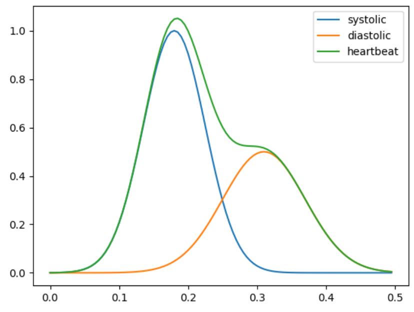
4.2 Filtering

To test filters, I will run two different tests. Firstly I will test how much noise the filter can remove alongside how much the information we want is actually retained. Additionally I will test how the speed of the filters varies, as in a heart-rate calculation algorithm we want to run the filters at speed.

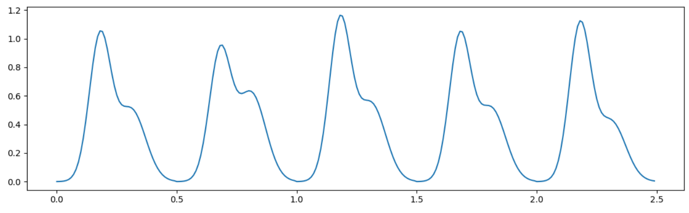
To test the quality of the filter I devised a test where I simulate a possible PPG signal without noise, then add noise. The noise I add will be random amount of high and low frequency noise, as we get in an actual PPG signal. Then to calcualte the quality of the filter, I filter the signal, and compare the ouput to the clean signal, using cross-correlation.

To simulate PPG signals, I found that two Guassians could be used [2]. One Gaussian represents the higher systolic peak, and the other represents the lower diastolic peak. See figure 4.1a for an example of the clean signal my simulation produces. As heart-rate is relatively consistent during running, I can assume the peaks are in approximately the same position, with slight variation. I simulate a heart-rate of 120 bpm, by repeating the pattern heart-beat every 0.5 seconds. In each beat I randomly vary the amplitude of both peaks, as would happen with an actual heart-beat. See figure 4.1b for a simulated heart-beat of 5 beats.

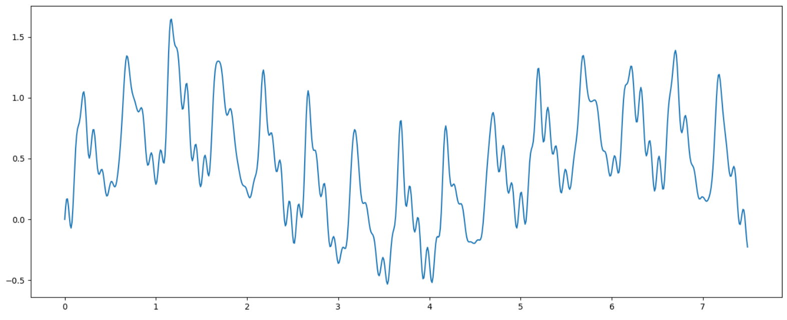
To represent the noise, I use random frequency and amplitude sin-waves at frequencies which would be considered noise (below 30 bpm and above 240 bpm). The results of adding this noise to a signal can be seen in figure 4.1c.



(a) A single simulated beat.



(b) Multiple simulated heart-beats.



(c) Simulated heart-beats with noise.

Figure 4.1: Simulating a noisy heart-beat.

Next, I must test the speed of the algorithms. To test this, I run the filter on a sample run from which I have collected 10 minutes of data. I run the test 500 times and take an average over the time taken.

I test the Chebyshev filter and the Butterworth filter with different parameters, in order to decide between the two of them, and additionally what parameters are optimal, in terms of both speed and validity.

4.2.1 Butterworth Filter

I test the speed by running the filter 1000 times using python's *timeit.timeit()* function, which provides the accurate time the computer has spent evaluating a given piece of code. I run the test on the same 35 minute PPG I recorded on a run.

The results of the timing test can be seen in figure 4.3. As can be seen in the graph, execution time increases almost linearly with filter order. Therefore, when choosing a filter it is important to choose the lowest order which provides sufficient filtering, so as not to be too expensive.

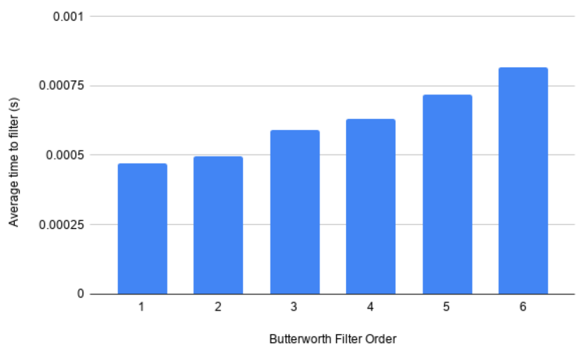


Figure 4.2: The effect of Butterworth filter order on execution time.

4.2.2 Chebyshev Type II Filter

I run the timing test again on the Chebyshev type II filter, taking the average amount of time taken to filter the same 35 minute segment as before. The results are plotted in figure ??. Firstly, there is a clear linear correlation between increasing filter order and increased execution time. This implies we should choose the lowest filter order which is feasible. Secondly, note that the Chebyshev filter takes over double the time as the Butterworth filter for each order. If there is no significant filtering performance improvement to using the Chebyshev filter, we should therefore use the Butterworth filter.

4.3 Motion Artefact Reduction

In this section I talk about the work done to evaluate the effectiveness of my motion artefact reduction techniques. I discuss the value of different approaches in terms of how effective they are at removing motion noise, and in terms of how computationally

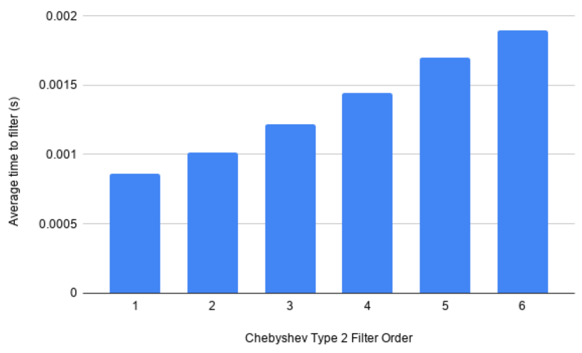


Figure 4.3: The effect of Chebyshev filter order on execution time.

expensive they are. I first detail the tests I ran, followed by the results of these tests on the different algorithms.

I will test the algorithms on different levels of motion corruption, to determine what solution is most effective for each level. I divide the data into thirty second segments of PPG data, labelled with one of three noise levels. I assign the noise level based on the speed at which that run was taking place, as the greater the speed, the more motion noise. I assign the noise levels as follows:

- Low - running speed 3 km/h or below
- Medium - running speed between 3 km/h and 10 km/h
- High - running speed 10 km/h or above

Now, to test my motion filtering algorithm, for each PPG data segment, I run the algorithm on it. With the results, I calculate the heart-rate that my heart-rate calculate algorithm would compute. I use my standard deviation minimisation algorithm from **??** I then compare this to the ground truth which is the ECG data associated with that segment, to get an error number.

I test two of my motion filtering algorithm - Least Mean Squares (LMS) and Normalised Least Mean Squares (NLMS). Both of these have a parameter called step-size, so I will compare both of them with a variety of step sizes to determine which is optimal for each class of motion noise.

4.3.1 LMS

4.3.2 NLMS

4.4 Heart Rate Calculation

To evaluate my Heart-Rate calculation, I establish the order of the algorithms in theory, followed by investigating how they actually perform when given data. I also establish the effectiveness of the algorithms, when given a clean PPG signal. I use a clean signal as I am assuming the filtering is done elsewhere, leaving a clean signal.

For the clean data, I have recorded a section of data using the wristwatch and ECG where I do not move. I then pass it through a Butterworth filter to remove noise out of the valid range. I do not attempt to remove motion noise, as there should be none given the lack of movement.

Finally I compare the result of the heart-rate calculation to the actual heart-rate measure provided by the ECG.

4.5 Post Processing

4.6 Earbuds

In this section, I evaluate the performance of the earbuds alone, and then evaluate some of the fixes I propose to issues identified in the evaluation.

The first step I took when evaluating the earbuds was to go on a fast run, which would introduce much motion noise on the watch. You can see the heart-rate from the earbuds plotted against the hear-rate I received from the ECG in Figure 4.4.

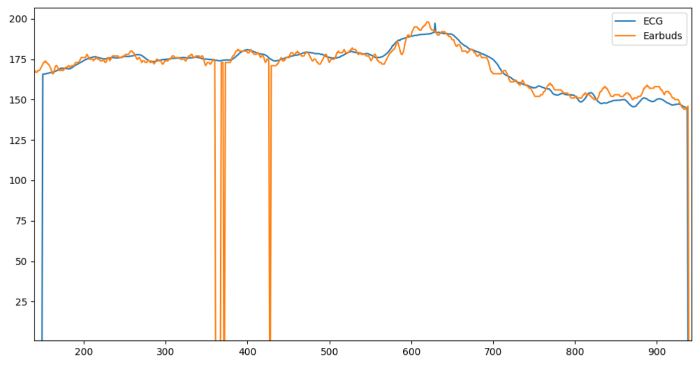


Figure 4.4: Plot of Earbud Heart-rate against ECG.

4.6.1 Problem

The problem with the earbuds is that they occasionally lose contact with the ear. This can happen as a combination of sweating and motion dislodges the earbud. In addition, I found when running I would often have to stop and adjust the earbud as I suspected it was going to fall out.

The earbuds are quite sensitive to location, so if they become dislodged or are rotated, the reported heart-rate drops to zero. This can be seen in Figure 4.4. Where the heart-rate is tracking accurately, but then suddenly just drops to zero.

Chapter 5

Conclusion

Bibliography

[1] John Allen, John R Frame, and Alan Murray. Microvascular blood flow and skin temperature changes in the fingers following a deep inspiratory gasp. *Physiological measurement*, 23(2):365, 2002.

[2] Rohan Banerjee, Avik Ghose, Anirban Dutta Choudhury, Aniruddha Sinha, and Arpan Pal. Noise cleaning and gaussian modeling of smart phone photoplethysmogram to improve blood pressure estimation. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 967–971, 2015.

[3] Wikimedia Commons. File:puls oxymeter.jpg — wikimedia commons, the free media repository, 2013. [Online; accessed 21-February-2020].

[4] Mohamed Elgendi. On the analysis of fingertip photoplethysmogram signals. *Current cardiology reviews*, 8(1):14–25, 2012.

[5] J. Kim, T. Lee, J. Kim, and H. Ko. Ambient light cancellation in photoplethysmogram application using alternating sampling and charge redistribution technique. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 6441–6444, Aug 2015.

[6] Scott Michael, Kenneth S Graham, and Glen M Davis. Cardiac autonomic responses during exercise and post-exercise recovery using heart rate variability and systolic time intervalsa review. *Frontiers in physiology*, 8:301, 2017.

[7] Hangsik Shin. Ambient temperature effect on pulse rate variability as an alternative to heart rate variability in young adult. *Journal of clinical monitoring and computing*, 30(6):939–948, 2016.

[8] B. Widrow, J. R. Glover, J. M. McCool, J. Kaunitz, C. S. Williams, R. H. Hearn, J. R. Zeidler, J. Eugene Dong, and R. C. Goodlin. Adaptive noise cancelling: Principles and applications. *Proceedings of the IEEE*, 63(12):1692–1716, Dec 1975.

[9] R. W. C. G. R. Wijshoff, M. Mischi, and R. M. Aarts. Reduction of periodic motion artifacts in photoplethysmography. *IEEE Transactions on Biomedical Engineering*, 64(1):196–207, Jan 2017.

[10] L. B. Wood and H. H. Asada. Noise cancellation model validation for reduced motion artifact wearable ppg sensors using mems accelerometers. In *2006 International*

Conference of the IEEE Engineering in Medicine and Biology Society, pages 3525–3528, Aug 2006.

Appendix A

Latex source

A.1 diss.tex

```
% Template for a Computer Science Tripos Part II project dissertation
\documentclass[12pt,a4paper,twoside,openright]{report}
\usepackage[pdfborder={0 0 0}]{hyperref}    % turns references into hyperlinks
\usepackage[margin=25mm]{geometry}  % adjusts page layout
\usepackage{graphicx}  % allows inclusion of PDF, PNG and JPG images
\usepackage{verbatim}
\usepackage{pdfpages}
\usepackage{caption}
\usepackage{subcaption}


\raggedbottom                           % try to avoid widows and orphans
\sloppy
\clubpenalty1000%
\widowpenalty1000%

\renewcommand{\baselinestretch}{1.1}    % adjust line spacing to make
                                        % more readable

\begin{document}

\bibliographystyle{plain}


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Title


\pagestyle{empty}

\rightline{\LARGE \textbf{Charlie Maclean}}

\vspace*{60mm}
\begin{center}
\Huge
\textbf{Synthesis of Heart-Rate Detection Methods} \\[5mm]
Computer Science Tripos -- Part II \\[5mm]
King's College \\[5mm]
\today  % today's date
\end{center}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Proforma, table of contents and list of figures

\pagestyle{plain}
```

```
\chapter*{Proforma}

{\large
\begin{tabular}{ll}
Name:               & \bf Charlie Maclean                       \\
College:            & \bf King's College \\
Project Title:      & \bf Synthesis of Heart-Rate Detection Methods  \\
Examination:        & \bf Computer Science Tripos -- Part II, July 2020  \\
Word Count:         & \bf TODO\footnotemark[1] \\
Project Originator: & Dr Robert Harle                   \\
Supervisor:         & Dr Robert Harle                   \\
\end{tabular}
}
\footnotetext[1]{This word count was computed
by \texttt{detex diss.tex | tr -cd '0-9A-Za-z $\tt\backslash$n' | wc -w}
}
\stepcounter{footnote}


\section*{Original Aims of the Project}

To research and implement the detection of heart rate from smartwatch
sensors. To investigate the effectiveness of a selection of filters and
peak finding algorithms. To use accelerometer data to find motion artifacts
within the data, and compare methods of removing these artifacts.


\section*{Work Completed}

All that has been completed appears in this dissertation.

\section*{Special Difficulties}

Learning how to incorporate encapulated postscript into a \LaTeX\
document on both Ubuntu Linux and OS X.

\newpage
\section*{Declaration}

I, Charlie Maclean of King's College, being a candidate for Part II of the
Computer Science Tripos, hereby declare that this dissertation and the work
described in it are my own work, unaided except as may be specified below,
and that the dissertation does not contain material that has already been
used to any substantial extent for a comparable purpose.

\bigskip
\leftline{Signed [signature]}

\medskip
\leftline{Date [date]}

\tableofcontents

\listoffigures

\newpage
\section*{Acknowledgements}

This document owes much to an earlier version written by Simon Moore
\cite{Moore95}.  His help, encouragement and advice was greatly
appreciated.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% now for the chapters

\pagestyle{headings}
```

```
\chapter{Introduction}

Elite runners have historically used heart rate to provide an accurate
measure of fitness, and allow them to train more effectively. [Expand on uses
of heart rate]. Previously, Electrocardiography (ECG) chest straps have been
used to measure heart rate, by detecting the electrical signals controlling
the expansion and contraction of the heart. They are accurate devices however
often prohibitively expensive, and hence inaccessible to casual runners.

In recent years, a new technology has emerged - Photoplethysmogram (PPG)
- light is directed at the skin, and sensors measure how much blood vessels
scatter it. PPG sensors are cheaper than ECG sensors, and hence are
available in a variety of products, particularly smartwatches. This
innovation has bought a new wave of advanced training and monitoring onto
the wrists of any runner.

Switching from ECG to PPG is not without flaws though - ECG sensors return a
clean signal, as opposed to PPG signals which are contaminated with
noise. [More details about noise].

In particular when running, the motion can cause blood velocity to
change, and the sensor can slip across the skin \cite{Wijshoff17},
\cite{Wood06}. These result in a distortion to the PPG signal, known as a
motion artifact (MA). Fortunately, smartwatches contain other sensors, such
as accelerometers and gyroscopes which can be used to predict the presence of
MAs, and hence compensate for them.

My task is to research and develop a heart rate detection algorithm
for smartwatches worn during running.

\chapter{Preparation}

This chapter details the steps I took to determine how to develop my
implementation. It first describes photoplethysmography, the method with which heart-beats are
detected on watches. Next,
I give an overview of the algorithms I will use to extract the heart-rate from
the PPG signal: filters, peak-detectors and motion artefact removers. Finally, I
outline my development methodology.

\section{Photoplethysmography (PPG)}

To develop an algorithm to track heart-rate on wrist-watches it is first important to
understand how watches track heart activity. PPG is a technique where light is
used to detect the volume of blood in veins. In hospitals, this is used in
finger pulse oximeters (Figure \ref{fig:fingerppg}) to record the heart-beat
of patients. These work by transmitting light on one side of the finger, and
then measuring how much light is received on the other side of the finger.
The amount of light which permeates through the finger is related to how much
blood is currently in the veins.


\begin{figure}[h!]
\centerline{\includegraphics[width=0.3\textwidth]{figs/fingerppg.jpg}}
\caption{Finger pulse oximeter. Image source \cite{wiki:fingerppg}.}
\label{fig:fingerppg}
\end{figure}

With watches, we cannot transmit light on one side, and
receive light on the other side, as the wrist is far too large. Hence, instead
of monitoring the absorption of light by the skin, we monitor the reflection
of light by the skin. A light is shone into the wrist, and sensors nearby
monitor how much is reflected back. When the wrist is full of blood, more
light is reflected back as blood scatters lights.

We know now how the signals are recorded, but we have not yet explored the
signal that is actually received from this technique. In Figure
```

```
\ref{fig:typicalppgsignal} we see a
clean PPG recording. There are two peaks in the data - a sytolic peak and a
diastolic peak. The systolic peak represents the point at which the heart has
beat - and hence pushed blood through the body. The challenge is to find this
peak.

\begin{figure}[h!]
\centerline{\includegraphics[width=0.7\textwidth]{figs/typicalppgsignal.jpeg}}
\caption{Example PPG signal. Image and annotations from \cite{elgendi12}.}
\label{fig:typicalppgsignal}
\end{figure}

\chapter{Implementation}

\section{Gathering Data}

In this section I detail the process of developing an
application to record PPG data on a Wear OS watch. This turned out to be more
complicated than I presumed due to interesting issues inherent with developing for a small
wearable with limited power.

\subsection{Wear OS Development}

\paragraph{Language Choices}

The watch application was developed using the official IDE for Android
development - Android Studio. I programmed in a language I'd never used before
- Kotlin. Kotlin is an open source language based on Java, which aims to
reduce boilerplate code, adds null-safety, and remains interoperable with
Java, allowing libraries written for Java to be used in Kotlin.

\paragraph{Structure}

The program was split into classes as in Figure \ref{fig:classes}. The
behaviour of each class is as follows:

\begin{itemize}
\item \emph{MainActivity} - central class responsible for starting,
stopping and saving the recordings.

\item \emph{SensorListener} - interface which overrides the Android
\emph{SensorEventListener}. Contains methods which
are called by the WearOS system whenever new sensor data comes
in. Additionally contain methods which save the received data.
We need one child for each sensor as each different sensor
provides different data, and must be saved in a different
format. So we have the following children:

\begin{itemize}
\item \emph{PpgListener}
\item \emph{AccelerometerListener}
\item \emph{RotationListener}
\end{itemize}

\end{itemize}

When the user presses start recording, \emph{MainActivity} registers each
\emph{SensorListner} with the OS so they receive any sensor changes.

When the user presses stop recording, \emph{MainActivity} unregisters the
listners and gets each one to save their recording.

\paragraph{Storing Recordings}

It was critical that I could easily import the recordings into any program, and
hence I chose to export the data into a CSV text file. Each \emph{Listener}
class uses a \emph{CSVWriter} object from library \emph{opencsv} to write to a
```

```
unique file for each sensor, within a directory chosen by \emph{MainActivity}.
For example, at the end of the recording in the directory
\emph{YYYY-MM-DD/HH.MM.SS/} we have files \emph{ppg.csv},
\emph{accelerometer.csv} and \emph{rotation.csv}.

\paragraph{Interface}

I chose a simple yet functional interface to allow the user to start and stop
a recording. The interface changes colour when recording, which makes it very
easy to verify the recording is happening. Pictures of interface in figure
\ref{fig:interface}.

\paragraph{Problems}

After I'd developed the application and was testing it out I found two key
issues which I will explain here.

\subparagraph{Power Saving}

Power is an enormous issue on wearable devices, as the form factor drastically
constrains the size of the battery. Hence developers have explicitly designed
Wear OS to limit power usage of any application as much as possible.
As part of these optimisations Wear OS automatically suspends any application
if it thinks they are not being used. While this is useful for most users, my
application needs to run continuously without interruption.

Initially in testing there were long periods of time where no sensor values
were being recorded, due to the application being suspended.

To fix this issue, a wake lock was included. Once a wake lock is acquired, the
OS does not suspend the application. I added one which is acquired every time
a recording is started, and released when the recording ends.

\subparagraph{Sampling Rate}

When I subscribe to values from the PPG sensor and accelerometer sensors, the
API asks for a parameter - sampling period, which is the amount of delay we
want between sensor readings being delivered. In practise, Android only uses
this number as a hint - values may be provided at a higher or lower rate.

Additionally, Android does not expose the actual rate at which sensor values
are being provided to the OS. The sensors which are placed within devices
always have a constant sampling rate. Sensors write to a very short buffer,
which Android then reads from at a rate it decides. This means although we
have a reliable sensor, Android occasionally misses readings.

I record the data with a sampling period of 0 - which signifies that I want
updates as soon as they are available. However, Android is not guaranteed to
provide this. I mitigate this by noting the time each recordings is provided to the
application. Then, I define sampling rate to be equal to the time difference
between the first and last sample, divided by the number of samples.

So, I have created an application which allows the user to record
accelerometer and PPG values at a constant sampling rate, then save them to
the watch.

\subsection{Uploading Recordings}

Now data has been recorded, the values must be uploaded somewhere to enable
later analysis. I created a server which would run on a Raspberry Pi that the
watch could connect to over the local network. The files would be passed to
the server, and from there the files are uploaded to Google Drive where they
can be accessed anywhere. In this section I go over the details of this
implementation. See figure \ref{fig:upload} for the overall uploading process.

\begin{figure}[tbh]
\centerline{\includegraphics[width=\textwidth]{figs/upload.png}}
```

```
\caption{The overall process of uploading files.}
\label{fig:upload}
\end{figure}

\subsubsection{Changes to Wear OS Application}

First a 'sync' button was included in the interface, which users can press to
trigger the sending of files to the server.

Then I added the logic to be triggered when the button is pressed. This logic
iterates through every file in the recordings directory and performs a HTTP
PUT request with them. I used\ldots

\subsection{Raspberry Pi Access Point}

To connect the watch to the server I needed to create a local access point on the
Raspberry Pi. The
packages I used to do this were \emph{hostapd} and \emph{dnsmasq}.

The package \emph{hostapd} is used to setup the access point, so I used it to configure
the name you can find the network from, the password of the network and the
channel it can be accessed on.

The package \emph{dnsmasq} is used to setup the DHCP service, assigning IP
addresses to devices. I set it up to assign the range of adresses from 192.168.0.1 to
192.168.0.255 on the same interface as \emph{hostapd} is broadcasting over.

\subsubsection{Flask Web Application on Raspberry Pi}

Now, to setup the server I used Flask, a Python Web App framework. I chose
this over other frameworks like Django as it provides a flexible framework
which is easy to understand and deploy.

I created an application \emph{server.py} which
accepts PUT requests containing a file from URL \emph{/upload/path} where
\emph{path} is the location we want to place the file. Then, the server
makes the directories up to \emph{path} if they don't exist, placing the file
in this directory. Unlike in a normal server implementation, I do not secure
this file upload process. I am aware there are a number of security issues
concerned with leaving a file upload with arbitrary path options, for example
a user might be able to upload a file in the parent direction by placing a PUT
request for \emph{/upload/../../example}. However, I do not worry about these
attacks as the access point is secured by \emph{hostapd} and only the watch
will be connected.

Once the file has been saved, I upload the new file to google drive, by
calling \emph{gdrive.py}, which I detail below.

\subsubsection{Python Scripts to Upload to Google Drive}

To manage the process of uploading files to Google Drive, I created a new
application \emph{gdrive.py}. To upload to Google Drive using Python, I used
the library \emph{googleapiclient}, which was a surprisingly complicated API,
as I will explain here.

First, to connect to Google servers, the API must authenticate to ensure that
it has permission to access a user's Drive. The service requires a credential
token which expires periodically. Hence, I store the credentials in a binary
file using library \emph{pickle}, and when I need to access them, I
refresh them if they have expired.

Then, I must actually upload a file. This process is complicated as Google
Drive does not use a path system to store files. Instead, it is a set of files
each with a unique IDs. A folder is simply a file with a \emph{folder} file
type, and a collection of files associated with it. Furthermore, as files are
not referenced by filename, it is possible to have multiple files in the same
folder with exactly the same filename.
```

```
The Google Drive API for Python only contains two methods which are useful for
my application:

\begin{itemize}
\item \emph{list(parent)} which lists the files within folder
\emph{parent}.

\item \emph{create(file, metadata)} which creates a \emph{file} with
\emph{metadata}. Note that as folders
are just files, this is also how folders are created, the
fact a file is a folder is specified within \emph{metadata}.
\end{itemize}

Using these methods, I define three methods within \emph{gdrive.py}, as follows:

\begin{itemize}
\item \emph{createFolder(name, parent)} creates folder by creating a
file with metadata specifying the type as a folder.

\item \emph{getOrCreatePath(path)} is used to obtain a 'path' from Google
Drive. It works by iterating through each level of the path,
creating the necessary folders if they don't already exist,
then entering them.

\item \emph{uploadFile(filepath)} created the path up to the file
given by filepath, and then upload the file to the folder.
\end{itemize}

Now, I have created a system which allows the user to upload recordings from
the watch to Google Drive via my server. And thus, I have completed the data
recording process. The data is recorded on the Wear OS app, and then with a
click transmitted to a Google Drive so it can be analysed.

\subsection{Synchronising Signals} \label{section:sync}

Once I have collected the PPG signal from the watch, and have recorded the ECG
from the chest, I need a method to synchronise the signals, such that they
start at the same time and we can compare them accurately.

I set out to produce a solution with the following properties:
\begin{itemize}
\item Able to synchronize signals with \(\pm0.3\)s accuracy. With a
heartbeat of 200 bpm this represents being within a heartbeat.
This is an acceptable level of delay, as heart rate is
averaged over several beats anyway.

\item Can synchronize signals given the two recordings are started
within two minutes of each other. This constraint is helpful
as it prevents wasted time searching through the signal.

\item Does not use the clocks built into the device. We must assume
the clock within the ECG is unreliable. Additionally, the two
devices may be synchronized to different time, and hence could
be out by any amount of time.
\end{itemize}

Given I know both devices have an accelerometer, I realised I could ask the
wearer to move in some motion which is picked up by both devices. Then, I
could compare the two accelerometer signals in order to discover the
movement. Then comparing the starting times of the two signals would enable
calculation of the time difference between them.

The type of motion I chose was important, as it had to be easy to explain to
the wearer, but also provide sufficient motion for it to be identifiable
against normal motion. I decided jumping was appropriate -
I would ask the wearer to hold the watch close to their chest and jump three
```

```
to five times.

The cross correlation of two signals \(f\) and \(g\) is a measure of
similarity as a function of the displacement between them. It is simple to
calculate, as a sum of the products between the samples of \(f\) and of \(g\)
displaced by \(n\).

To synchronise the accelerometer, I developed the following three step
algorithm:
\begin{enumerate}
\item Normalize signals, by moving to zero mean,

\item Crop PPG signals to two minutes, crop ECG signals to four
minutes, compute cross correlation with \(f=\) PPG
acceleration, \(g=\) ECG acceleration.

\item Compute the other way - crop ECG to two minutes, PPG to four
minutes, compute cross correlation with \(f=\) ECG
acceleration, \(g=\) PPG acceleration. I compute both ways, to
ensure that we can synchronise regardless of which device
started recording first.

\item Find the maximum cross correlation across each of the
calculations.








\end{enumerate}

\subsection{Informed Synchronisation}

Running cross correlation on two signals is an effective way to compare their
similarity, however it is not efficient. Cross correlation has complexity
\(O(n^3)\). In this section I develop a new algorithm which uses knowledge of
the problem to speed up the cross correlation calculation.

By graphing the cross correlation of the acceleration signals, we can draw
some useful features that help us to work out a more optimal solution. See
figure \ref{fig:cross-correlation} for an example of absolute cross correlation plotted
between the two accelerometer signals over time. It is clear that there is one
peak which is above all the others, where the jumps line up perfectly.
However, in addition we can see that there is a significant rise in cross
correlation around the peak, where we are seeing peaks due to one or two of
the jumps aligning. This is useful as we hence do not need to scan at each
possible time difference, instead we can to a lower resolution initial scan.
After this initial scan is done, we find the peak, and check each possible
time difference around the lower resolution peak to find the actual peak.

\begin{figure}[h!]
\centerline{\includegraphics[width=0.8\textwidth]{figs/cross-correlation.png}}
\caption{Absolute Cross Correlation over time. }
\label{fig:cross-correlation}
\end{figure}

The algorithm I designed takes two parameters: step size (\emph{g}) and
correct range (\emph{r}). Step size is the resolution of the initial scan, so the
higher it is the more likely we are to find the peak but the longer the
algorithm takes. Correct range gives the range of the time period we want to
scan once we have found an initial peak, in order to find the actual peak.

\section{Filtering}

PPGs produce a large amount of noise from various sources:
```

```
\begin{itemize}
\item Light pollution. Both DC and AC lights in the environment can
infiltrate the sensor, affecting the PPG signal \cite{kim15}.
\item Temperature. As temperature increases, so does the volume of
blood, which will be picked up by the PPG sensor
\cite{shin16}.
\item Breathing. The change in pressure accosiated with respiration
causes variations in the flow of blood, and hence can be seen
by the PPG sensor \cite{allen02}.
\end{itemize}

In order to remove much of this noise, we can use filters to remove
frequencies we know are irrelevant. We know the heartbeat can vary from
30 to 220 beats per minute, and hence we would like to disregard any noise
outside of this range. In this section, I will introduce the concept of
filters, and describe two filters useful for PPG signals.

\begin{figure}[h!]
\centerline{\includegraphics[width=0.8\textwidth]{figs/filter.png}}
\caption{Diagram showing filter characteristics}
\label{fig:filterdiag}
\end{figure}

Figure \ref{fig:filterdiag} displays an example filter magnitude response
diagram. This plots the amount of gain applied to each frequency. A gain of
zero means the signal is removed, a gain of one means the signal is unchanged.
The characteristics displayed are as follows.

The passband is the range of frequencies we would like to remain. In an ideal
filter, there is no loss within the passband.

Passband ripple describes the variation in amplitude \emph{within} the
passband. An ideal filter will have no passband filter, such that all
frequencies within the passband are permitted equally.

The transition width is the frequency range between the start and stop band.
Ideally, this is zero, such that frequencies outside of the passband are
instantly reduced.

The stopband is the range of frequencies we wish to remove. An ideal filter
completely removes all frequencies within the stopband.

The ideal filter, as described above, is impossible - and hence we have a
variety of filters which compromise between the desirable characteristics.
Following this, I detail two of these compromises - the Butterworth filter,
and the Chebyshev filter.


\subsection{Butterworth Filter}

The Butterworth filter aims to minimize passband ripple, at the expense of a
larger transition width. To define a Butterworth filter, multiple parameters
are used, which I will describe further here.

\begin{figure}[h]
\centerline{\includegraphics[width=0.8\textwidth]{figs/butter-order-comparison.png}}
\caption{Comparison of Butterworth filter orders}
\label{fig:butterworth-order}
\end{figure}

Order (\(N\)) controls the transition width - the higher the order, the lower
the transition width. This is desirable - we want to keep the passband as
large as possible, and transition quickly to the stopband. However, it comes
at the cost of computation complexity, as higher order filters take longer to
apply. See figure \ref{fig:butterworth-order} for a plot displaying the effect
of different orders on a filter's response.
```

```
Critical frequency gives the frequency at which we want gain to drop below
\(1/\sqrt2\) from the passband. Hence we use this parameter to define the
point from which we wish to cut out noise. Figure \ref{fig:butterworth-order}
uses critical frequencies 0.4 and 4 hz, which correspond to 24 and 240 bpm, a
reasonable heart rate range.

Type describes which frequencies we cut out. There are four different
types of Butterworth filters, as follows:

\begin{itemize}
\item Lowpass - allow frequencies below the critical frequency.

\item Highpass - allow frequencies above the critical frequency.

\item Bandpass - combination of lowpass and highpass - given two
frequencies, we want the passband to be between those
frequencies.

\item Bandstop - given two frequencies, put the stopband between them,
leaving the passband outside those frequencies.
\end{itemize}

We will be using a bandpass filter, as there is low and high frequency noise
we wish to remove.

\subsection{Chebyshev Filter}

The Chebyshev filter aims to reduce the transition width as much as possible,
but to do this it introduces increased ripple.

\begin{figure}[h]
\begin{subfigure}{.5\textwidth}
  \centering
  \includegraphics[width=\linewidth]{figs/cheby1-rp-comparison.png}
  \caption{Chebyshev Type 1}
  \label{fig:cheby1rs}
\end{subfigure}%
\begin{subfigure}{.5\textwidth}
  \centering
  \includegraphics[width=\linewidth]{figs/cheby2-rp-comparison.png}
  \caption{Chebyshev Type 2}
  \label{fig:cheby2rs}
\end{subfigure}
\caption{Comparison of Chebyshev filter rs values}
\label{fig:chebyrs}
\end{figure}

The filter is defined with similar parameters to a Butterworth filter - order,
critical frequencies, and type all have very similar meanings. However,
there are type 1 and 2 Chebyshev filters, with different aims. Additionally,
there is a new parameter \emph{rs}. I will explain these differences here.

Type 1 Chebyshev filters introduce passband ripple, whereas type 2 Chebyshev
filters introduce stopband ripple. Figure \ref{fig:chebyrs} shows how these
differences manifest in the response of a second order Chebyshev filter.

The parameter \emph{rs} is defined as the maximum ripple permitted below unity
gain. Higher \(rs\) results in more ripple for a type 1 filter, but less
ripple for a part 2 filter, as is demonstrated in figure \ref{fig:chebyrs}.
Introducing more ripple is undesirable as leads to more signal distortion,
however can be advantageous as it reduces the transition width.

\section{Motion Artefact Reduction}

In this section I explore a method which can be used to reduce the effect of
Motion Artefacts (MAs) on the PPG Signal.
```

```
\subsection{Adaptive Noise Cancellation}

Adaptive Noise Cancellation (ANC), as described by Widrow et al.
\cite{Widrow75}, is a technique which allows us to remove noise from a signal,
given we have another signal correlated to the noise in some way. In our
situation, we know MAs are correlated to motion, so we can use the
accelerometer to remove MAs. This technique is extremely useful, as it can
still produce good results when the MAs are at the same frequency as the heart
beat, even if the MAs have a larger amplitude.

%TODO: https://ieeexplore.ieee.org/abstract/document/7867772 this may be
%helpful


\begin{figure}[tbh]
\centerline{\includegraphics[width=\textwidth]{figs/ANC-concept.png}}
\caption{ANC overall flow}
\label{epsfig}
\end{figure}


An overview of the algorithm follows. We have signal from heartbeat \(s\) that
we want to figure out, but this is contaminated by noise from MAs \(n_0\).
We assume the PPG sensor reading \(p\) is such that \(p=s+n_0\). Additionally,
we have accelerometer sensor readings \(n_1\). We filter \(n_1\), producing
signal \(y\) and subtract this from \(p\), to produce \(z=p-y\). Our aim is to
adjust the filter, such that \(y\) is as close to \(n_0\) as possible.

The type of filter we adjust is known as a Finite Impulse Response (FIR)
filter. An \(N\)th order filter is defined by \(N\) weights.
These weights are convolved with the signal, hence filtering it.

The adaptive filter aims to choose a filter such that the power of the output
\(E[z^2]\) is minimized. Given \(s\) is uncorrelated with \(n_0\) and \(n_1\), and \(n_0\)
is correlated with \(n_1\), it can be proven that minimizing the output power
is equivalent to finding \(z=s\).

There are multiple algorithms which exist in order to adaptively filter a
signal. I'll look at two, first describing the least mean squares (LMS) algorithm from
Widrow and Hoff. LMS first initializes all of the filter weights to 0. At each
step, it finds the gradient of the mean square output \(E(z^2)\).
Each weight's gradient describes what would happen to the signal if we kept
using that weight value. So if the gradient of a weight is positive we know
that using that weight again would increase the mean squared output. Hence, at
the end of each step we shift the weights down if the gradient is positive,
and visa versa if the gradient is negative.

LMS takes a parameter - step size \(\mu\), which scales how much we move the
weights by. The equation used to update weights is \[w_{n+1}=w_n-\mu \nabla
E(y^2)[n]\]
where \(W_n\) is the \(n\)th filter weight, and \(\nabla E(y^2)[n]\) is the
\(n\)th weight gradient.

Step size must be chosen very carefully, as if it is too large there is a risk
of LMS missing the solution as it overcompensates for the gradient. If it is
too small, then the algorithm becomes too expensive as we need many more
iterations. The situation is made worse as the ideal step size changes with
respect to the power of the input, making it very difficult to choose a step
size which gives stability.

To solve these stability issues, NLMS was introduced. It is the same as LMS, except you normalize the input power each
iteration. This makes choosing a step size which gives stability a lot easier.

So now we have a stable adaptive filter, which will allow us to tune a filter
that will remove motion noise from our PPG signal.

\section{Heart Rate Calculation}
```

```
Now we have removed noise from the signal, we should have a signal
representing the volume of blood in veins within the arm. As the heart beats,
it sends a wave of blood through the body, forcing more blood through these
veins. Hence, peaks in the PPG signal correspond to heart beats.

There are multiple approaches to finding heart-rate from our filtered
signals, and I will detail a few here.

\subsection{Local Maxima}

If we can find the peaks in the signal, we can count them up and divide by the
time period we count those peaks within, giving us a heart rate. One way we
can define peaks is as local maxima.

The local maxima in a signal are defined as samples which are larger than both
neighbouring samples. Finding local maxima is an easy problem to solve -
intuitively, we can create an algorithm which iterates through the samples,
comparing each sample to it’s neighbours, which has complexity \(O(n)\).

\subsection{Peak-Peak Standard Deviation Minimization}

Finding the local maxima is a naive approach that doesn’t adapt to any problem
specific knowledge. For example, we know heart-rate will not climb above 220
bpm, however the prior solution could report 300 bpm. Additionally, it might
find multiple local peaks around a single heart-beat.

Ideally, we want to include knowledge about what a normal heart-beat looks
like in our detection algorithm. One thing we can do is look at peak-peak
intervals - the time that passes between heart beats. Surprisingly, this is
normally not consistent - a healthy heart actually beats with some
variation. However, when running it has been found that the heart beats with
very little variation \cite{michael17}, and hence in our case we can assume that the heart
beats regularly. The aim of this method is to use this assumption, along with
minimum and maximum heart-rate to better detect heart-beats.

The method begins by calculating an initial set of peaks, by calculating the
rolling mean of the signal, and marking points above that mean as peaks. Then,
we compute the standard deviation of the time between these peaks (peak-peak
interval). Then, we iteratively increase the rolling mean and again calculate
the peaks, along with
the standard deviation of their intervals. The idea is every time we increase the rolling mean we
exclude more peaks. Then at the end, the peaks we actually select are the peaks which give a reasonable
heart-rate (in the range 20-240 bpm), and have the smallest standard deviation.

So, we have developed an algorithm that attempts to find a heart-beat with
regular peak-peak intervals, and is within a reasonable range.

\section{Post Processing}

\subsection{Kalman Filter}

So far, we have not considered previous heart-rate readings when considering
the current calculation. This is naive, as heart-rate over the course of a run
follows a very fixed pattern.

Figure    shows heart-rate over a steady run. It is clear initially heart-rate
ramps up until it reaches a ’steady-state’. From here on the heart-rate
remains pretty constant, not responding to slight changes in exertion.
Finally, the heart-rate decreases rapidly once the run has completed.

The Kalman filter is an algorithm which can compensate for noisy and
inaccurate measurements such as what we get from the PPG signal. It consists
of two phases:
\begin{enumerate}
\item Prediction step - predict the current heart rate based on the
previous heart rate, and some model. In this step we can
```

```
incorporate the accelerometer as a reasonable estimator for
the change in running effort.

\item Update step - take the actual reading from the PPG signal and combine
with the output of the prediction step.
\end{enumerate}

The Kalman filter relies on modelling current heart-rate as a Gaussian
distribution with it's standard deviation representing the certainty of the
current estimate.

Kalman gain is the parameter used to weight the measurements against the
prediction. A higher gain assigns more trust to the measurements, meaning the
filter will respond quicker, a low gain produces more smoothing.



\section{Earbuds}

As an extension, I wanted to collect data from a new kind of PPG-enabled
device: earbuds. In theory, these devices are less effected by motion noise
as they are not affected by arm movement. Additionally, the ear has a rich set
of veins from which we can gather PPG. For the purposes of this project I used
a set of Jabra elite sport earbuds, which have a PPG sensor under the right
bud - see figure \ref{fig:jabra}.

\begin{figure}[tbh]
\centerline{\includegraphics[width=0.4\textwidth]{figs/jabra.png}}
\caption{Positioning of the PPG sensor on Jabra elite sport earbuds.}
\label{fig:jabra}
\end{figure}

In this section, I discuss how I obtained data from the earbuds, and then how
I processed the data I received.

\subsection{Gathering data}

To collect data, I connected the earbuds to the phone, and then data is
sent through the Bluetooth Light Energy (LE) protocol. There are a few key
pieces of terminology, which I explain here:

\paragraph{Generic Attribute Profile (GATT)} describes the overall methods
with which data values are transported by Bluetooth LE.

\paragraph{GATT Server} is a device which stores and distributes information. In
our case the earbuds are a GATT client.

\paragraph{GATT Client} is a device which requests information from a GATT
server, in our case a mobile phone.

\paragraph{Characteristics} are single data points, for example in our case a
heart-rate integer. Characteristics also have a number of descriptors
describing the value.

\paragraph{Services} are logical group of characteristics, for example the
heart-rate service contains the heart-rate measurement characteristic along
with sensor position characteristic.

\paragraph{Characteristic Value Notifications} are sent from client to server
to indicate that the client would like to receive updates when a certain
characteristic is changed. This is how frequently changing values are passed
over Bluetooth LE, as it is more power efficient than having the client
periodically request updates.

Figure \ref{fig:gatt-flow} shows the flow I aimed to introduce in my data collection program to run
on the phone.
```

```
\begin{figure}[tbh]
\centerline{\includegraphics[width=0.8\textwidth]{figs/gatt-flow.png}}
\caption{Retrieving Data from Headphones}
\label{fig:gatt-flow}
\end{figure}

\subsection{Sensor Fusion}

Once I had gathered data from the earbuds, I evaluated it and found a problem,
see section \ref{section:earbud-problem} for details. In essence, the issue I
found was that the earbuds would lose the heartbeat signal due to the sensor
changing position during a run. Particularly annoyingly, the earbuds report a
heart-rate of zero whenever this happens.

To rectify this, I will implement a sensor-fusion technique. This technique
makes use of a Kalman filter. The idea is it takes measurements from the earbuds, which
we know are accurate, and then in periods where the heart-beat signal drops
out we guide the output using the less accurate PPG from the wrist-watch.



\chapter{Evaluation}

In this chapter, I describe the steps I took in order to evaluate both
functionality and performance of my project. I go through all the
implementation stages - gathering data, filtering data, motion artefact
reduction, heart-rate calculation, post-processing and dealing with earbuds -
for each commenting on their validity and performance.

\section{Gathering Data}

\subsection{Synchronising Signals}

In section \ref{section:sync}, I detail an algorithm which allows me to
calculate the difference in start-times between different recordings from a
watch and from an ECG. Initially I use cross-correlation of accelerometer signals to
determine the time at which the signals are the most similar, however this is
quite ineficient, so I also develop an improved algorithm, referred to as
informed cross correlation from now on.

In this section, I will evaulate both the validity and the performance of the
solutions. To prove validity of imformed cross correlation, I compare the
time difference it predicts to the actual time difference calculated by cross
correlation, and then take the average absolute error over all recorded runs,
in seconds.

To evaulate performance of informed cross correlation I run the
algorithm on a sample set of accelerometer data, 500 times, using the python
module timeit. This module avoids some easy mistakes that can be made when
measuring execution times in python, for example it uses
\emph{time.perf\_counter()} to measure time, which only measures execution
time, not time spent sleeping.

Informed cross correlation takes an additional parameter,
\emph{g} which describes how many seconds to skip whilst
calculating the initial cross correlation. I will test the algorithm with different
values of this parameter, so I can determine which gives the best peformance
whilst stil being valid.

The results of my test are displayed below
\begin{center}
\begin{tabular}{ | c | c | c | }
\hline
Algorithm & Average Absolute Error (s) & Time for 500
iterations (s)\\
\hline
Cross correlation & 0 & 112.227 \\
Informed g = 0.1s & 0 & 17.935 \\
Informed g = 0.25s & 0 & 11.089 \\
Informed g = 0.5s & 0 & 8.648 \\
Informed g = 1s & 0.137 & 7.069  \\
\hline
   \end{tabular}
\end{center}

The first result I draw from this table is that my informed algorithm leads to
a massive improvement in performance. Even the slowest informed runtime is 6
times faster than the naive implementation. Next, it can be seen that the
runtime of the informed algorithm increases as the size of g decreases. This
makes sense as the smaller the size of g, the fewer cross-correlation values
we calculate.

In terms of validity, the table shows that the total absolute error of the
informed algorithm is 0, for all values of \emph{g} up to \emph{g} = 1s. As
\emph{g} increases, the likelihood that the initial scan misses the optimal
solution increases because the step size is too large.

In conclusion, I have discovered that the informed algorithm is capable of
giving a valid solution, although if the value of \emph{g} rises too high the
solution may no longer be valid. In addition, I have discovered that it gives
a much quicker solution, which gets quicker as \emph{g} rises. For safety, I
would choose \emph{g} = 0.25s, which ensures the result will be valid, whilst
still providing a ten-fold performance increase.


\section{Filtering}

To test filters, I will run two different tests. Firstly I will test how much
noise the filter can remove alongside how much the information we want is
actually retained. Additionally
I will test how the speed of the filters
varies, as in a heart-rate calculation algorithm we want to run the filters at
speed.

To test the quality of the filter I devised a test where I simulate a possible
PPG signal without noise, then add noise. The noise I add will be random
amount of high and low frequency noise, as we get in an actual PPG signal.
Then to calcualte the quality of the filter, I filter the signal, and compare
the ouput to the clean signal, using cross-correlation.

To simulate PPG signals, I found that two Guassians could be used
\cite{Banerjee15}. One Gaussian represents the higher systolic peak, and the
other represents the lower diastolic peak.
See figure \ref{fig:sim-beat} for an example of the clean signal my simulation
produces. As heart-rate is relatively
consistent during running, I can assume the peaks are in approximately the
same position, with slight variation. I simulate a heart-rate of 120 bpm, by
repeating the pattern heart-beat every 0.5 seconds. In each beat I randomly
vary the amplitude of both peaks, as would happen with an actual heart-beat.
See figure \ref{fig:sim-beats} for a simulated heart-beat of 5 beats.

To represent the noise, I use random frequency and amplitude sin-waves at
frequencies which would be considered noise (below 30 bpm and above 240 bpm).
The results of adding this noise to a signal can be seen in figure
\ref{fig:sim-beats-noise}.

\begin{figure}
\centering
\begin{subfigure}[b]{0.4\textwidth}
\centerline{\includegraphics[width=\textwidth]{figs/sim-beat.png}}
\caption{A single simulated beat.}
\label{fig:sim-beat}
\end{subfigure}
\hfill
```

```
\begin{subfigure}[b]{0.7\textwidth}
\centerline{\includegraphics[width=\textwidth]{figs/sim-beats.png}}
\caption{Multiple simulated heart-beats.}
\label{fig:sim-beats}
\end{subfigure}
\hfill
\begin{subfigure}[b]{0.8\textwidth}
\centerline{\includegraphics[width=\textwidth]{figs/sim-beats-with-noise.png}}
\caption{Simulated heart-beats with noise.}
\label{fig:sim-beats-noise}
\end{subfigure}
\caption{Simulating a noisy heart-beat.}
\end{figure}

Next, I must test the speed of the algorithms. To test this, I run the filter
on a sample run from which I have collected 10 minutes of data. I run the test
500 times and take an average over the time taken.

I test the Chebyshev filter and the Butterworth filter with different
parameters, in order to decide between the two of them, and additionally what
parameters are optimal, in terms of both speed and validity.

\subsection{Butterworth Filter}

I test the speed by running the filter 1000 times using python's
\emph{timeit.timeit()} function, which provides the accurate time the computer
has spent evaluating a given piece of code. I run the test on the same 35
minute PPG I recorded on a run.

The results of the timing test can be seen in figure \ref{fig:butter-time}. As can
be seen in the graph, execution time increases almost linearly with filter
order. Therefore, when choosing a filter it is important to choose the lowest
order which provides sufficient filtering, so as not to be too expensive.

\begin{figure}[tbh]
\centerline{\includegraphics[width=0.5\textwidth]{figs/butter-time.png}}
\caption{The effect of Butterworth filter order on execution time.}
\label{fig:butter-time}
\end{figure}

\subsection{Chebyshev Type II Filter}

I run the timing test again on the Chebyshev type II filter, taking the average amount
of time taken to filter the same 35 minute segment as before. The results are
plotted in figure \ref{fig:cheby2-time}. Firstly, there is a clear linear
correlation between increasing filter order and increased execution time. This
implies we should choose the lowest filter order which is feasible. Secondly,
note that the Chebyshev filter takes over double the time as the
Butterworth filter for each order. If there is no significant filtering
performance improvement to using the Chebyshev filter, we should therefore use the
Butterworth filter.


\begin{figure}[tbh]
\centerline{\includegraphics[width=0.5\textwidth]{figs/cheby2-time.png}}
\caption{The effect of Chebyshev filter order on execution time.}
\label{fig:butter-time}
\end{figure}


\section{Motion Artefact Reduction}

In this section I talk about the work done to evaluate the effectiveness of my
motion artefact reduction techniques. I discuss the value of different
approaches in terms of how effective they are at removing motion noise, and in
terms of how computationally expensive they are. I first detail the tests I
ran, followed by the results of these tests on the different algorithms.
```

```
I will test the algorithms on different levels of motion corruption, to
determine what solution is most effective for each level. I divide the
data into thirty second segments of PPG data, labelled with one of three
noise levels. I assign the noise level based on the speed at which that run
was taking place, as the greater the speed, the more motion noise. I assign
the noise levels as follows:

\begin{itemize}
\item Low - running speed 3 km/h or below
\item Medium - running speed between 3 km/h and 10 km/h
\item High - running speed 10 km/h or above
\end{itemize}

Now, to test my motion filtering algorithm, for each PPG data segment, I run
the algorithm on it. With the results, I calculate the heart-rate that my
heart-rate calculate algorithm would compute.
I use my standard deviation minimisation algorithm from \ref{}
I then compare this to the
ground truth which is the ECG data associated with that segment, to get an
error number.

I test two of my motion filtering algorithm - Least Mean Squares (LMS) and Normalised Least Mean
Squares (NLMS). Both of these have a parameter called step-size, so I will
compare both of them with a variety of step sizes to determine which is
optimal for each class of motion noise.

\subsection{LMS}


\subsection{NLMS}


\section{Heart Rate Calculation}

To evaluate my Heart-Rate calculation, I establish the order of the algorithms
in theory, followed by investigating how they actually perform when given
data. I also establish the effectiveness of the algorithms, when given a clean
PPG signal. I use a clean signal as I am assuming the filtering is done
elsewhere, leaving a clean signal.

For the clean data, I have recorded a section of data using the wristwatch and ECG
where I do not move. I then pass it through a Butterworth filter to remove
noise out of the valid range. I do not attempt to remove motion noise, as
there should be none given the lack of movement.

Finally I compare the result of the heart-rate calculation to the actual
heart-rate measure provided by the ECG.

\section{Post Processing}

\section{Earbuds}

In this section, I evaluate the performance of the earbuds alone, and then
evaluate some of the fixes I propose to issues identified in the evaluation.

The first step I took when evaluating the earbuds was to go on a fast run,
which would introduce much motion noise on the watch. You can see the
heart-rate from the earbuds plotted against the hear-rate I received from the
ECG in Figure \ref{fig:earbud-fast}.

\begin{figure}[tbh]
\centerline{\includegraphics[width=0.7\textwidth]{figs/earbud-fast-graph.png}}
\caption{Plot of Earbud Heart-rate against ECG.}
\label{fig:earbud-fast}
\end{figure}
```

```
\subsection{Problem} \label{section:earbud-problem}

The problem with the earbuds is that they occasionally lose contact with the
ear. This can happen as a combination of sweating and motion dislodges the
earbud. In addition, I found when running I would often have to stop and
adjust the earbud as I suspected it was going to fall out.

The earbuds are quite sensitive to location, so if they become dislodged or
are rotated, the reported heart-rate drops to zero. This can be seen in Figure
\ref{fig:earbud-fast}. Where the heart-rate is tracking accurately, but then
suddenly just drops to zero.

\chapter{Conclusion}


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the bibliography
\addcontentsline{toc}{chapter}{Bibliography}
\bibliography{refs}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the appendices
\appendix


\chapter{Latex source}

\section{diss.tex}
{\scriptsize\verbatiminput{diss.tex}}

\chapter{Makefile}

\section{makefile}\label{makefile}
{\scriptsize\verbatiminput{makefile.txt}}

\section{refs.bib}
{\scriptsize\verbatiminput{refs.bib}}


\chapter{Project Proposal}

\includepdf[pages=-,pagecommand={},width=\textwidth]{proposal.pdf}

\end{document}
```

Appendix B

Makefile

B.1 makefile

```
# This is the makefile for the Part II demonstration dissertation
#
# Note that continuation lines require '\' and
# that a TAB character preceeds any shell command line

.DELETE_ON_ERROR:

# Rules for building LaTeX documents (see Unix Tools course)
%.pdf %.aux %.idx: %.tex
        pdflatex -halt-on-error $<
        while grep 'Rerun to get ' $*.log ; do pdflatex $< ; done
%.ind: %.idx
        makeindex $*
%.bbl: %.aux
        bibtex $*
# Rules for exporting xfig diagrams into PDF or EPS
%.pdf: %.eps
        epstopdf --outfile=$@ $<
%.eps: %.fig
        fig2dev -L eps $< $@
%.pdftex %.pdftex_t: %.fig
        fig2dev -L pdftex_t -p $*.pdftex $< $*.pdftex_t
        fig2dev -L pdftex $< $*.pdftex

help:
        @echo
        @echo "USAGE:"
        @echo
        @echo "make                display help information"
        @echo "make proposal.pdf   format the proposal as PDF"
        @echo "make diss.pdf       format the dissertation as PDF"
        @echo "make all            make proposal.pdf and diss.pfd"
        @echo "make view-proposal  format and view the proposal"
        @echo "make view-diss      format and view the dissertation"
        @echo "make count          display an estimated word count"
        @echo "make pub            put demodiss.pdf onto your homepage"
        @echo "make clean          delete all intermediate files"
        @echo

view-%: %.pdf
        ( okular --unique $< || evince $< ) &

diss.pdf: figs/cuarms.pdf figs/diagram.pdf makefile.txt diss.bbl

makefile.txt: makefile
```

```
        expand makefile >makefile.txt

count:
        detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w

all: proposal.pdf diss.pdf

pub: diss.pdf
        rsync -t $+ $(HOME)/public_html/demodiss.pdf

clean:
        rm -f *.aux *.log *.err *.out
        rm -f *~ *.lof *.toc *.blg *.bbl
        rm -f makefile.txt

distclean: clean
        rm -f figs/*.pdf proposal.pdf diss.pdf
```

B.2 refs.bib

```
@INPROCEEDINGS{Wood06,
author={L. B. {Wood} and H. H. {Asada}},
booktitle={2006 International Conference of the IEEE Engineering in Medicine and Biology Society},
title={Noise Cancellation Model Validation for Reduced Motion Artifact Wearable PPG Sensors Using MEMS Accelerometers},
year={2006},
pages={3525-3528},
doi={10.1109/IEMBS.2006.260359},
ISSN={1557-170X},
month={Aug},}

@ARTICLE{Wijshoff17,
author={R. W. C. G. R. {Wijshoff} and M. {Mischi} and R. M. {Aarts}},
journal={IEEE Transactions on Biomedical Engineering},
title={Reduction of Periodic Motion Artifacts in Photoplethysmography},
year={2017},
volume={64},
number={1},
pages={196-207},
doi={10.1109/TBME.2016.2553060},
ISSN={1558-2531},
month={Jan},}

@ARTICLE{Widrow75,
author={B. {Widrow} and J. R. {Glover} and J. M. {McCool} and J. {Kaunitz} and C. S. {Williams} and R. H. {Hearn} and J. R.
journal={Proceedings of the IEEE},
title={Adaptive noise cancelling: Principles and applications},
year={1975},
volume={63},
number={12},
pages={1692-1716},
doi={10.1109/PROC.1975.10036},
ISSN={1558-2256},
month={Dec},}

@INPROCEEDINGS{Kim15,
author={J. {Kim} and T. {Lee} and J. {Kim} and H. {Ko}},
booktitle={2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)},
title={Ambient light cancellation in photoplethysmogram application using alternating sampling and charge redistribution tec
year={2015},
volume={},
number={},
pages={6441-6444},
doi={10.1109/EMBC.2015.7319867},
ISSN={1558-4615},
month={Aug},}
```

```
@article{allen02,
  title={Microvascular blood flow and skin temperature changes in the fingers following a deep inspiratory gasp},
  author={Allen, John and Frame, John R and Murray, Alan},
  journal={Physiological measurement},
  volume={23},
  number={2},
  pages={365},
  year={2002},
  publisher={IOP Publishing}
}

@article{shin16,
  title={Ambient temperature effect on pulse rate variability as an alternative to heart rate variability in young adult},
  author={Shin, Hangsik},
  journal={Journal of clinical monitoring and computing},
  volume={30},
  number={6},
  pages={939--948},
  year={2016},
  publisher={Springer}
}

@article{michael17,
  title={Cardiac autonomic responses during exercise and post-exercise recovery using heart rate variability and systolic ti
  author={Michael, Scott and Graham, Kenneth S and Davis, Glen M},
  journal={Frontiers in physiology},
  volume={8},
  pages={301},
  year={2017},
  publisher={Frontiers}
}

@misc{ wiki:fingerppg,
   author = "Wikimedia Commons",
   title = "File:Puls oxymeter.jpg --- Wikimedia Commons{,} the free media repository",
   year = "2013",
   url = "https://commons.wikimedia.org/w/index.php?title=File:Puls_oxymeter.jpg&oldid=111119859",
   note = "[Online; accessed 21-February-2020]"
 }

@article{elgendi12,
  title={On the analysis of fingertip photoplethysmogram signals},
  author={Elgendi, Mohamed},
  journal={Current cardiology reviews},
  volume={8},
  number={1},
  pages={14--25},
  year={2012},
  publisher={Bentham science publishers}
}

@article{Banerjee15,
  title={Noise cleaning and Gaussian modeling of smart phone photoplethysmogram to improve blood pressure estimation},
  author={Rohan Banerjee and Avik Ghose and Anirban Dutta Choudhury and Aniruddha Sinha and Arpan Pal},
  journal={2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)},
  year={2015},
  pages={967-971}
}
```

Appendix C

Project Proposal

Charlie Maclean
King's
cm927

Diploma in Computer Science Project Proposal

Synthesis of Heart-Rate Detection Methods

December 18, 2019

Project Originator: Dr Robert Harle

Project Supervisor: *Dr Robert Harle*

Signature:

Director of Studies: *Dr Timothy Griffin*

Signature:

Overseers: *Dr Anil Madhavapeddy* and *Professor John Daugman*

Signatures: <*no need to obtain Overseers' signatures yourself*>

Introduction and Description of the Work

Heart-rate signals from watches are unreliable while exercising. Watches make use of photoplethysmography (PPG) sensors - sensors which detect the volume of blood in the skin and use variances in this to reconstruct a heart-rate. PPG sensors are preferred to the more accurate electrocardiogram (ECG) due to user comfort. However, the signals they provide are harder to process - I want to compare strategies to process these signals to extract heart rate.

There are several sources of noise within a PPG signal. There is often high frequency contamination caused by electrical interference or light from external sources. Additionally, there is a constant low frequency variation in the DC background of the signal, as a result of capillary density, blood volume and temperature variations.

In the context of running, motion caused by the arms swinging forward and back causes the sensor to slide along the skin, creating motion artifacts (MAs). These are particularly challenging as they can have a much larger amplitude than the pulse we are looking for. Additionally, they can be at the same frequency as the heart rate signal, making them challenging to filter out. Therefore, there is research [1][2] that suggests using accelerometer data in order to predict MAs. I will implement algorithms which find the MAs based on data from the accelerometer. Following this, I will look into the implementation of filters to remove these MAs.

In order to get accurate heart-rate measurements with which to compare the PPG signals, I will make use of an chest-mounted portable ECG. I will need to synchronize the signals received from the ECG and PPG, due to clock skew between the different devices. Potentially, there could be drift between the two clocks as well.

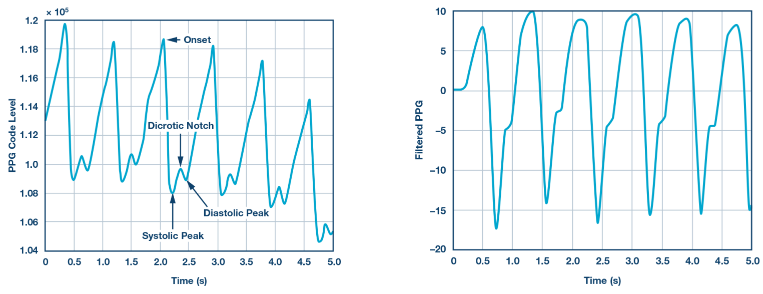


Figure 1: PPG signals before and after filtering [3]

Starting Point

Wearable Development

I will need to develop an application for the smart watch which will record PPG and motion signals without interruption. I will use Android Studio and Kotlin.

I have never used Kotlin before and have used Android Studio once before, but never to develop something for a wearable.

Digital Signal Processing

Manipulation of the PPG signals I receive will require much digital processing, and there are two languages I am considering using: MATLAB and Python.

I have never used MATLAB before, but I am familiar with Python.

Substance and Structure of the Project

Core

1. Developing a wearable application to capture PPG signals and motion data. I would like to develop this using Android Studio and Kotlin. I will need to make use of wake locks in order to ensure that the application can continuously record data.
2. Collecting data using a PPG-enabled watch and a portable ECG. I will record my heartbeat over the course of several runs.
3. Synchronizing signals. Implementation of an algorithm to synchronize the data output from the watch records to the data output from the portable ECG. I will have to develop a program to find the lag between the two signals.
4. Removing noise. Due to the various sources of noise, I will investigate potential low and high pass filters, to remove both high and low frequency disturbance.
5. Peak finding - implementing algorithms to find the actual beat given a clean PPG signal. Some potential algorithms are:
   (a) Adaptive threshold [4]
   (b) Wavelet transformation [5]
6. Finding MAs - implement an algorithm which uses the accelerometer in order to detect segments of the PPG which are likely to have been affected by motion.

7. Removal of MAs - implementation of filters to remove the previously detected MAs.

Possible Extensions

- Investigating PPG-enabled earbuds
  - Evaluation of the quality of heart-rate provided by earbuds
  - Exploring the potential to merge signals from a smartwatch and earbuds in order to provide a higher quality signal.
- There is research [6] to suggest gyroscope information is also helpful in filtering out MAs. I could include gyroscope data in my MA filtering technique.
- Comparing more heart-rate detection algorithms:
  - Digital filters
  - Adaptive filters
  - Singular value decomposition
  - Empirical mode decomposition
  - Spectrum analysis

References

[1] Z. Zhang. Photoplethysmography-based heart rate monitoring in physical activities via joint sparse spectrum reconstruction. *IEEE Transactions on Biomedical Engineering*, 62(8):1902–1910, Aug 2015.

[2] Z. Zhang, Z. Pi, and B. Liu. Troika: A general framework for heart rate monitoring using wrist-type photoplethysmographic signals during intensive physical exercise. *IEEE Transactions on Biomedical Engineering*, 62(2):522–531, Feb 2015.

[3] Foroohar Foroozan. Music-based algorithm for on-demand heart rate estimation using photoplethysmographic (ppg) signals on wrist, 2018.

[4] Ivaylo I. Christov. Real time electrocardiogram qrs detection using combined adaptive threshold. *BioMedical Engineering OnLine*, 3(1):28, 2004.

[5] Jake D. Campbell, Christopher G. Pretty, J. Geoffrey Chase, and Phillip J. Bones. Near-real-time detection of pulse oximeter ppg peaks using wavelet decomposition. *IFAC-PapersOnLine*, 51(27):146 – 151, 2018. 10th IFAC Symposium on Biological and Medical Systems BMS 2018.

[6] Alexander J. Casson, Arturo Vazquez Galvez, and Delaram Jarchi. Gyroscope vs. accelerometer measurements of motion from wrist ppg during physical exercise. *ICT Express*, 2(4):175 – 179, 2016. Special Issue on Emerging Technologies for Medical Diagnostics.

Success Criteria

The following should be achieved:

- Develop an application which records and stores PPG signals on a watch.
- Create program which synchronises ECG signals with PPG signals.
- Implement at least two filtering algorithms, demonstrate filtering works by displaying signals before and after filtering.
- Implement at least two peak finding algorithms, demonstrate they work by comparing peaks on the PPG signal to peaks on the ECG signal.
- Implement a MA detection algorithm.
- Be able to remove MAs, demonstrating that the peak finding algorithm is not affected by signals caused by motion.

Resources Required

I will use my own laptop. I will regularly backup my project to GitHub and an external HDD, so that I can recover data in the event of hardware failure. I accept full responsibility for this machine and and I have made contingency plans to protect myself against hardware and/or software failure.

I will use a smart watch, and a chest-mounted ECG sensor in order to collect data.

Timetable and Milestones

Weeks 1-2 (28/10/19 - 10/11/19)

- Project set-up:
  - Installation and setup of Android Studio.

  - Investigate and install an IDE for Python or MATLAB.
  - Use GitHub to set up a backup system for the project.
  - Learn to use Kotlin for the wearable app.
- Create a wearable app to record and store PPG and motion data, using Android Studio and Kotlin. Research into wake logs and interface design.
- Additionally, test application by recording data while running.

Weeks 3-4 (11/11/19 - 24/11/19)

- Collect data from runs.
- Build program to synchronize data between ECG and PPG (based on either heart rate variation or motion information).
- Researching and then implementing filters to remove both high and low frequency noise from the PPG signal.

Weeks 5-6 (25/11/19 - 08/12/19)

- Implement program to detect peaks in the signal.
- Begin writing dissertation document.

I will take the next week off.

Weeks 7-8 (16/12/19 - 29/12/19)

- Implement MA detection algorithm.
- Continue writing dissertation.

Weeks 9-10 (30/12/19 - 12/01/20)

- Implement MA removal algorithm.
- Continue writing dissertation.

Weeks 11-12 (13/01/20 - 26/01/20)

- Begin extension work, testing earbud HR detection against wrist-watches and portable ECGs.
- Continue writing dissertation.

Friday 31st January - Progress Report due

Weeks 13-14 (27/01/20 - 09/02/20)

- Extension work, testing whether earbud HR can be merged with wristwatch signal to create a more accurate output.
- Continue writing dissertation.

Weeks 15-16 (10/02/2020 - 23/02/2020)

- Extension work, including gyroscope feedback into the MA detection algorithm.
- Continue writing dissertation.

Weeks 17-18 (24/02/2020 - 08/03/2020)

- Extension work, including gyroscope feedback into the MA detection algorithm.
- Continue writing dissertation.

Weeks 19-20 (09/03/2020 - 22/03/2020)

- Writing dissertation.

Weeks 20-21 (23/03/2020 - 05/04/2020)

- Writing dissertation, aim to have first draft by second week of Easter holiday.

Weeks 22-26 (06/04/2020 - 08/05/2020)

- Continue improving dissertation.
- Leaving time for exam preparation.

Final deadline for dissertation 08/05/2020