University of Glasgow | School of Engineering

Photoplethysmography using the camera of a mobile device

Executed by, Mustafa Biyikli 2190523

Supervised by, Dr Bernd Porr

August 2020

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Mechatronics

Abstract

This project aimed to extend an existing mobile application, AttysECG, to implement additional features to perform real-time photoplethysmography (PPG) for heart rate detection written in Java. Image data was collected from the rear camera of a mobile phone running on Android OS 8.0. Images were sampled at 24Hz and processed to output the average RGB intensities of each frame in real-time. RGB data was converted to YUV format as it was found to contain more information and a 4th order Butterworth band-pass filter with cut-off frequencies at 0.5Hz and 5Hz was implemented to filter the signal. PPG data of a 24-year-old, healthy male subject was recorded by placing the camera directly in front of the index finger, close to hand, close to face and far from face. For each PPG recording, Attys hardware was used simultaneously to record real-time electrocardiography (ECG) data. Recorded signals were compared to find and calculate the correlations between the R-peaks and image data using Python. PPG signal obtained from the mobile camera was found to have a low signal-to-noise ratio (SNR), therefore, a Java matched filter was implemented using a time reversed characteristic template as its coefficients. This was found to be the optimal linear filter for maximising the SNR. Fingertip PPG achieved an average heartrate detection accuracy within ±0.5% of the equivalent ECG detection for measurements longer than 15 seconds. Measurements taken with an increased proximity were found to be very sensitive to ambient light and motion artefacts. Therefore, although it was possible to use the image data to measure the heart rate of a perfectly still subject with a device fixed in position under consistent lighting, the practical and accurate use of PPG for heart rate detection using a mobile device camera was found to be limited to close proximity.

Acknowledgments

I would like to present my special thanks and gratitude to my supervisor Dr Bernd Porr for providing me with the opportunity to develop an idea into a functional product. Additionally, I would like to thank him for the guidance and help he provided throughout the project on numerous occasions.

I would also like to take this opportunity to thank the University of Glasgow for providing me with the necessary resources whenever I needed.

Table of Contents

List of Figures

Glossary

AC – Alternating current

Aliasing – Misidentification of an electrical signal, introducing distortions or errors

API – Application programming interface

BPM – beat per minute

DC – Direct current

ECG – Electrocardiography

FIR – Finite impulse response

HRV – Heart rate variability

HSI – A standard colour format; hue (H), saturation (S), intensity (I)

IIR – Infinite impulse response

PPG – Photoplethysmography

RGB – A standard colour format; red (R), green (G), blue (B)

SNR – Signal-to-noise ratio

YUV – A standard colour format; luminance (Y) and chrominance (UV)

1 Introduction

This chapter contains the project background, the problem definition and sets the aims & objectives of the project. An introduction is made to methods employed and the equipment exploited to achieve these objectives and the report layout and structure is explained.

1.1 Project Background

This is a project proposed by Dr Bernd Porr with the description of using photoplethysmography (PPG) to perform heart-rate detection with the camera of a mobile device.

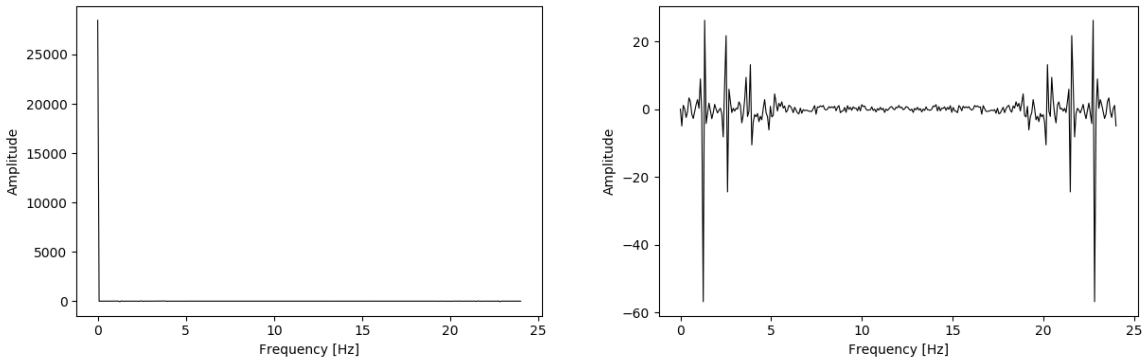
PPG is an optical measurement technique for detecting volumetric changes in blood circulation. The subtle changes in an individual's skin colour, caused by a cardiac pulse that are undetectable to human eye can be detected by an optical sensor placed on the surface of the skin [1]. Heart rate variability (HRV) analysed from momentary heart rate measurements of recorded electrocardiography (ECG) signals is an accurate and well known analysis for diagnosing several heart conditions [2]. Therefore, calculating momentary heart rate from ECG R-peaks is a crucial clinical assessment. PPG is an alternative to accurate momentary heart rate measurements by registering the volumetric changes of blood in a live body. The term "photoplethysmography" was first introduced in 1938 as a non-invasive optical technique that was capable for registering transcutaneous changes of blood volume in blood vessels [3] by Hertzman. The basic principle of PPG was described in 1937 by Hertzman & Spealman as an empirical observation of light being reflected from a living tissue obtaining a modulation in time at the heartbeat frequency [4].

As the skin is well perfused, detecting pulsatile components of the cardiac cycle becomes relatively easy, however, as the technique requires an optical sensor, the first device capable of performing PPG measurement was invented years after in 1972 [5]. Pulse oximeters using PPG for heart rate detection from patient's fingers became commercially available in 1983 and are being clinically used up to date.

Although the heart rate accuracy of the PPG devices was comparable to the ECG equivalent, they required specific hardware.

1.2 Problem Definition

The bulk of a PPG signal is caused by its DC component, however, the information related to the cardiac cycle is found in its AC component [6]. This is shown in Figure 1.



(a) Frequency Spectrum with DC component

(b) DC removed frequency spectrum

Figure 1, Frequency spectrum of a PPG signal recorded with a mobile phone camera

Utilising a PPG signal for accurate and reliable heart rate detection requires real-time digital filtering. Causal signal processing generally requires dedicated hardware that runs software written in fast and efficient compiled languages such as C or C++. Additionally, like other biological signals, PPG has a low signal-to-noise ratio which requires matched filtering to improve. Therefore, up until recent years, accurate and reliable PPG heart rate detection was limited to hardware and software that is specifically developed for that purpose, such as the pulse oximeters.

1.3 Solution Approach

With the advancements in nano technology and improved manufacturing techniques, handheld mobile devices are getting more and more powerful. Many of the newer devices running on both Android and IOS platforms provide their

users with more computing power than an average PC which makes them capable of performing causal signal processing. Additionally, a large majority of "smartphones" have built in optical sensors with flashes used for capturing images and videos. Combining the computational power of these devices with their built-in sensors and actuators allows for capturing clean PPG measurements which can be used for heart rate detection. FaceBEAT [1] and iPhysioMeter [7] have demonstrated the accuracy of performing heart rate detection with PPG on IOS platform running Swift code. Java can be used for performing the same measurement on Android platform by converting the information captured by the optical sensor into a standard digital colour format such as RGB. As a PPG signal is formed by the volumetric changes in blood, measuring for the changes in average red intensity from the skin of a live body would allow detecting blood passing through the surrounding blood vessels.

With an estimated number of 3.5 billion [8], smartphones being used globally, developing a simple and accurate heart rate detector is a solution to eliminating additional hardware.

1.3.1 ECG & PPG

Unlike PPG, ECG deals with the electrical impulses generated by the depolarisation and repolarisation of the myocardial fibres [9]. However, like the conventional methods of registering a PPG signal, ECG requires dedicated hardware with the addition of carefully positioned electrodes for registering the changes in the electrical potential difference. As ECG is the validated clinical standard for heart rate detection, understanding it is crucial before attempting to develop a comparable method utilising PPG. An ECG signal carries a large amount of information regarding the health of an individual's cardiovascular system within a PQRST(U) complex. However, the R-peak and the RR interval is the most important characteristic of an ECG signal for heart rate detection. This is shown in Figure 2.

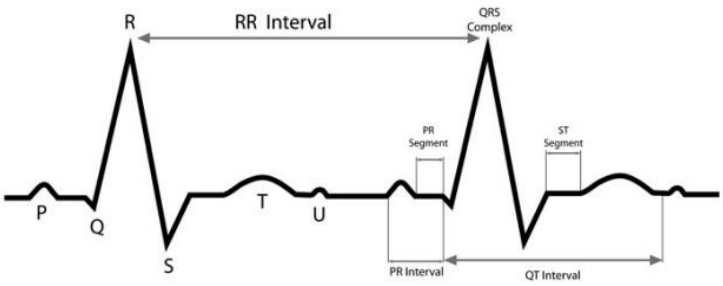


Figure 2, ECG signal characteristics [9]

Heart rate is calculated by recording the timestamps at the R peaks and finding the differences between the two. This can then be converted into standard units such as beats-per-minute (bpm). A DC removed PPG signal conveys information related to the systolic and diastolic blood pressure and comparing the two, similarities can be found as shown in Figure 3.

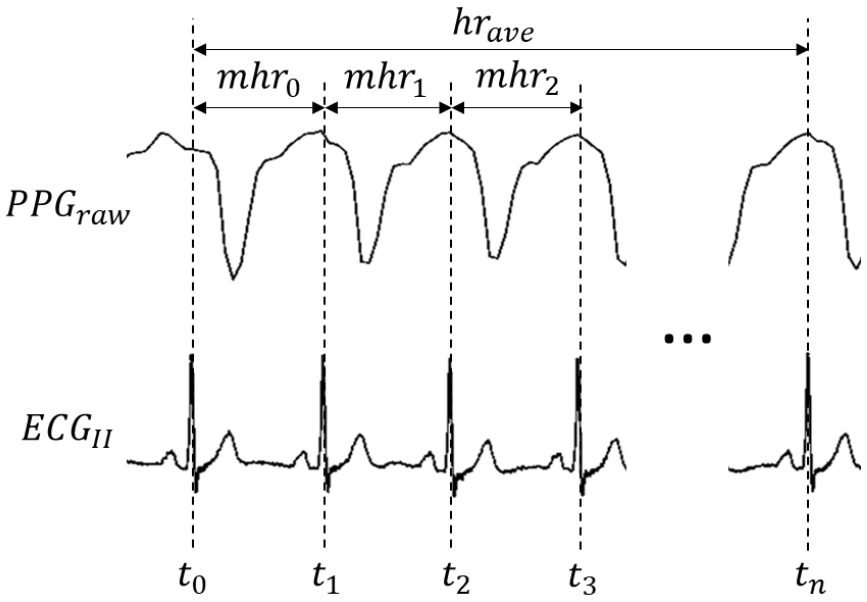


Figure 3, ECG & PPG signal comparison

The RR interval technique can be applied to the PPG signal by detecting its peaks and finding the time difference between the two. However, as PPG peaks are flatter than R-peaks, the possibility of bogus detections is much higher. Therefore, a custom matched filter needs to be implemented with coefficients taken from a carefully selected template to obtain peaks that are easily detectable.

1.4 Project Aim & Objectives

1.4.1 Aim

This project aimed to extend an existing mobile app, AttyECG, by developing additional features to perform real-time, PPG heart rate detection using the device camera.

1.4.2 Objectives

Previous studies have already validated the possibility of performing contact PPG where the user is asked to place their index finger over the device camera. The objectives of this project were to develop and implement an algorithm for accurate heart rate detection at the fingertip and investigate the possibility of performing non-contact PPG heart rate detection from an increased proximity.

1.5 Project Structure

This section details the structure of the chapters. This report details the development and engineering principles used to achieve the project aim. Methods of causal signal processing as well as efficient software development are discussed. XML and Java code used for mobile app development as well as Python code used for analysis and non-causal processing is provided to exploit the approach used to efficiently solve the engineering problem.

Subject Review introduces the existing methods of performing PPG for heart rate detection and discusses the effectiveness and differences between them. Thereafter, the challenges involved in performing both contact and non-contact PPG are detailed and discussed. Additionally, a review of programming methods and languages used and the differences in their performance is discussed.

Description of Project details the full design and development process including the mathematical and practical methods used for efficient signal processing. Assumptions and simplifications made while developing the mobile application as well as the methods used for testing the digital filters and the performance of the

application are detailed. This chapter involves the equations used when designing the FIR, IIR and matched filters for improving the SNR.

Results and Discussion presents the comparison of the FIR and IIR filters in terms of their filtering performance, power efficiency and delay. PPG and ECG data for contact and non-contact measurements are introduces and their accuracy and repeatability for heart rate detection is discussed. The algorithm developed for the application is presented in a block diagram. The limitations of contact and non-contact PPG methods are discussed. The robustness of the thresholding algorithm is introduced and discussed. Finally, the extended application is presented.

Conclusions provide a brief description of the project summarising the work done throughout the semester. A discussion of the remaining challenges in the field of smartphone PPG are discussed. Produced work is referenced back to the project aim and objectives stated at the beginning of the project and overall success of the project is rated. Additionally, methods that can be exploited for improving the work carried out in this project is introduced in this section. A pixel filter that can be used for further improving the SNR by removing non-skin related pixel out of the image is introduced and discussed.

Project Management Review compares the actual timescale of the project to the GANTT chart produced at the beginning of work. A discussion is made on why minor changes and additions were made to the initial plan. An updated GANTT chart reflecting the actual workflow carried out in the project is produced and provided for comparison.

APPENDIX provides the full extension code developed for the AttysECG mobile application including both Java and XML. Full application code can be found in https://github.com/MustafaBiyikli/AttysECG.git.

2 Subject Review

This chapter reviews the existing methods of heart rate detection with PPG and discussed the differences between them along with their effectiveness. Thereafter, methods of contact and non-contact PPG are contrasted.

2.1 Review of Existing Methods

Transmittance and reflectance methods are the existing procedures used for PPG measurements. Transmittance method directs a light source, actuator, onto living tissue where the photodetector, sensor, placed on the opposite side of the tissue measures the amount of light transmitted through. This method isn't very sensitive to motion artefacts, however, there are limited measurement sites on body for its use. Reflectance method does the same, however, the actuator and the sensor are placed on the same side of the living tissue and the light that is reflected from the tissue is measured. This greatly improves the amount of measurement sites available, however, makes the measurement very sensitive to motion artefacts. Figure 4 details how each of these methods work.

Figure 4, Transmittance & Reflectance methods for PPG measurement [10]

Several companies have developed and commercialised products capable of performing PPG heart rate detection using both described methods. Pulse oximeters using the transmittance method are clinically used and take measurements from the fingers of patients.

Several wearable wrist devices and smartwatches also provide heart rate detection features using the reflectance or the transmittance method such as the Apple Watch by Apple Inc. Other devices such as earlobe clips and rings have been developed for research and commercial purposes and all worked similar to the

previously mentioned devices. Studies have shown the importance of the anatomical site chosen for measurements as certain sites provided better perfusion values. These areas were identified to be the fingers, palm, face and ears as measurements taken from these sites resulted in a superior SNR compared to the rest of the anatomy [10].

Video plethysmography is a recent, contact or non-contact PPG measurement method. A video camera is used as the photodetector by converting the image data into RGB colour channels. This can be performed with the video recorder of a handheld device such as a smartphone camera and detects the reflected light from the skin [10]. Using this method, plethysmography signals can be measured remotely under ambient light. For non-contact measurements, green channel contains the most information due to a preoccupation peak caused by oxy-haemoglobin, nevertheless, red and blue channels still convey information regarding PPG [10]. Contact PPG requires the user place their index finger over the device camera and uses the built-in flashlight as the light source. Given that the device flashlight is white, most of the signal is contained in the red channel. iPhysioMeter [7] and FaceBeat [1] have already shown the world a working concept with the use of an efficient high-pass filter to remove the DC noise and a matched filter to improve the SNR. This type of PPG heart rate detection has already been commercialized by Azumio Inc and is available for smartphones running both on Android and IOS.

2.2 Performance Review of Existing Methods

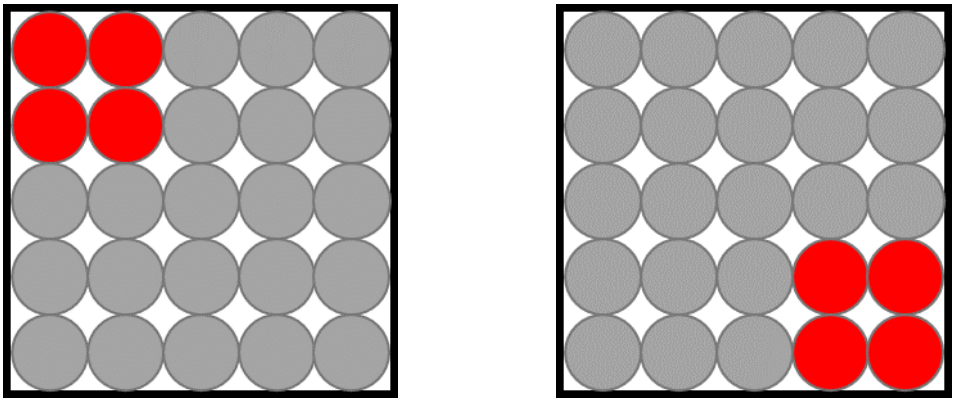
Previous studies have validated the robustness and performance of pulse oximeters [11] and they are clinically used around the world. Amongst the wearable devices capable of PPG, Apple Watch was found to achieve the most accurate heart rate measurements [12]. However, the only relevant comparison to this project would be the available smartphone applications.

To review the robustness and accuracy of existing methods using PPG heart rate detection, several applications available on Play Store and AppStore were compared to Attys ECG hardware. Cardiio and Instant Heart Rate by Azumio Inc performed the best, providing the user with a live plot of their PPG signal and an average heart rate detection within ±5% of Attys ECG. Therefore, contact PPG with a mobile camera is a robust and accurate method of heart rate detection. Previous studies have validated the possibility of performing non-contact PPG for heart rate detection [13], however, varying ambient light and the hardware stability proved to reduce the robustness of the method.

The best applications using these methods applied a high-pass filter to remove the DC noise and a matched filter to improve the SNR.

2.3 Review of Programming Methods

Android applications are typically developed in Java, however, C++ would be the optimal choice for real-time signal processing as it is a faster and more efficient language [14]. Previous studies have tested the performance of the languages by running identical sorting algorithms using both languages. C++ was proven to require 50% of the time taken by the equivalent Java program, additionally C++ programs were found to require 7 to 9 times less memory and 40 times less virtual memory [14]. Language performance is crucial as live video feed textures sampled at 24Hz to 30Hz need to be averaged for their colour intensity. On a digital screen, this is done by analysing the image pixel by pixel. Thereafter, the 8-bit RGB colours of each pixel is analysed and an average integer is calculated for each colour channel. The reason for this is illustrated in Figure 5.



(a) Red pixels on top left (b) Red pixels on bottom right

Figure 5, Pixels colour intensity analysis

Although both Figure 5 (a) & (b) have the same total red intensity, (a) has a collection of red pixels on its top left corner where (b) has red pixels on its bottom right corner. Therefore, a "for" loop is needed to go through the view, pixel by pixel, vertically and horizontally. Additionally, the loop needs to re-run every time the image is updated. On a live video feed capturing image at 30 frames per second, total loop execution time needs to be less than 30 milliseconds. Otherwise, a framerate drop will be experienced which will alter the sampling rate. The mathematical expression for colour averaging of an image is shown in equation (1).

$$\overline{s_n}(t_k) = \frac{1}{N_p} \sum_{i=r_0}^{r_f} \sum_{j=c_0}^{c_f} C(i,j) \qquad (1)$$

Where N is the total number of pixels in the image, r_0 and c_0 are the first row and column of the image, r_f and c_f are the last row and column of the image and $C(i,j)$ is the pixel value the corresponding component [15].

2.3.1 Filtering & Detection Methods

As the fundamental frequencies of a PPG signal is found within the 0.5Hz to 5Hz band [16], depending on the prefiltering done by the device operating system, either a band-pass or a high-pass filter needs to be applied. If a finite-impulse-response (FIR) filter is applied, the frequency resolution needs to be lower than 0.5Hz to capture the PPG fundamentals. This would determine the number taps, and consequently, the additional delay and processing power required.

Alternatively, an infinite-impulse-response (IIR) filter can be used to achieve a clean signal. Previous studies have validated the success of these filters and determined the 4th order Chebyshev II filters to be the optimal IIR filter for PPG signals [17]. The choice depends on the application and whether if the phase response can be non-linear.

Having filtered the signal, the typical practice is to apply a template matching process. This process uses a template that matches the desired signal. Thereafter, the signal is sample-by-sample multiplied with the coefficients of the template within a ring buffer. Although this process causes a further sample delay equivalent to half the total number of coefficients, it greatly improves the SNR. The matched signal can be fed to a square function to further improve the SNR and a suitable threshold is applied to detect the peaks. This is shown in Figure 6.

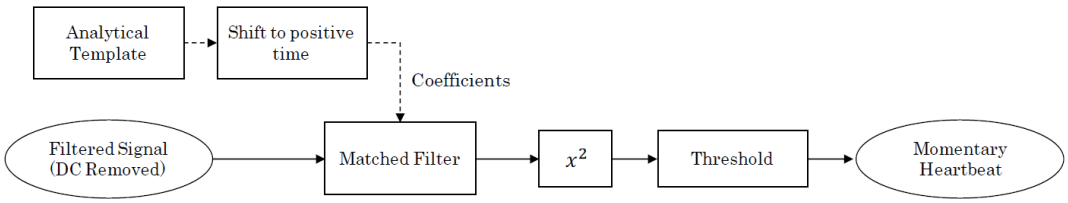


Figure 6, Typical PPG heart-rate detection process

A timestamp is taken from each peak that is detected and the momentary heart rate is calculated using equation (2).

$$hr_m = \frac{60}{\Delta t} \tag{2}$$

2.3.2 Camera2 API

This project uses the Camera2 API as the additional features were developed for API 26+ (Oreo and beyond) and Camera1 API was deprecated in API 21. To start a live video feed with the chosen API, the following parameters must be understood.

- CameraCaptureSession used for capturing and processing images from the device camera within the same session.

- CameraDevice that is the camera hardware connected to the Android device.
- CameraCharacteristics used for getting or setting the properties of the CameraDevice.
- CameraManager used for detecting and connecting to the available device cameras.
- CaptureRequest is the package that contains the settings and outputs needed to capture and image from the device camera. It also contains the sensors and actuators related to the camera device such as the flashlight.
- CaptureResult is the results of a single image captured from the sensor. It also contains information related to the configuration of related sensors and actuators such as the flashlight.

Additionally, a camera capture request requires user permission due to privacy reasons. Therefore, the permissions must be handled by adding the camera permission into the application manifest and handling them in the onCreate Java activity.

2.4 Typical Image Formats

As previously mentioned, to obtain a PPG signal, the image captured by the device camera must be converted into a colour model. Although there are many image colour-models available to use, RGB and YUV are the two main formats used for PPG signals. RGB defines colours in 8-bit to 32-bit per channel format, for red, green, and blue channels. Higher value for a channel means greater intensity of the colour corresponding to that channel. For certain applications, this information can be represented in hexadecimal format, however, the colour format remains the same. In 1970s, graphics researchers designed a colour model from pixel hue, saturation and intensity, HSI. It was designed to better align with human colour perception and is typically used for enhancing low-light images [18]. Although not a typical PPG colour model, HSI can be compared to RGB and YUV colour models. The mathematical relationship between RGB and HSI colour models are shown in equations (3), (4), (5).

$$I = \frac{R + G + B}{3} \tag{3}$$

$$S = 1 - \frac{3}{R + G + B}[\min(R, G, B)] \tag{4}$$

$$\theta = cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)^{\frac{1}{2}}]}\right\} \quad H = \begin{cases} \theta, & B \leq G \\ 360 - \theta, & B > G \end{cases} \tag{5}$$

YUV being a typical PPG colour model, defines the colour space in luminance (Y) and two chrominance (UV) components. Like the HSI model, it takes all RGB channels for each of its components. The mathematical relationship between RGB and YUV is shown in equation (6).

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{6}$$

The default colour model for the Android OS is the YUV colour format with NV21 encoding meaning that a conversion from YUV to RGB colour model may be necessary. In that case the 3 by 3 constants matrix provided in equation (6) can be inverted. The resulting matrix can be used for YUV to RGB conversion shown in equation (7).

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix} \tag{7}$$

3 Description of Project

This chapter details the full development of the mobile application, methods of non-causal and causal methods of signal processing and the procedures followed to achieve the project aim. See Appendices for full code.

3.1 Camera2 Implementation

The initial step to proceeding was to output a live video recording preview from the Camera2 API at a specified frequency using Java. Therefore, a simple application was developed to preview a live video for devices running Android Oreo 8 an above. As the newer devices have additional camera hardware such as multiple rear cameras with different lenses, a string array was produced including the camera identification. Then, the first available rear camera was chosen. In order to determine the number of pixels available in the images, a preview size of 1920x1080 pixels was chosen as most recent devices were determined to be well capable of capturing a video of this size at 30Hz. An additional thread was started for tasks that shouldn't block the main thread such as requesting additional hardware. Therefore, the flash methods were implemented in the background thread along with setting the camera characteristics such as the sampling rate. Following these procedures, the resulting preview was displayed on a texture view as it has call-back functions for when the surface texture is available, destroyed, changed its size or updated. The camera preview was started when the surface texture was available and closed when destroyed. Additional transformation features were added to scale the video preview up or down for and if the size was changed such as when the screen orientation changes from portrait to landscape. As the surface texture would be updated every time a new image was received, a loop counter variable was added to the texture update call-back to assess the frame rate and hence, the device performance. This was performed as a control measurement as well as a performance feedback of the implemented methods. Additionally, as the image processing methods would require additional processing power, performance measurements taken at this stage of the development process could be compared to future measurements. Camera frame rate was recorded from the launch of the application. On a Samsung Galaxy S7

running on Android 8.0, frame rate settled at the target 30fps within 2 seconds and no drops were experienced.

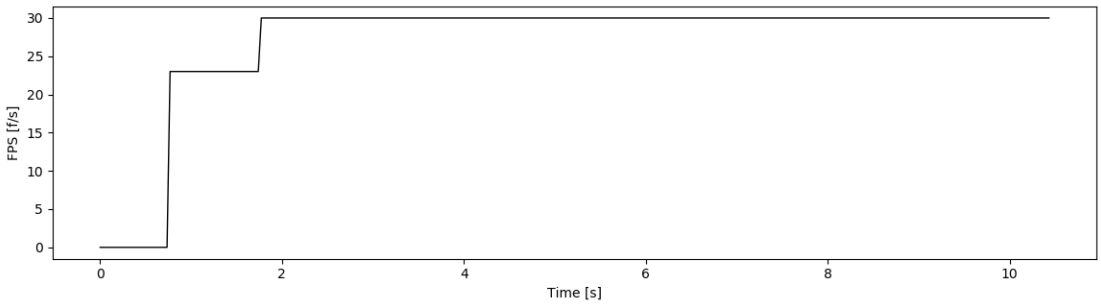
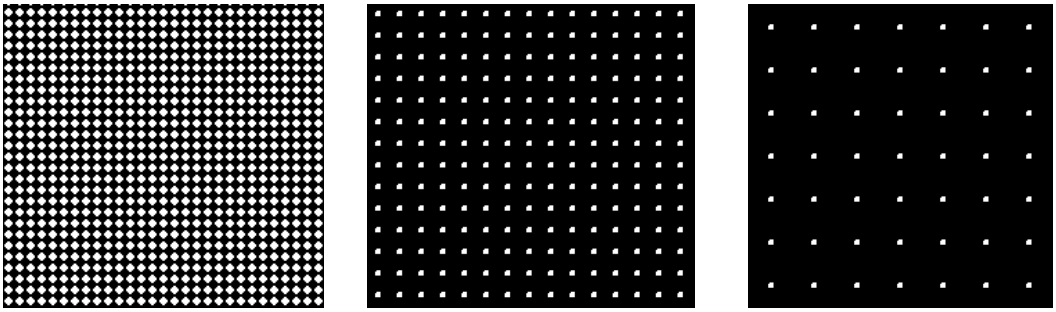


Figure 7, Control performance evaluation at 30Hz target frame rate

3.2 Live Video to RGB Conversion

The next step was to get the colour intensity of the images in the surface texture update call-back function. Therefore, the incoming image was converted to a bitmap to be able to call each image pixel by its row and column number. Thereafter, the standard Java colour class was used to obtain the red, green and blue intensities of the pixel.

As previously mentioned, surface update call-back would run every time a new image was received from the live video. Therefore, loop iteration speed is directly proportional to the camera frame rate. The total number of pixels available in the image was previously set to 1920x1080 which is 2,073,600 pixels. Following equation (1) for colour averaging, a set of for loops were generated to go through the image rows and columns, pixel by pixel. As the number of pixels were immensely large, the device performance was significantly dropped. To keep a constant 30 frames per second, image processing should be performed under 33ms. The development device used in this project, Samsung Galaxy S7, took 1612ms to run through 2,073,600 pixels meaning the resolution had to be dropped. This was done by skipping several pixels and sampling at equidistant points. Figure 8 shows the scaling principle used to improve the performance of the algorithm.



(a) 1:2 (b) 1:4 (c) 1:8

Figure 8, Scaling principle for reduced resolution

The number of pixels to iterate through were reduced until the execution time was below 30ms. The highest resolution that would execute the colour averaging around 30ms to 40ms was 120x120, 14400 pixels. The highest resolution that would execute the process consistently under 30ms was 60x60, 3600 pixels. Therefore, a frame rate consistency test was performed with the image processing algorithm looping through 100, 400, 3600 and 14400 pixels of the live video input. Figure 9 shows the findings.

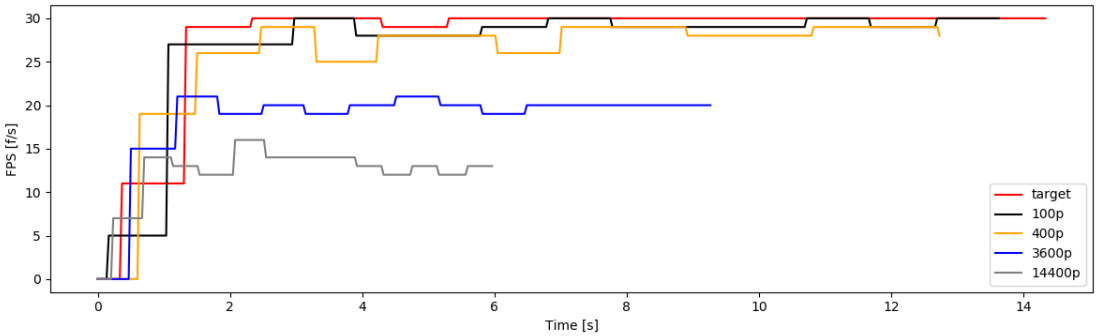


Figure 9, Performance comparison of different resolutions running the colour format algorithm

As the colour averaging wasn't the only process that the device had to perform, the performance was slightly below the expectations. However, averaging over 100 and 400 pixels, the device was able to keep the frame rate quite well with minimal drops. As expected, as the number of pixels increased, the frame rates kept dropping, limiting the maximum sampling rate for data acquisition which would be crucial when recording the data.

Following the performance evaluation, the colour detection ability of the algorithm was tested using the RGB colour format. The device camera was pointed towards

coloured A4 sheets and the registered RGB channel values were analysed. Evaluations were carried out under different scenarios to test the colour averaging process. Initially, the camera was pointed to a sheet containing the same colour throughout. This is shown in Figure 10.

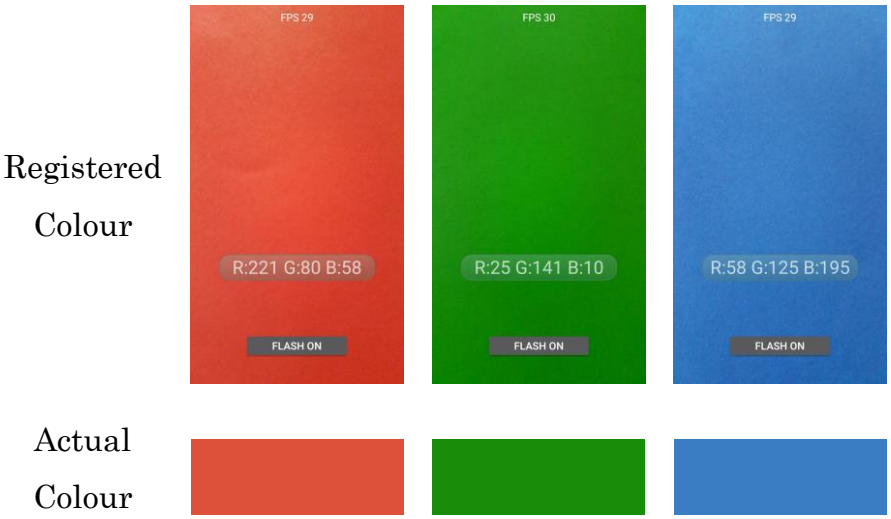


Figure 10, Colour averaging algorithm with solid colours

This was followed by pointing the camera towards sheets containing multiple colours. The aim of this test was to see if the algorithm would pick an average colour. This is shown in Figure 11.

Figure 11, Colour averaging algorithm with multiple colours

3.2.1 Recording PPG

Having implemented the colour averaging algorithm, the next step was to implement a recorder to record PPG data by placing the finger over the rear camera of the device. Therefore, a recorder function was added to the colour averaging application and a file writer was used to save the data into device memory as arrays in simple text format.

3.2.2 RGB and YUV Formats Comparison

As an initial step, RGB and YUV data were recorded with the device flash at 30Hz from the right index fingertip of a 24-year-old healthy male subject. As expected, and mentioned in the project introduction, most of the raw PPG signal was formed by its DC component where the actual signal was contained within its AC component. Recorded data was imported to Python and plotted against time in seconds which was calculated using equation (8).

$$T_{max} = \frac{N}{Fs} \tag{8}$$

Where, N is the total number of samples and Fs is the sampling rate. Figure 12 shows the initial recording of RGB and YUV data against time.

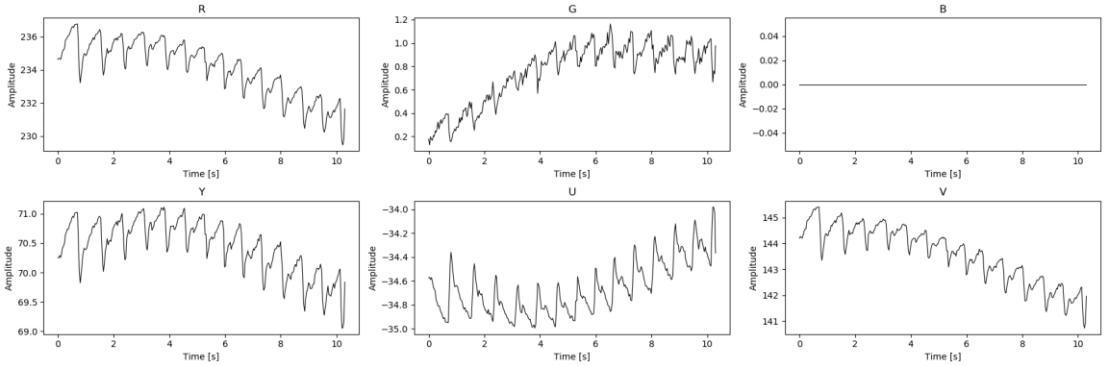


Figure 12, Raw PPG signal in RGB and YUV formats recorded from the rear camera of a Samsung Galaxy S7

To visualise the frequency spectrum of the signals, they were transformed from their time domain to frequency domain using the discrete Fourier Transform following equation (9) where N stands for the total number of samples, n stands for the sample index and k is the frequency index.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi kn}{N}} \quad (9)$$

The maximum signal frequency (x-axis) was determined by the Nyquist frequency, which is half the sampling frequency. This was done as the frequency spectrum included the complex conjugate values. As expected, the frequency spectrum had a very large 0Hz (DC) component. This is shown in Figure 13.

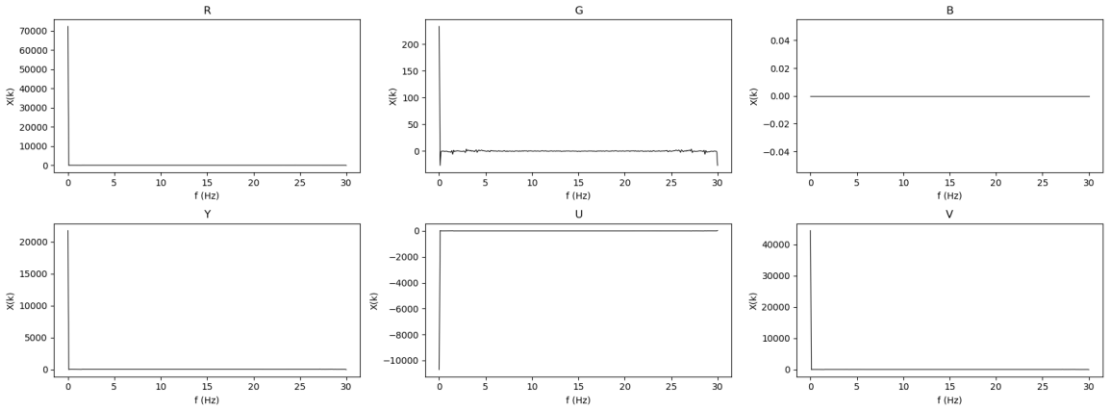


Figure 13, Frequency spectrum of raw RGB & YUV

DC noise was filtered out to analyse the frequency spectrum of the signals for additional noise and fundamental PPG frequencies. DC filtered frequency spectrum is shown in Figure 14.

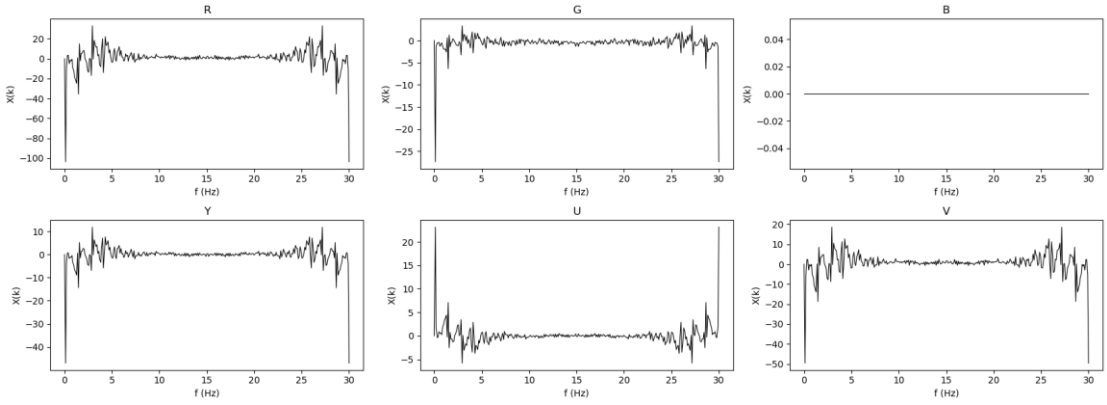


Figure 14, Frequency spectrum of DC removed RGB & YUV

As shown by previous studies [17], the fundamental frequencies of a PPG signal were analysed to be below 5Hz for both image formats. Following the frequency spectrum analysis, the signal was transformed back to its time domain using the inverse, discrete Fourier Transform following equation (10).

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k)e^{\frac{j2\pi kn}{N}} \quad (10)$$

The resulting, DC removed signal was plotted for further analysis. This is shown in Figure 15.

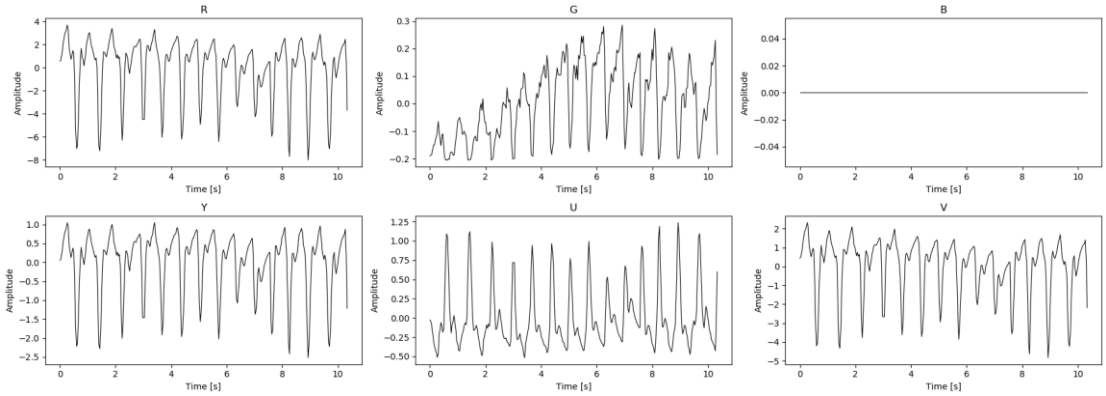


Figure 15, DC removed PPG signal in RGB and YUV formats recorded from the rear camera of a Samsung Galaxy S7

3.3 Causal Signal Processing

The output of a non-causal system depends on the past, present and future inputs of the signal. The discrete Fourier Transform used previously is a non-causal method of signal processing and requires a recorded signal with all inputs being available to use. Obviously, this cannot be performed in real time.

Therefore, implementing causal methods of signal processing was the next step to achieve an identical output shown in Figure 15. The two obvious choices are the FIR and IIR filters. As mentioned previously, FIR filters have a linear phase, meaning, time delay is identical at all frequencies, however, they are computationally expensive, introduce larger delays and require more memory. IIR filters require less computational power however, they offer a non-linear phase, meaning, time delay varies at different frequency outputs.

To decide on the type of digital filter to use, a quick performance comparison of the two filters were made. Having determined that the fundamental frequencies of the PPG signal in the 0.5Hz to 5Hz band, a band-pass filter was chosen for the comparison. Therefore, both the FIR and IIR filters were implemented in Java and

their coefficients were calculated analytically using Python. Both filters were tested for their performance.

3.3.1 FIR Filter

A FIR filter requires its coefficients to be analytically calculated from the ideal frequency response. Therefore, the ideal frequency response needs to be designed as an array with its length equal to the number of taps. This can be seen in the topology of a FIR filter, shown in Figure 16 where x is the input, T represents a delay, h are the coefficients, M is the number of taps and y is the output.

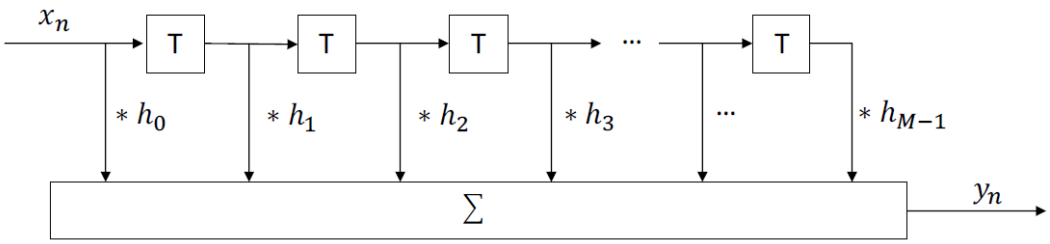
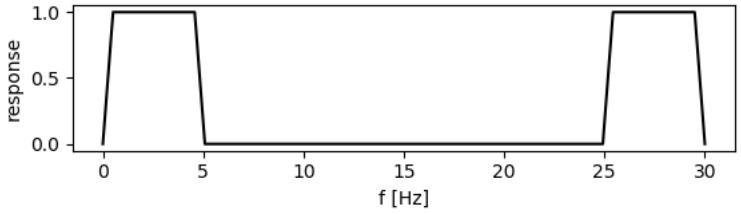


Figure 16, FIR filter topology

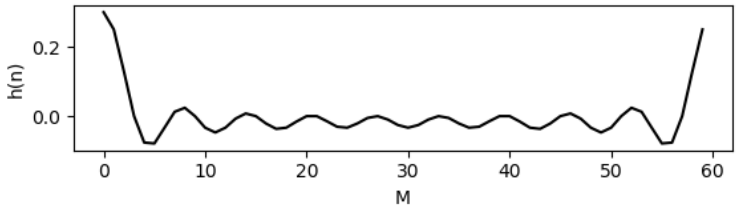
As the total output delay of a FIR filter is equal to half the number of taps, it's best to choose the least possible number of taps. As the PPG signal has its fundamental frequencies between 0.5Hz to 5Hz, the frequency resolution of the filter needs to be at least 0.5Hz. With a sampling rate of 30Hz, the number of taps needs to be at least 60. Therefore, a FIR filter would introduce a 30-sample delay, equivalent to 1 second in this application. To test the FIR filter, the coefficients had to be derived. The ideal frequency response was defined using amplitudes of 0 and 1 for stop and pass respectively. 0.5Hz to 5Hz band was set to pass and the complex conjugates of the real part were treated equally following equation (11) where N is the total number of samples.

$$x^*(k) = x(-k) = x(N-k) \tag{11}$$

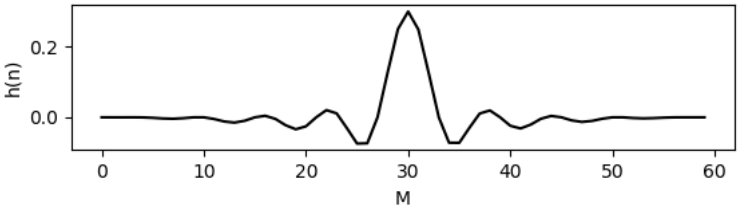
The ideal frequency response of the filter is shown in Figure 17 (a). To obtain the FIR coefficients, the ideal frequency response was transformed to time domain and shifted to positive time. These are shown in Figure 17 (b) and (c).



(a) Ideal frequency response



(b) FIR impulse response (time domain)



(c) FIR impulse response shifted to positive time

Figure 17, FIR coefficients derivation

To achieve an impulse response with minimal ripples whilst keeping a narrow transition width and good noise rejection, a Hanning window function was applied by simply multiplying the coefficients with the window function following equation (12).

$$h(n)_{new} = h(n).w(n) \qquad (12)$$

The coefficients were fed into the previously implemented Java FIR filter and the outputs were recorded.

3.3.2 IIR Filter

The coefficients of an IIR filter are based on the numerator (zeros) and the denominator (poles) of its transfer function, H(z). The transfer function zeros and poles are determined from the ideal frequency response of the filter. Equation (13) shows the canonical form of a second order IIR filter where b_0, b_1 and b_2 are the FIR coefficients, a_1 and a_2 are the IIR coefficients and the z values are the delays.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \tag{13}$$

Higher order IIR filters can be constructed by cascading 2nd order IIR filters to achieve improved filtering performance. The topology of a 2nd order Direct Form II filter would look like Figure 18.

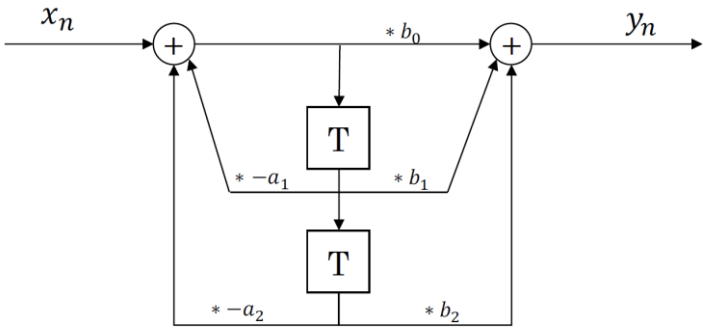
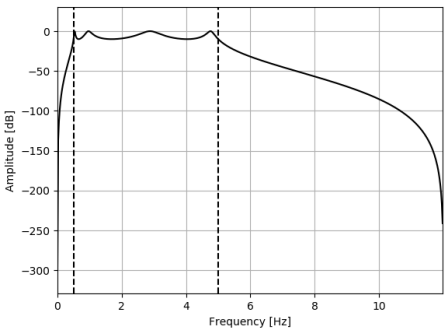
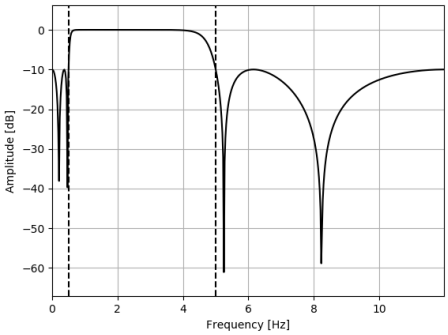


Figure 18, Topology of a 2nd order Direct form II filter

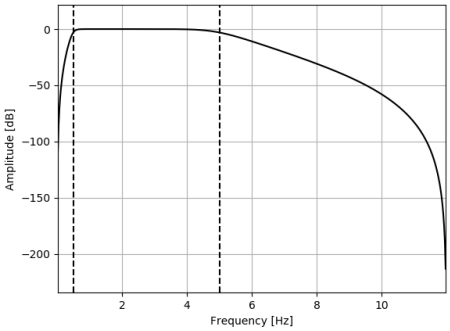
Therefore, to test the abilities of an IIR filter, a Java IIR filter was implemented that allowed cascading multiple IIR filters to achieve any given order. SOS coefficients for a 4th order band-pass Chebyshev I, Chebyshev II, Butterworth and Bessel filters were calculated using python scipy.signal library. The cut-off frequencies were set to 0.5Hz and 5Hz. Having calculated the filter coefficients, the transfer functions of all filters were derived following equation (13) for further analysis. To decide on the best filter to use, frequency response and group delays of all previously mentioned filters were analysed. Figure 19 shows the filter frequency responses where Figure 20 shows the corresponding group delay.



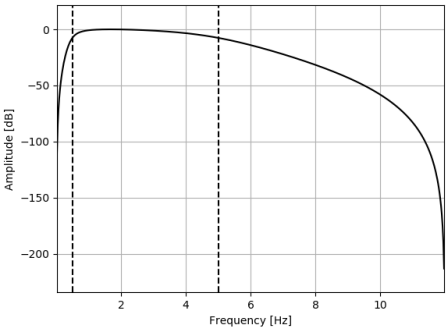
(a) Chebyshev I



(b) Chebyshev II

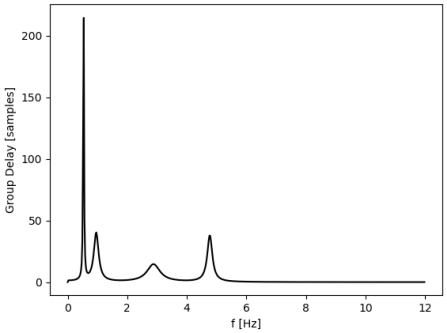


(c) Butterworth

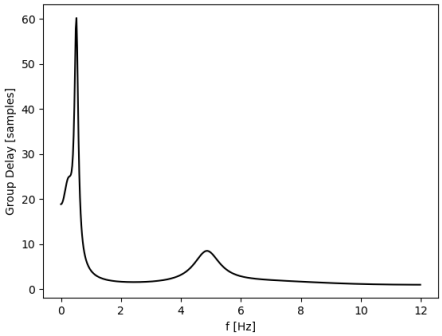


(d) Bessel

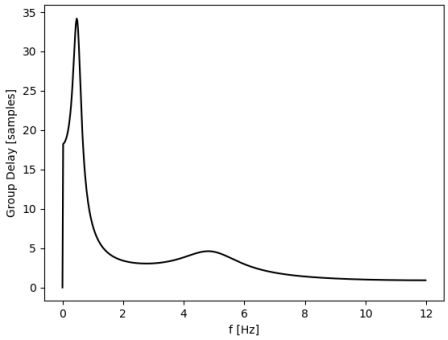
Figure 19, Frequency responses of chosen IIR filters



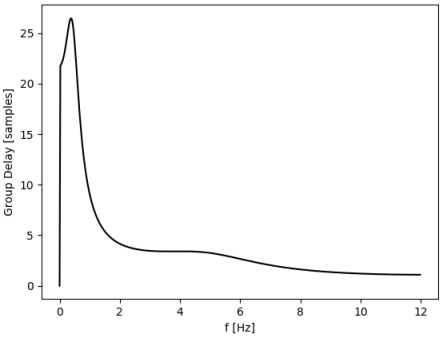
(a) Chebyshev I



(b) Chebyshev II



(c) Butterworth

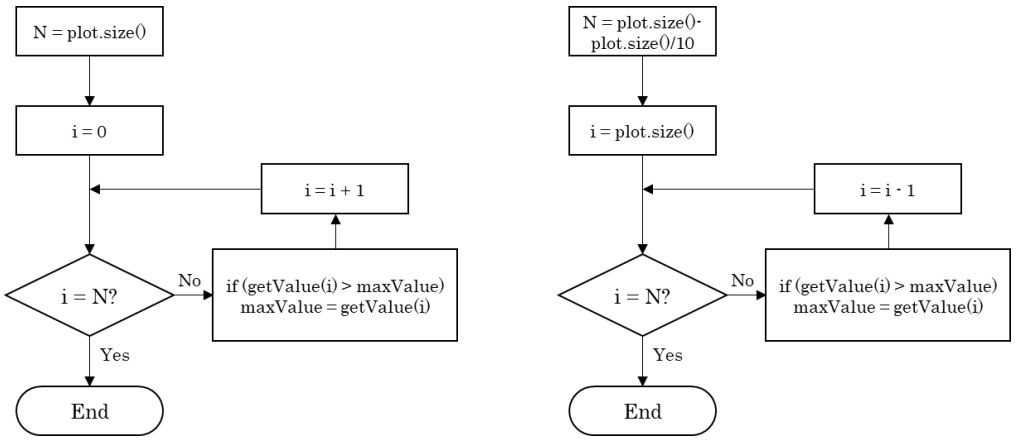


(d) Bessel

Figure 20, Group delays of chosen IIR filters

3.4 Live Plotting & Threshold

So far, debugging and analysis was performed remotely by applying the implemented signal processing algorithm to the detected signal, then recording the output and viewing the results remotely on Python. At this stage, the filtered signal could be used to detect the momentary heart rate by applying a threshold. However, even if the signal was DC removed, the amplitudes varied, meaning a fixed threshold wouldn't perform very well. Therefore, a dynamic threshold algorithm needed to be applied. To efficiently debug the threshold algorithm, both the signal and threshold needed to be viewed in real time. Android Plot, SimpleXYSeries library was used to plot the PPG signal and the threshold in real time. Plot boundaries were set to be dynamic to adjust to the highest and lowest points of the visible plot. To find the overall maximum value in the visible plot, a loop was executed through all visible points of the plot each time it was updated. However, updating the threshold with an overall maximum would result in a slow threshold adjustment. Therefore, the threshold was updated using the local maximums. To find the local maximums, the same loop was executed backwards through a single PPG pulse. This was done to adjust the threshold to the most recent signal. Overall maximum and local maximum algorithms are shown in Figure 21 (a) & (b) respectively.



(a) Overall Maximum

(b) Local Maximum

Figure 21, Overall & Local maximum algorithms

3.5 Heartrate Detection

Having implemented the thresholding, the next step was to get the timestamp every-time a rising edge was detected. Additionally, bogus detections needed to be removed. To do so, a bogus timestamp and an overall timestamp variable was set. Both timestamps were initially set to 0 for the initial execution of the algorithm. Additionally, a Boolean value was set for above threshold and another for initial loop. If a signal was detected to be above the threshold with the difference of detection timestamp and bogus timestamp being greater than 330ms (180 bpm), above threshold was set to false. Thereafter, an initial loop was executed where the timestamp of that detection was saved. When a signal above the threshold was detected for the second time, a new timestamp was recorded and the difference between the previous and current timestamp was used to calculate the momentary heartrate. Then, a second check was performed to see if the heart rate was below 180 as a higher bpm value would most likely indicate a bogus detection. The logic behind the heartrate detection algorithm is shown in Figure 22.

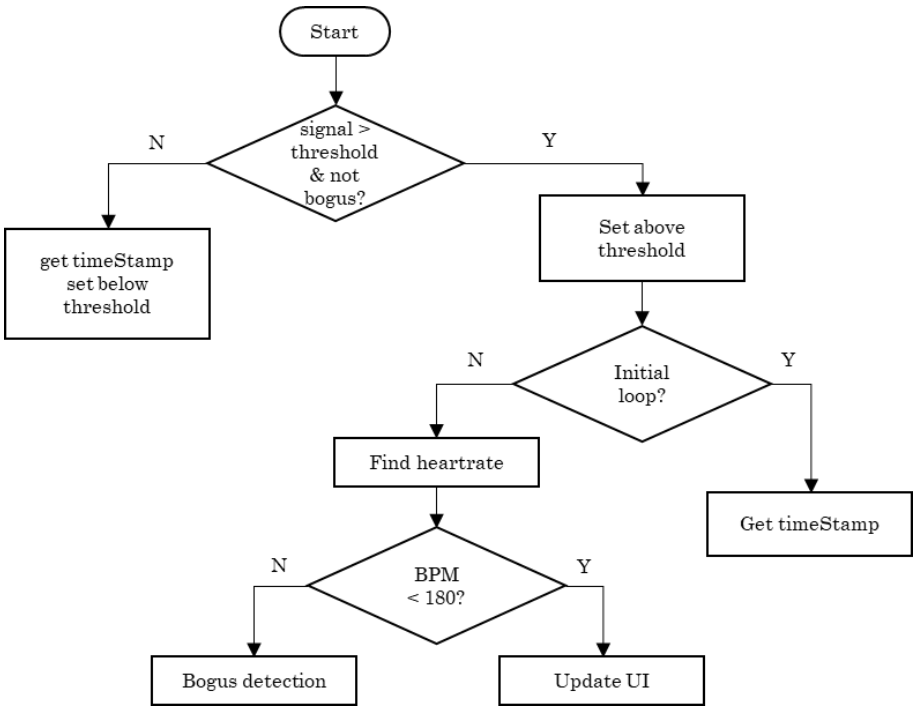
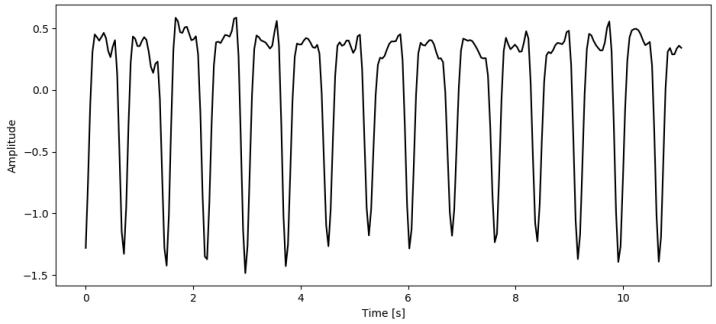


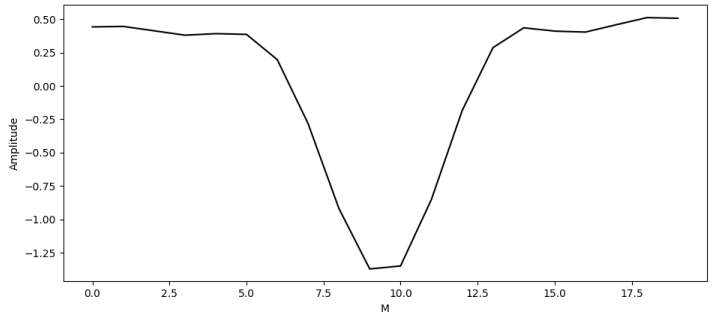
Figure 22, Heartrate detection

3.6 Matched Filter

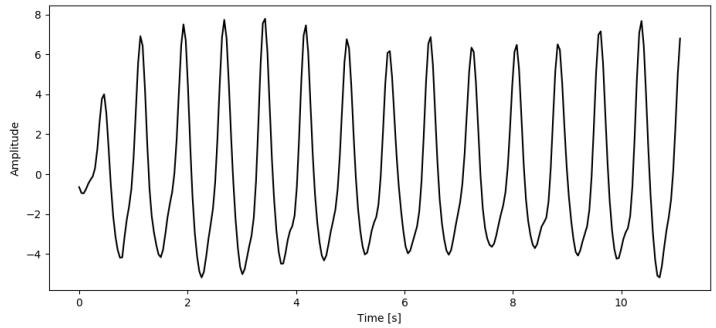
Although, the previously implemented heartrate detection algorithm could successfully perform the detections, a matched filter was implemented to further improve the SNR. This followed the same steps as the FIR filter implementation did. However, this time the coefficients were calculated from a characteristic signal using Python. Figure 23 shows the type of template used for the matched filter design.



(a) Band-pass filter output



(b) Matched filter coefficients



(c) Matched filter output

Figure 23, Matched Filter Design

Thereafter, previously implemented FIR filter was used with the matched filter coefficients to boost the SNR.

3.7 Non-Contact PPG

Having performed the heart rate detection at the fingertip, the next step was to investigate the possibility of increasing the camera distance from the skin. As mentioned before, both transmittance and reflectance methods require a light source and a detector. So far, measurements were performed using the device flashlight as the light source. However, for measurements taken away from the skin, the bright white LED flashlight was expected to overexpose the image. Therefore, the same algorithm applied for the fingertip recordings was used with minor changes to record PPG data away from the skin. Data was recorded from the subject's hand and face with the following configurations.

- ~5cm away from hand (palm) with rear camera
- ~10cm away from hand (palm) with rear camera
- ~15cm away from face with front camera
- ~30cm away from face with front camera

For all measurements, the same band-pass filters were used to remove the noise, however, a custom matched filter was implemented for each signal using its characteristics to improve the SNR as best as possible. The subject was asked to hold the device as the application was intended for self-measurement. For all PPG measurements, AttysECG hardware was used in parallel to record ECG measurements. Python was used to plot both signals together. Both measurements were analysed for any correlations between them. ECG R-peaks were used as a reference to detect the heart rate. All recordings were performed in a closed area under natural light with similar luminance. Motion artefacts were kept to a minimum by asking the subject to hold the device as still as possible.

Following the recordings, the ECG R-peaks were added to the PPG signal to boost the regions where a heartbeat was expected. An attempt was made to detect the heart rate for each proximity and the results were compared to the fingertip detections.

4 Results and Discussion

This section presents the outputs of each implemented causal filter and discusses their effectiveness. The decision on the type of filter used is introduced and justified. PPG average and momentary heart rate calculations are compared to Attys ECG and their accuracy and repeatability are shown. Additionally, the robustness of the heart rate detection algorithm is tested, non-contact PPG results are introduced and using them for heartrate detection is discussed.

4.1 Best Colour Format for real-time PPG

Apart from the blue channel of RGB format, all colour formats were found to convey information regarding PPG. However, there were crucial differences between their performances depending on the ambient light. As investigated by previous studies, YUV data was found to be superior over RGB [19]. Under dark conditions, <100lux, the development device (Samsung Galaxy S7) could not register any green or blue data, however, the PPG data was easily extracted from the red channel. In daylight, >1000lux, PPG data was shared between the green and red channels. Therefore, using the red channel alone didn't provide repeatable detections. On the other hand, Y, U and V channels containing information related to all RGB channels provided repeatable peaks under most light conditions when coupled with a band-pass filter. The findings are shown in Figure 24 for low ambient light and Figure 25 for daylight.

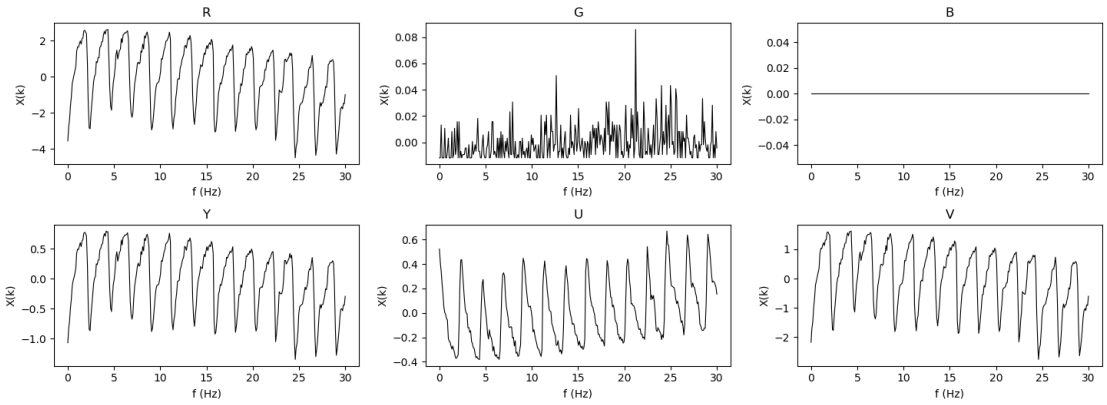


Figure 24, PPG recordings with low ambient lighting

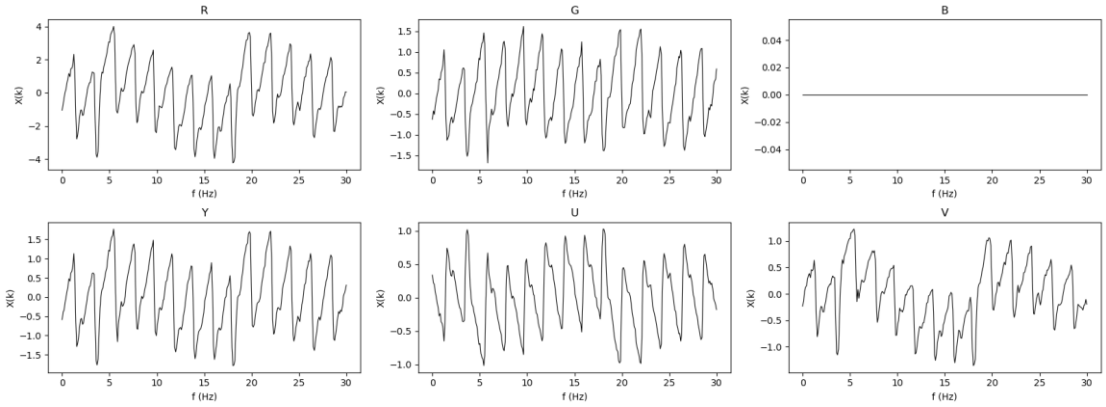


Figure 25, PPG recordings in daylight

As it can be seen, both low ambient light and daylight measurements contain no PPG signal in the blue channel where all YUV channels are populated with data. Therefore, YUV data was used for further implementations.

4.2 FIR Filter

As described, a FIR filter was implemented, and its output was tested by recording the filtered U data from the subject's finger. The filter worked as expected allowing the 0.5Hz – 5Hz band to pass while attenuating the rest of the frequencies as shown in Figure 26.

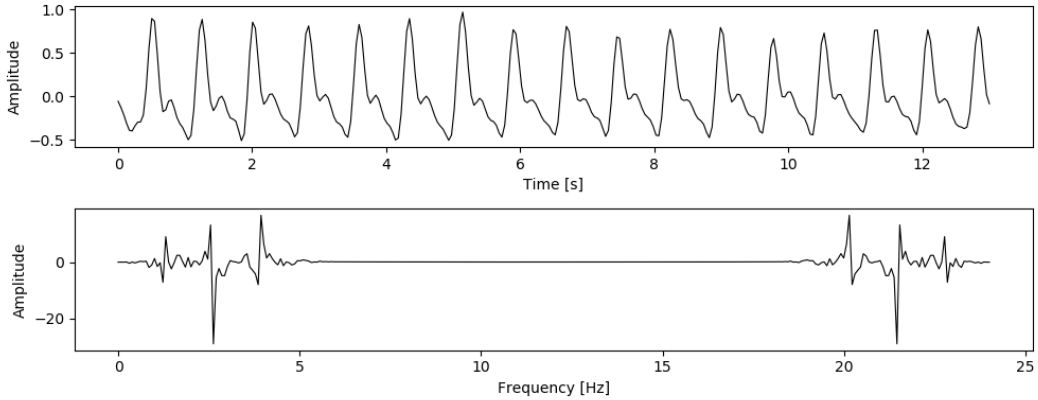


Figure 26, FIR filter output from Samsung Galaxy S7

However, in a mobile application, battery life consumption is one of the main factors to consider. Therefore, the most power efficient signal processing methods that achieve the same results are usually desirable.

4.3 IIR Filter

The IIR filter was tested the same way as the FIR filter for its output. As expected, it performed identical to the FIR filter, removing frequencies below 0.5Hz and above 5Hz. However, the IIR filter performed the process with higher efficiency and lower delay at the cost of introducing a non-linear phase. As the output delay depended on the signal frequency, peaks were expected to have an identical delay. Therefore, having a non-linear phase did not introduce uncertainties into the heart-rate measurements. Filter output is shown in Figure 27.

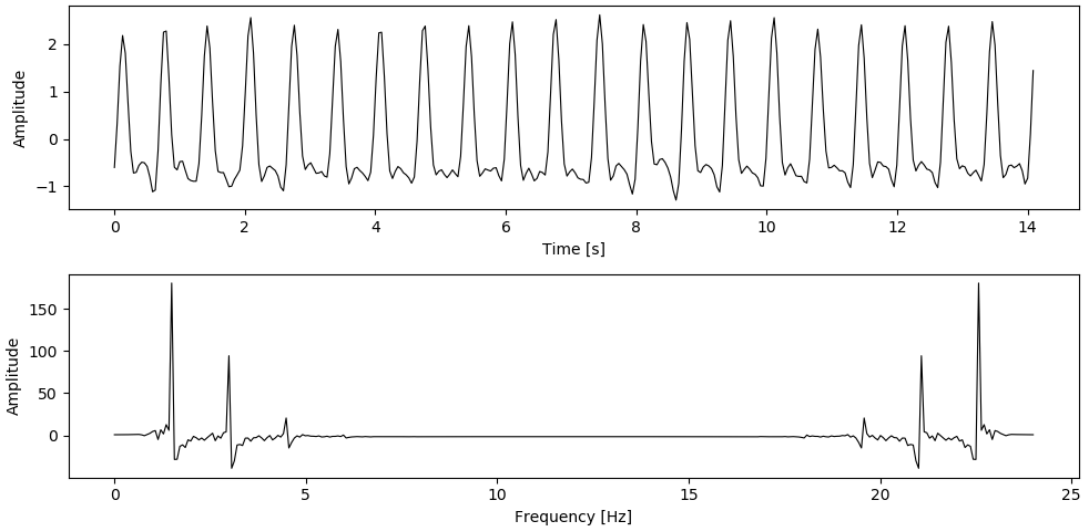


Figure 27, IIR filter output from Samsung Galaxy S7

As it can be seen, despite the wide transition width of the Butterworth filter, the band-pass filter performed the filtration as expected. Power consumption being the prior concern, the IIR filter was chosen for the application.

4.4 Heartrate Detection

Heart rate detection was performed by applying an adaptive threshold to the signal and recording the timestamp each time the threshold line crossed a rising edge of the PPG signal. As this type of detection only required the presence of two peaks, the momentary heart rate could be calculated within a second after the detection of the initial heart rate for heart rates above 60bpm. As mentioned in the work description, the threshold was programmed to adapt to the signal from beat to beat. This greatly improved the robustness of the detection algorithm as

constant or slow reacting threshold algorithms would fail to capture varying signal amplitudes. Detection robustness is shown in Figure 28.

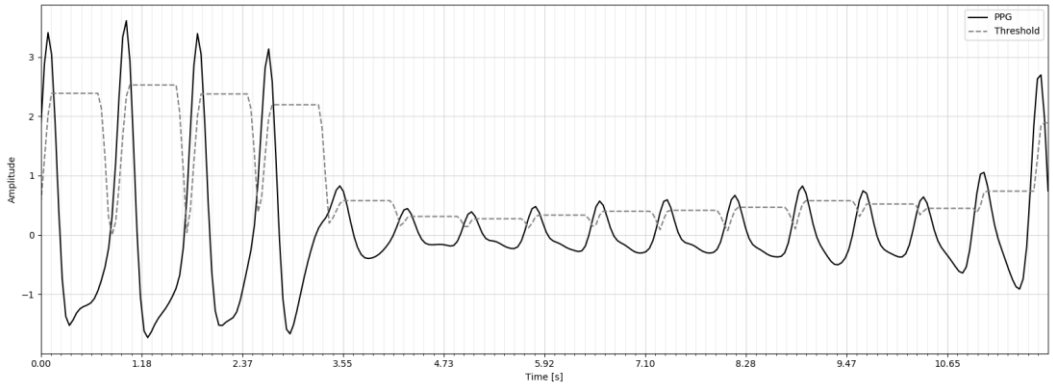


Figure 28, Heartrate detection robustness test

This type of signal could be reproduced if a user pressed harder or softer onto the camera. A user that presses harder would see lower signal amplitudes as the increased pressure at the fingertip would reduce the blood flow. However, as seen in Figure 28, the adaptive threshold algorithm managed to capture all the peaks. To improve the accuracy of the detections, the magnitudes of the incoming detections were added and divided by the total number of detections after the user placed their finger onto the camera.

Having determined that the detection algorithm was robust and quick, its accuracy was compared to Einthoven II, ECG, data that was recorded simultaneously.

The momentary heart rate detection with the PPG data recorded by the Android device was found to have an accuracy within ±5% compared to the ECG data. However, the average heart rate detection was found to be within ±1% of the ECG signal for measurements between 10s to 15s and within ±0.5% for measurements above 15s. Although the comparison varied for each signal, Figure 29 shows a comparison of the heart rate detection performed by AttysECG (ECG) and the Samsung Galaxy S7 (PPG) used in the project development.

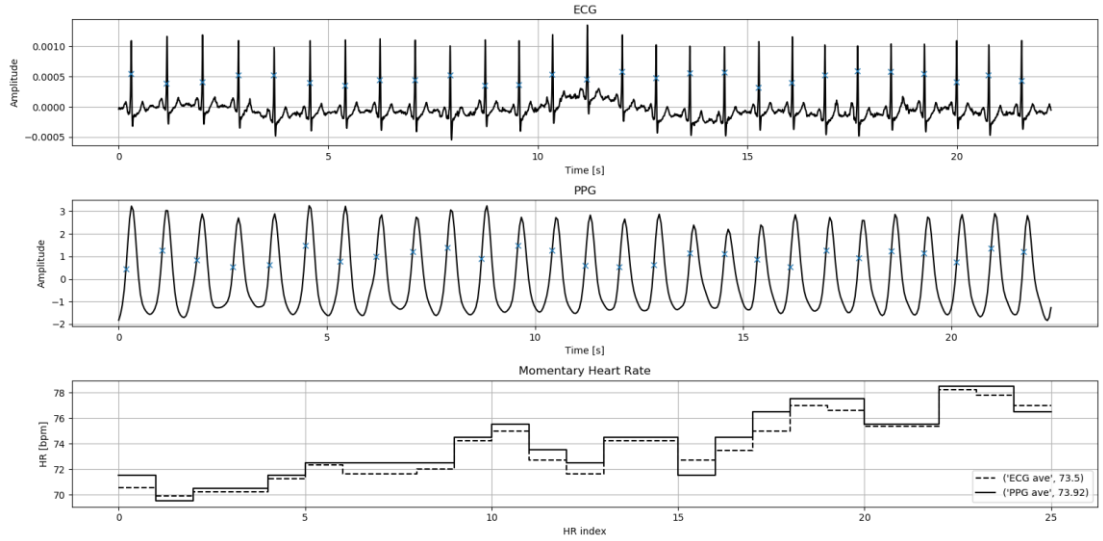


Figure 29, ECG & PPG heart rate detection comparison

As it can be seen, an average heart rate of 73.5bpm was detected from the ECG signal and 73.9bpm was detected from the PPG signal. Additionally, the maximum deviation between the two momentary heart rate detections was 1bpm. The final setup used to achieve these results are shown as a block diagram in Figure 30.

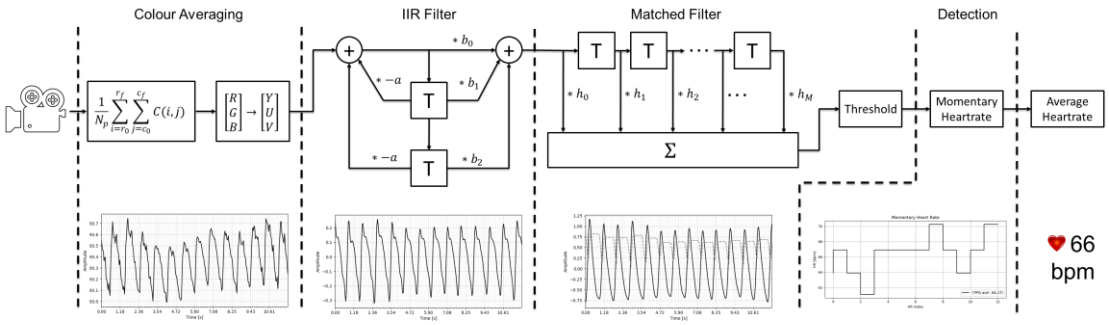


Figure 30, Block diagram [Camera to heartrate detection]

4.5 Non-Contact PPG

It was possible to detect the heart rate of the subject using the development device for all non-contact scenarios mentioned in the work description, however, as expected the SNR reduced as the distance was increased. However, as previously mentioned, fingers, palms and face provide good perfusion values [10], desirable for PPG measurement.

Measurements taken at the fingertips provided data with minimal ambient light and motion artefacts as the camera was fully covered with the subject's finger. For

measurements taken from the face, the front camera was used and although the subject was asked to hold as still as possible, changes in the ambient light and motion artefacts presented themselves within the 0.5Hz to 5Hz band. As expected, measurements taken closer to the skin provided a better SNR. Figure 31 shows a measurement taken from the subject's face from 30cm.

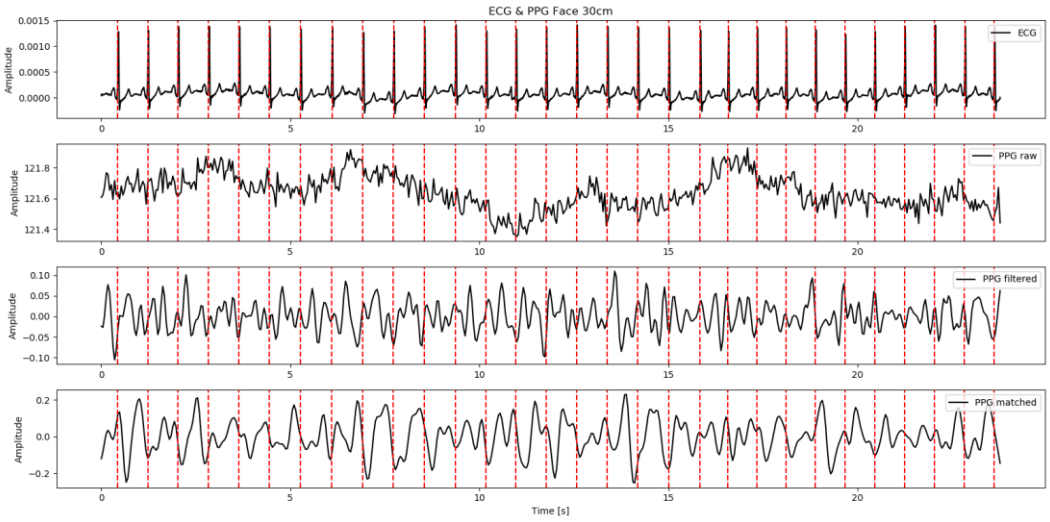


Figure 31, PPG 30cm from face

The raw signal provided no understandable data, however, passing the signal through a 4th order Butterworth band-pass revealed the PPG signal. The filtered signal was further improved by feeding it into a matched filter with coefficients taken from the time reversed template of an ideal PPG signal. As it can be seen, this improved the SNR further. Using the ECG signal as a template for the expected heartbeat locations, heartrate data was successfully extracted from the recorded PPG signal after applying a similar bogus removal algorithm used for the fingertip measurements.

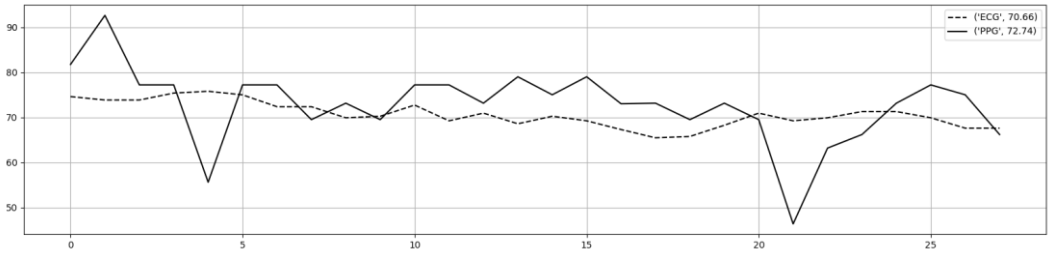


Figure 32, ECG & PPG heartrate comparison, face 30cm

Even after applying a post bogus removal algorithm using Python, the momentary heartrate detections varied by a large margin, ±15% for some detections. However, for the 25 second recording, the average heartrate detection for PPG was within ±3% of the ECG detections. The results improved for recordings taken closer to the subject's skin. Figure 33 shows the measurement taken from the subject's face at a distance of 15cm.

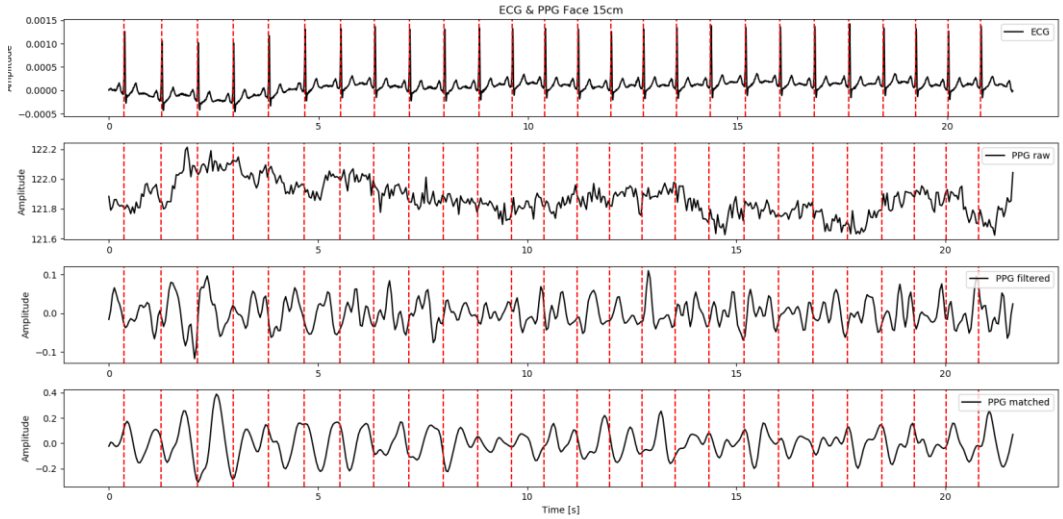


Figure 33, PPG 15cm from face

As it can be seen, although the raw data was similarly noisy to the 30cm recording, the matched filter output was much better with fewer bogus detections. Switching back to the rear camera for the palm measurements, under ideal conditions, even the raw signal provided visible PPG data. Passing the signal through the band-pass and matched filters, a momentary heartrate detection accuracy within ±10% and an average heartrate detection accuracy within ±2% was achieved for measurements over 20 seconds. Measurements taken 10cm from the palm are shown in Figure 34.

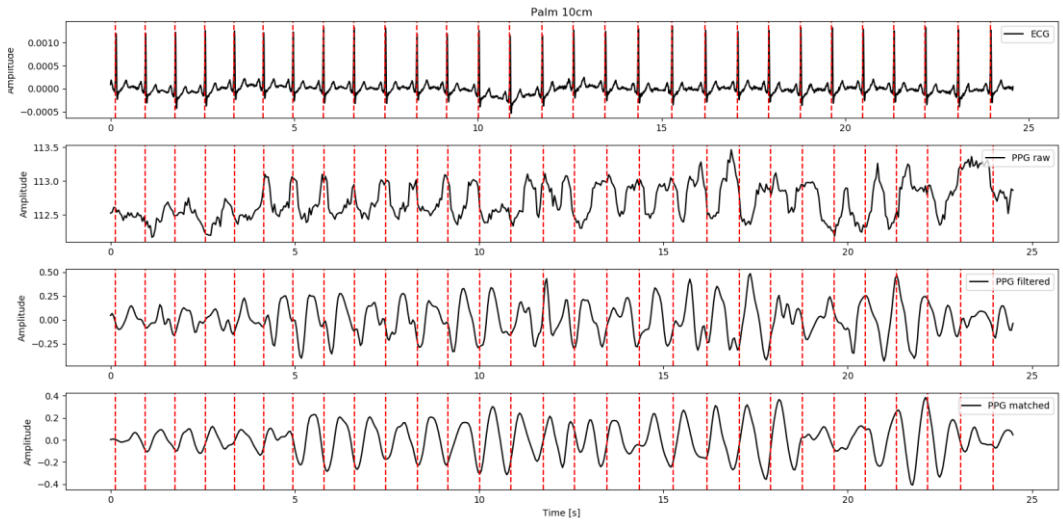


Figure 34, PPG 10cm from palm

Moving the camera closer to the skin, the SNR greatly improved, and the detections were almost identical to the fingertip measurements with reduced amplitudes. Measurements taken within 5cm distance to the palm are shown in Figure 35.

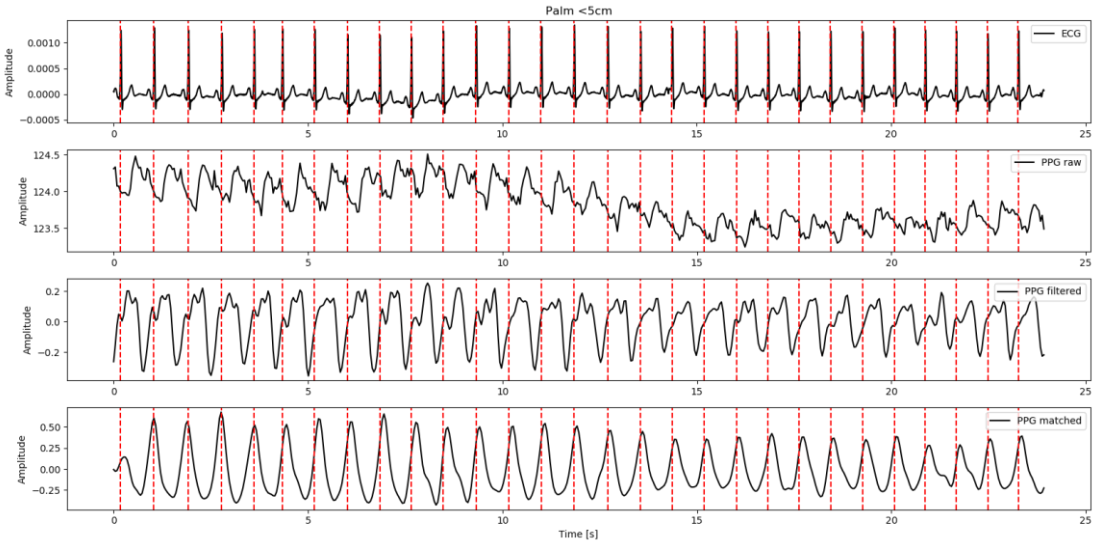


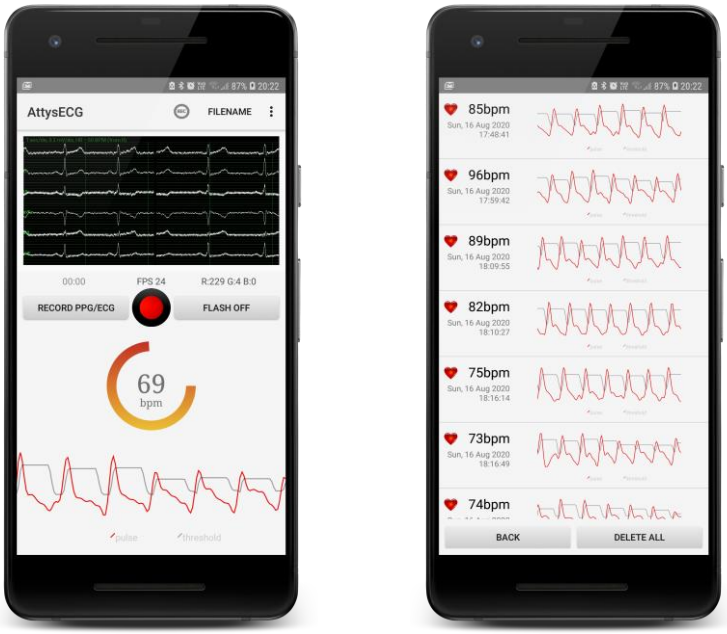
Figure 35, PPG <5cm from palm

As expected, with the camera being closer to the skin, the effect of inconsistent ambient light and motion artefacts were reduced leaving the 0.5Hz to 5Hz band clean. Therefore, measurements taken at this distance were within ±5% for momentary heart rate and ±1% for average heart rate for periods over 20 seconds.

Although non-contact PPG measurements provided heartrate data, the practicality of the method was reduced by its sensitivity to inconsistent ambient light and motion artefacts. The fingertip measurements provided the user with quick and accurate data regardless of the uncontrollable factors that affected the non-contact measurements.

4.6 Attys ECG

Having obtained easy to use, high accuracy results at the fingertip, the application was tailored to fit the needs of an average user. A measurement interval of 15 seconds was set as it was found to provide an average heartrate accuracy within ±0.5% of ECG. A camera preview was provided for guiding the user to place their finger correctly. A manual flash button was added for users to experiment with the application if necessary, and the recorder used to achieve the project aims and objective were provided to the users for future implementations or research.
Additionally, further options were provided to the users such as an artificial beeper, history of recordings, adjustable frame rates and a frame rate analyser for monitoring the device performance. The extended application is shown in Figure 36.



(a) Main PPG activity (b) Recordings history

Figure 36, Extended application, AttysECG

5 Conclusions

In this project, an existing mobile application was extended for devices running on Android 8.0 or higher to add features for performing PPG heart rate detection using the mobile camera. The images were sampled at 24Hz and an average RGB intensity of each image was found by applying an efficient averaging algorithm. Sampled RGB data was converted to YUV format as it was found to convey more information regarding PPG [19]. A 4^{th} order Butterworth band-pass filter with cut-off frequencies at 0.5Hz and 5Hz was implemented to remove the noise. The filtered signal was fed to a matched filter for improving the SNR and an adaptive, robust thresholding was applied to detect the momentary heart rate.

PPG data of a 24-year-old healthy male subject was recorded at the fingertip, close to hand, close to face and far from face. With each PPG recording AttysECG hardware was used to record ECG data for comparison and calculation of the correlations. All non-causal signal processing was done using Python where causal processing methods were implemented in Java. Measurements taken from the fingertip were found to yield average heartrate detections within ±0.5% of the equivalent ECG measurement for periods over 15 seconds. Non-contact measurements taken away from the skin were found to be very sensitive to ambient light and motion artefacts, however, with a matched filter with variable coefficients for different scenarios, it was possible to measure the heartrate of the user. For measurements taken 30cm away from the face for a duration of 25 seconds, a momentary heartrate accuracy within ±15% and an average heartrate accuracy within ±3% of ECG was achieved. As expected, the SNR improved by moving the camera closer to the skin. Measurements taken within 5cm to the palm provided a signal with no bogus detections and an average heartrate accuracy within ±1% of ECG for the 25 second recording.

However, the practicality of non-contact PPG was reduced by its sensitivity to inconsistent ambient light and motion artefacts. Fingertip measurements provided the user with quick and accurate data that was insensitive to the uncontrollable factors that affected the non-contact measurements. Therefore, the practical and simple use of photoplethysmography using the camera of a mobile device was found to be limited to mobile camera being in close proximity.

5.1 Further Development

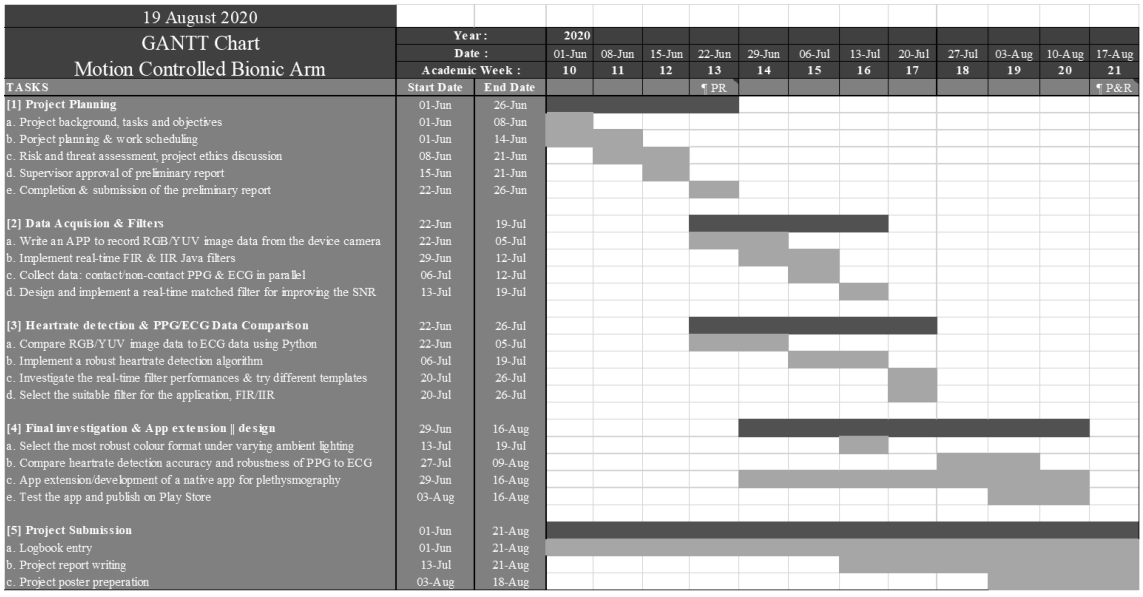
For the time being, the mobile application was kept limited to PPG measurements at the fingertip as it provided the user with a simple and accurate method of heartrate measurement. Non-contact measurements demanded the user to hold as still as possible. The algorithm could be modified to provide improved robustness to motion artefacts by spatial subspace rotation. This method was introduced as a concept by previous studies [20]. The proposed algorithm did not require the detection of skin-tone changes, instead, it estimated a spatial subspace of only the skin pixels and measured its temporal rotation for extracting the pulse. A recent study suggested a method of further improving the robustness of non-contact PPG by removing the non-skin pixels from the image [21]. The pixel colour averaging algorithm used in this project included all available pixels in the process. This worked well when all the available pixels were related to the skin, however, it reduced the SNR for distant measurements as the subject's eyes, eyebrows, hair and the ambient background were all included in the colour averaging process. The SNR can be improved by filtering non-skin related pixels out before performing the colour averaging process.

In this project, both research and development were done with the Samsung Galaxy S7 as it was the only available Android device. Therefore, the mobile application was not tested on devices with multiple rear cameras. Although multiple-camera-support was introduced for the Camera2 API, adapting the code to work with the newer, CameraX API would future-proof the application. Additionally, due to the COVID-19 pandemic, the PPG data was collected from a single subject. Measurements taken from a larger number of subjects would allow for improving the matched filter performance.

6 Project Management Review

As the project aim was software development, no social contact or access to facilities were required. Therefore, the workflow was smooth, and tasks were executed as expected. However, the order of workflow was slightly altered, and several additional tasks were added to achieve the objectives. Data acquisition for distant measurements were performed after the fingertip heart rate algorithm was implemented. This was done as the distant measurement were expected to be noisy and to require real-time filtering. Fingertip measurements were easier to understand and develop with. Therefore, the filters were implemented using fingertip recordings and applied to distant recordings as the fundamental frequencies were expected to be the same.

The project plan suggested the design and implementation of an IIR filter only. However, a FIR filter was developed in addition to study the effects of having a linear phase for the output. Additionally, the preliminary GANTT chart did not include image data conversions as RGB data was expected to be sufficient. The updated GANTT chart is provided in Figure 37.



19 August 2020														
GANTT Chart	Year :		2020											
Motion Controlled Bionic Arm	Date :		01-Jun	08-Jun	15-Jun	22-Jun	29-Jun	06-Jul	13-Jul	20-Jul	27-Jul	03-Aug	10-Aug	17-Aug
	Academic Week :		10	11	12	13	14	15	16	17	18	19	20	21
TASKS	Start Date	End Date				¶ PR								¶ P&R
[1] Project Planning	01-Jun	26-Jun												
a. Project background, tasks and objectives	01-Jun	08-Jun												
b. Porject planning & work scheduling	01-Jun	14-Jun												
c. Risk and threat assessment, project ethics discussion	08-Jun	21-Jun												
d. Supervisor approval of preliminary report	15-Jun	21-Jun												
e. Completion & submission of the preliminary report	22-Jun	26-Jun												
[2] Data Acquision & Filters	22-Jun	19-Jul												
a. Write an APP to record RGB/YUV image data from the device camera	22-Jun	05-Jul												
b. Implement real-time FIR & IIR Java filters	29-Jun	12-Jul												
c. Collect data: contact/non-contact PPG & ECG in parallel	06-Jul	12-Jul												
d. Design and implement a real-time matched filter for improving the SNR	13-Jul	19-Jul												
[3] Heartrate detection & PPG/ECG Data Comparison	22-Jun	26-Jul												
a. Compare RGB/YUV image data to ECG data using Python	22-Jun	05-Jul												
b. Implement a robust heartrate detection algorithm	06-Jul	19-Jul												
c. Investigate the real-time filter performances & try different templates	20-Jul	26-Jul												
d. Select the suitable filter for the application, FIR/IIR	20-Jul	26-Jul												
[4] Final investigation & App extension || design	29-Jun	16-Aug												
a. Select the most robust colour format under varying ambient lighting	13-Jul	19-Jul												
b. Compare heartrate detection accuracy and robustness of PPG to ECG	27-Jul	09-Aug												
c. App extension/development of a native app for plethysmography	29-Jun	16-Aug												
e. Test the app and publish on Play Store	03-Aug	16-Aug												
[5] Project Submission	01-Jun	21-Aug												
a. Logbook entry	01-Jun	21-Aug												
b. Project report writing	13-Jul	21-Aug												
c. Project poster preperation	03-Aug	18-Aug												

Figure 37, GANTT chart produced after work completion

The extension code developed in this project is provided in the Appendix, see https://github.com/MustafaBiyikli/AttysECG.git for complete code.

7 References

[1] S. Kwon, H. Kim, and K. S. Park, "Validation of heart rate extraction using video imaging on a built-in camera system of a smartphone," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 2012, doi: 10.1109/EMBC.2012.6346392.

[2] A. P. Zaretskiy, K. S. Mityagin, V. S. Tarasov, D. N. Moroz, and A. S. Kuraleva, "Robust heart rate estimation using combined ECG and PPG signal processing ," *IOP conference series. Materials Science and Engineering*, vol. 537. p. 42077, 2019, doi: 10.1088/1757-899X/537/4/042077.

[3] A. B. Hertzman, "THE BLOOD SUPPLY OF VARIOUS SKIN AREAS AS ESTIMATED BY THE PHOTOELECTRIC PLETHYSMOGRAPH," *Am. J. Physiol. Content*, 1938, doi: 10.1152/ajplegacy.1938.124.2.328.

[4] A. B. HERTZMAN, "Observations on the finger volume pulse recorded photoelectrically," *Am. J. Physiol.*, vol. 119, pp. 334–335, 1937.

[5] A. A. Kamshilin and N. B. Margaryants, "Origin of Photoplethysmographic Waveform at Green Light," in *Physics Procedia*, 2017, doi: 10.1016/j.phpro.2017.01.024.

[6] A. A. Alian and K. H. Shelley, "Photoplethysmography," *Best Practice and Research: Clinical Anaesthesiology*. 2014, doi: 10.1016/j.bpa.2014.08.006.

[7] K. Matsumura, P. Rolfe, and T. Yamakoshi, "iPhysioMeter: A smartphone photoplethysmograph for measuring various physiological indices," *Methods Mol. Biol.*, 2015, doi: 10.1007/978-1-4939-2172-0_21.

[8] Statista, "Number of smartphone users worldwide from 2016 to 2021(in billions)," 2020. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. [Accessed: 04-Aug-2020].

[9] A. C. Gacek and W. Pedrycz, "ECG signal processing, classification and interpretation: a comprehensive framework of computational intelligence ."

Springer , London;New York; , 2012.

[10] T. Tamura, Y. Maeda, M. Sekine, and M. Yoshida, "Wearable photoplethysmographic sensors—past and present," *Electronics* . 2014, doi: 10.3390/electronics3020282.

[11] Q. J. W. Milner and G. R. Mathews, "An assessment of the accuracy of pulse oximeters," *Anaesthesia*, 2012, doi: 10.1111/j.1365-2044.2011.07021.x.

[12] E. Thomson, "Validity of Heart Rate Measurements for the Apple Watch and Fitbit Charge HR 2: 2736 Board #19 June 1 2: 00 PM - 3: 30 PM ," *Medicine & Science in Sports & Exercise* , vol. 50, no. 5S Suppl 1. American College of Sports Medicine , p. 666, 2018, doi: 10.1249/01.mss.0000538196.40788.41.

[13] L. A. M. Aarts *et al.*, "Non-contact heart rate monitoring utilizing camera photoplethysmography in the neonatal intensive care unit - A pilot study," *Early Hum. Dev.*, 2013, doi: 10.1016/j.earlhumdev.2013.09.016.

[14] J. Young, "Comparison of the Efficiency of Java and C++ Performance using Sorting Algorithms ," *Bulletin of the South Carolina Academy of Science* . South Carolina Academy of Science , p. 56, 2001.

[15] J. Rumiński, "Reliability of pulse measurements in videoplethysmography," *Metrol. Meas. Syst.*, 2016, doi: 10.1515/mms-2016-0040.

[16] F. Foroozan, M. Mohan, and J. S. Wu, "Robust Beat-To-Beat Detection Algorithm for Pulse Rate Variability Analysis from Wrist Photoplethysmography Signals," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018, doi: 10.1109/ICASSP.2018.8462286.

[17] Y. Liang, M. Elgendi, Z. Chen, and R. Ward, "Analysis: An optimal filter for short photoplethysmogram signals," *Sci. Data*, 2018, doi: 10.1038/sdata.2018.76.

[18] S. Ma, H. Ma, Y. Xu, S. Li, C. Lv, and M. Zhu, "A Low-Light Sensor Image Enhancement Algorithm Based on HSI Color Model ," *Sensors (Basel, Switzerland)* , vol. 18, no. 10. MDPI AG , Switzerland , p. 3583, 2018, doi:

10.3390/s18103583.

[19] S. Zaunseder, A. Trumpp, D. Wedekind, and H. Malberg, "Cardiovascular assessment by imaging photoplethysmography-a review," *Biomed. Tech.*, 2018, doi: 10.1515/bmt-2017-0119.

[20] W. Wang, S. Stuijk, and G. De Haan, "A Novel Algorithm for Remote Photoplethysmography: Spatial Subspace Rotation," *IEEE Trans. Biomed. Eng.*, 2016, doi: 10.1109/TBME.2015.2508602.

[21] R. M. Fouad, O. A. Omer, and M. H. Aly, "Optimizing Remote Photoplethysmography Using Adaptive Skin Segmentation for Real-Time Heart Rate Monitoring," *IEEE Access*, 2019, doi: 10.1109/ACCESS.2019.2922304.

8 APPENDIX

8.1 Real-Time Signal Processors

8.1.1 FIR Filter

```
import java.util.Arrays;

class FIR_Filter {
    private float[] coefficients, buffer;
    private int size, index = 0;

    void set_coefficients(float[] Coefficients) {
        coefficients = Coefficients;
        size = coefficients.length;
        buffer = new float[size];
    }

    float doFilter(float data) {
        buffer[index] = data;
        float[] array1 = reverse(Arrays.copyOfRange(buffer, 0, index));
        float[] array2 = reverse(Arrays.copyOfRange(buffer, index, size));
        int aLen = array1.length;
        int bLen = array2.length;
        float[] temporary = new float[aLen + bLen];
        System.arraycopy(array1, 0, temporary, 0, aLen);
        System.arraycopy(array2, 0, temporary, aLen, bLen);

        float output = sum(temporary, coefficients);

        if (index == size - 1) index = 0;
        else index++;
        return output;
    }

    private float sum(float[] first, float[] second) {
        int length = Math.min(first.length, second.length);
        float result = 0;

        for (int i = 0; i < length; i++) {
            result += first[i] * second[i];
        }
        return result;
    }

    private float[] reverse(float[] input) {
        float[] output = new float[input.length];
        int count = 0;
        for (int i = input.length - 1; i >= 0; i--) {
            output[count] = input[i];
            count++;
        }
        return output;
    }
}
```

8.1.2 IIR Filter

8.1.2.1 IIR2 Class

```
import java.util.List;

class IIR2Filter {
    // FIR coefficients
    private float b0, b1, b2;
    // IIR coefficients
    private float a0, a1, a2;
    private float buffer1, buffer2 = 0;

    void set_sos_coefficients(List<Float> sos) {
       b0 = sos.get(0); b1 = sos.get(1); b2 = sos.get(2);
       a0 = sos.get(3); a1 = sos.get(4); a2 = sos.get(5);
    }

    float doFilter(float data) {
       float acc_input = data - (buffer1*a1) - (buffer2*a2);
       float acc_output = (acc_input*b0) + (buffer1*b1) + (buffer2*b2);

       buffer2 = buffer1;
       buffer1 = acc_input;

       return acc_output;
    }
}
```

8.1.2.2 IIR Class [Cascades IIR2 Filters]

```
import java.util.ArrayList;
import java.util.List;

class IIRFilter {
    private List<IIR2Filter> slaves = new ArrayList<>();
    private int order;

    void set_sos_coefficients(List<List<Float>> sos_coefficients) {
        order = sos_coefficients.size();
        for (int i = 0; i < order; i++) {
           slaves.add(i, new IIR2Filter());
           slaves.get(i).set_sos_coefficients(sos_coefficients.get(i));
        }
    }

    float doFilter(float data) {
        float output = data;
        for (int i=0; i<order; i++) {
           output = slaves.get(i).doFilter(output);
        }
        return output;
    }
}
```

8.1.3 Matched Filter

```
import java.util.Arrays;

class MatchedFilter {
    private float[] coefficients;
    private int size;
    private float[] buffer;
    private int index = 0;

    void set_coefficients(float[] Coefficients) {
        coefficients = Coefficients;
        size = coefficients.length;
        buffer = new float[size];
    }

    float matched_filter(float data) {
        buffer[index] = data;
        float[] array1 = reverse(Arrays.copyOfRange(buffer, 0, index));
        float[] array2 = reverse(Arrays.copyOfRange(buffer, index, size));

        int aLen = array1.length;
        int bLen = array2.length;
        float[] temporary = new float[aLen + bLen];

        System.arraycopy(array1, 0, temporary, 0, aLen);
        System.arraycopy(array2, 0, temporary, aLen, bLen);

        float output = sum(temporary, coefficients);

        if (index == size - 1) index = 0;
        else index++;

        return output;
    }

    private float sum(float[] first, float[] second) {
        int length = Math.min(first.length, second.length);
        float result = 0;

        for (int i = 0; i < length; i++) {
            result += first[i] * second[i];
        }
        return result;
    }

    private float[] reverse(float[] input) {
        float[] output = new float[input.length];
        int count = 0;
        for (int i = input.length - 1; i >= 0; i--) {
            output[count] = input[i];
            count++;
        }
        return output;
    }
}
```

8.1.4 Constants

```
import android.graphics.Bitmap;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

class Constants {
    static List<String> savedBPMs = new ArrayList<>();
    static List<String> savedDateTimes = new ArrayList<>();
    static List<Bitmap> savedPlots = new ArrayList<>();

    // Matched filter coefficients (from template)
    static float[] float_coefficients = {(float) 0.00631364, (float) -0.03792876,
(float) -0.02550926,  (float) 0.13042721,  (float) 0.42606094,
            (float) 0.712838  ,  (float) 0.7865376 ,  (float) 0.5640064 ,  (float)
0.16566347, (float) -0.19206063,
            (float) -0.37875745, (float) -0.4173598 , (float) -0.39792314, (float)
-0.36837435, (float) -0.32127404,
            (float) -0.25116646, (float) -0.18220228, (float) -0.13577957, (float)
-0.10202894, (float) -0.06183188};

    static float[] FIR_coefficients = { (float) -1.15577091e-04, (float) -
6.05338384e-06, (float) -1.15577091e-04, (float) 1.16939048e-04,
            (float) 2.01344659e-03, (float) 3.16797021e-03, (float)-1.15577091e-
04, (float)-3.82797463e-03,
            (float) -1.15577091e-04,(float)  6.18990093e-03,(float) -1.15577091e-
04,(float) -1.74093089e-02,
            (float) -2.16450558e-02,(float) -5.19936522e-03,(float) -1.15577091e-
04,(float) -3.14430467e-02,
            (float) -6.42023418e-02,(float) -4.56894505e-02,(float) -1.15577091e-
04,(float) -1.64276912e-02,
            (float) -1.07833444e-01,(float) -1.44137197e-01,(float) -1.15577091e-
04,(float)  2.49878094e-01,
            (float) 3.74465713e-01, (float) 2.47650016e-01, (float)-1.15577091e-
04, (float)-1.40274964e-01,
            (float) -1.03958640e-01,(float) -1.56874805e-02,(float) -1.15577091e-
04,(float) -4.27110949e-02,
            (float) -5.93211922e-02,(float) -2.86946751e-02,(float) -1.15577091e-
04,(float) -4.62001897e-03,
            (float) -1.88635835e-02,(float) -1.48779203e-02,(float) -1.15577091e-
04,(float)  4.99716107e-03,
            (float) -1.15577091e-04,(float) -2.89706810e-03,(float) -1.15577091e-
04,(float)  2.01442620e-03,
            (float) 1.09459613e-03, (float)-1.14626808e-05, (float)-1.15577091e-
04, (float)-1.15577091e-04};

    static List<List<Float>> sos_butter_order4_bp = Arrays.asList(
            Arrays.asList((float) 0.02607772, (float) 0.05215544, (float)
0.02607772, (float) 1, (float) -0.60740687, (float) 0.19313234),
            Arrays.asList((float) 1, (float) 2, (float) 1, (float) 1, (float) -
1.47916952, (float) 0.57498111),
            Arrays.asList((float) 1, (float) -2, (float) 1, (float) 1, (float) -
0.44995709, (float) 0.58165366),
            Arrays.asList((float) 1, (float) -2, (float) 1, (float) 1, (float) -
1.79629148, (float) 0.86333651));
}
```

8.2 PPG History Activity

```
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.Toast;

public class ppgHistory extends AppCompatActivity {
    public ListView historyList;
    Intent mainActivity;
    SQLiteDatabase database;
    Cursor cursor;

    tech.glasgowneuro.attysecg.ListAdapter lAdapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ppg_histroy);

        database = openOrCreateDatabase("database", MODE_PRIVATE, null);
        cursor = database.rawQuery("Select * from RecordingsTable", null);
        cursor.moveToFirst();
        mainActivity = getIntent();

        Constants.savedBPMs.clear(); Constants.savedDateTimes.clear();
Constants.savedPlots.clear();

        for (int i=cursor.getPosition(); i < cursor.getCount(); i++) {
            String recordedBPM = cursor.getString(0);
            String recordedDateTime = cursor.getString(1);
            byte[] byteArray = cursor.getBlob(2);

            Log.i("Length", byteArray.length + "");

            Bitmap recordedPlot = BitmapFactory.decodeByteArray(byteArray, 0,
byteArray.length);

            Constants.savedBPMs.add(recordedBPM);
Constants.savedDateTimes.add(recordedDateTime);
Constants.savedPlots.add(recordedPlot);

            cursor.moveToNext();
        }

        historyList = findViewById(R.id.historyList);
        lAdapter = new ListAdapter(this, Constants.savedBPMs,
```

```
Constants.savedDateTimes, Constants.savedPlots);
        historyList.setAdapter(lAdapter);

        historyList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, final int
position, long id) {
                ImageView preview = new ImageView(ppgHistory.this);
                preview.setImageBitmap(Constants.savedPlots.get(position));

                AlertDialog.Builder builder = new
AlertDialog.Builder(ppgHistory.this);
                builder.setView(preview)
                        .setTitle(Constants.savedBPMs.get(position))
                        .setIcon(R.drawable.heart)
                        .setMessage(Constants.savedDateTimes.get(position))
                        .setCancelable(false)
                        .setPositiveButton("Delete", new
DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which)
{

deleteFromDatabase(Constants.savedDateTimes.get(position));
                                Constants.savedBPMs.remove(position);
                                Constants.savedDateTimes.remove(position);
                                Constants.savedPlots.remove(position);
                                lAdapter.notifyDataSetChanged();
                            }
                        })
                        .setNegativeButton("Keep", null)
                        .show();
            }
        });
    }

    public void deleteAll(View view) {
        if (Constants.savedBPMs.size() != 0) {
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Are you sure you want to delete all your
recordings?")
                    .setPositiveButton("Yes", new
DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialog, int which) {
                            deleteAllFromDatabase();
                            Constants.savedBPMs.clear();
                            Constants.savedDateTimes.clear();
                            Constants.savedPlots.clear();
                            lAdapter.notifyDataSetChanged();
                        }
                    })
                    .setNegativeButton("No", null)
                    .show();
        }
        else Toast.makeText(this, "No recordings to delete",
Toast.LENGTH_SHORT).show();
    }
```

```
    public void back(View view) {
        finish();
    }

    public void deleteFromDatabase(String DateTime) {
        database.delete("RecordingsTable", "DateTimes=?", new String[]{DateTime});
    }

    public void deleteAllFromDatabase() {
        database.delete("RecordingsTable", null, null);
    }
}
```

8.2.1 ListAdapter [for PPG History Activity]

```
import android.content.Context;
import android.graphics.Bitmap;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

import java.util.List;

public class ListAdapter extends BaseAdapter {
    private Context context;
    private final List<String> heartRate;
    private final List<String> dateTimes;
    private final List<Bitmap> plots;

    ListAdapter(Context context, List<String> heartRate, List<String> dateTimes,
List<Bitmap> plots){
        //super(context, R.layout.single_list_app_item, utilsArrayList);
        this.context = context;
        this.heartRate = heartRate;
        this.dateTimes = dateTimes;
        this.plots = plots;
    }

    @Override
    public int getCount() {
        return heartRate.size();
    }

    @Override
    public Object getItem(int i) {
        return i;
    }

    @Override
    public long getItemId(int i) {
        return i;
    }
```

```
    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView, @NonNull
ViewGroup parent) {


        ViewHolder viewHolder;

        if (convertView == null) {
            viewHolder = new ViewHolder();
            LayoutInflater inflater = LayoutInflater.from(context);
            convertView = inflater.inflate(R.layout.history_item, parent, false);
           viewHolder.bpm = convertView.findViewById(R.id.bpm);
            viewHolder.date_time = convertView.findViewById(R.id.date_time);
            viewHolder.plot = convertView.findViewById(R.id.plot);

            convertView.setTag(viewHolder);
        } else {
            viewHolder = (ViewHolder) convertView.getTag();
        }

        viewHolder.bpm.setText(heartRate.get(position));
        viewHolder.date_time.setText(dateTimes.get(position));
        viewHolder.plot.setImageBitmap(plots.get(position));

        return convertView;
    }

    private static class ViewHolder {
        TextView bpm;
        TextView date_time;
        ImageView plot;
    }
}
```

8.2.2 PPG History Layout XML

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="?attr/colorPrimary"
    tools:context=".ppgHistory">

    <ListView
        android:id="@+id/historyList"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="55dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="10dp">
        <Button
            android:layout_width="match_parent"
            android:layout_height="35dp"
            android:layout_weight="1"
            android:onClick="back"
            android:background="@drawable/button"
            android:layout_marginStart="10dp"
            android:layout_marginEnd="5dp"
            android:text="BACK"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"/>
        <Button
            android:layout_width="match_parent"
            android:layout_height="35dp"
            android:layout_weight="1"
            android:onClick="deleteAll"
            android:background="@drawable/button"
            android:layout_marginStart="5dp"
            android:layout_marginEnd="10dp"
            android:text="DELETE ALL"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintBottom_toBottomOf="parent"/>
    </LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

8.2.2.1 History List Item Layout XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingStart="8dp"
    android:paddingEnd="0dp">

    <ImageView
```

```
        android:layout_width="20dp"
        android:layout_height="20dp"
        android:layout_marginBottom="5dp"
        android:src="@drawable/heart"
        android:layout_alignStart="@+id/bpm"
        android:layout_alignBottom="@+id/bpm"
        android:scaleType="fitCenter"/>

    <TextView
        android:id="@+id/bpm"
        android:text="60bpm"
        android:textColor="#000"
        android:textAlignment="viewEnd"
        android:textSize="20sp"
        android:layout_marginTop="10dp"
        android:maxLines="1"
        android:layout_width="100dp"
        android:layout_height="30dp"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:id="@+id/date_time"
        android:text="Sat, 18 Aug 2020 @ 13:40:22"
        android:textColor="#000"
        android:alpha="0.5"
        android:textSize="12sp"
        android:textAlignment="viewEnd"
        android:maxLines="2"
        android:layout_width="100dp"
        android:layout_height="30dp"
        android:layout_below="@+id/bpm"
        android:layout_alignStart="@+id/bpm" />

    <ImageView
        android:id="@+id/plot"
        android:layout_height="90dp"
        android:layout_width="match_parent"
        android:layout_toEndOf="@+id/bpm"
        android:layout_marginTop="10dp"
        android:layout_alignParentEnd="true"
        android:scaleType="fitCenter"/>
</RelativeLayout>
```

8.3 PPG Activity

```
import android.Manifest;
import android.animation.Animator;
import android.animation.ObjectAnimator;
import android.annotation.SuppressLint;
import android.content.ContentValues;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.database.sqlite.SQLiteDatabase;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.graphics.ImageFormat;
import android.graphics.Matrix;
import android.graphics.Point;
import android.graphics.RectF;
import android.graphics.SurfaceTexture;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCaptureSession;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraDevice;
import android.hardware.camera2.CaptureRequest;
import android.hardware.camera2.params.StreamConfigurationMap;
import android.media.ImageReader;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.SystemClock;
import android.os.VibrationEffect;
import android.os.Vibrator;
import android.util.Log;
import android.util.Size;
import android.view.LayoutInflater;
import android.view.Surface;
import android.view.TextureView;
import android.view.View;
import android.view.ViewGroup;
import android.view.animation.LinearInterpolator;
import android.widget.Button;
import android.widget.Chronometer;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.core.content.ContextCompat;
import androidx.fragment.app.Fragment;
import com.androidplot.xy.BoundaryMode;
import com.androidplot.xy.LineAndPointFormatter;
import com.androidplot.xy.SimpleXYSeries;
import com.androidplot.xy.StepMode;
import com.androidplot.xy.XYGraphWidget;
import com.androidplot.xy.XYPlot;
import java.io.ByteArrayOutputStream;
```

```
import java.io.File;
import java.io.FileWriter;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.Objects;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import static android.content.Context.MODE_PRIVATE;


/**
 * Created by Mustafa Biyikli on 25/06/2020
 */

public class PPGPlotFragment extends Fragment {

    private SQLiteDatabase database;
    private Intent historyActivity;

    private TextureView mTextureView;
    private CameraCaptureSession mCaptureSession;
    private CameraDevice mCameraDevice;
    private Size mPreviewSize;
    private HandlerThread mBackgroundThread;
    private Handler mBackgroundHandler;
    private ImageReader mImageReader;
    private CaptureRequest.Builder mPreviewRequestBuilder;
    private CaptureRequest mPreviewRequest;
    private Semaphore mCameraOpenCloseLock = new Semaphore(1);

    private ObjectAnimator progressAnimator;
    private ProgressBar loadBar;
    private ProgressBar spinner;

    private Highpass highpass = null;
    private MatchedFilter matchedFilter = null;
    private IIRFilter iirFilter = null;
    private FIR_Filter firFilter = null;

    private TextView rgbView;
    private TextView bpmView;
    @SuppressLint("StaticFieldLeak")
    static TextView fps;
    private Chronometer chronometer;
    private Bitmap bitmap;

    private Button flashButton;
    private Button recordButton;

    private Timer loopTimer;
```

```
    private boolean isFlashON = false;
    static boolean isRecording = false;
    private boolean aboveThreshold = true;
    private boolean bpmInitialLoop = true;
    static boolean fpsAnalyserON = true;
    static boolean thresholdAnalyserON = false;
    private int loop_counter = 0;
    private int fingerOnCounter = 0;
    private int lastBPM;

    private List<Float> U_raw_data = new ArrayList<>();
    private List<Float> U_filtered_data = new ArrayList<>();
    private List<Float> U_matched_data = new ArrayList<>();
    private List<Float> THRESHOLD_data = new ArrayList<>();
    static List<Float> ECG_II_data = new ArrayList<>();

    private String TAG = "PPGPlotFragment";
    private String flashONTag = "FLASH OFF";
    private String flashOFFTag = "FLASH ON";

    private final double maxRed = Math.pow(2, 8);
    private final int HISTORY_SIZE = AttysECG.fpsFixed.getLower() * 6;
    private long timeStamp;
    private long bogusStamp = 0;
    private double bpmTotal = 0;
    private int beatCounter = 0;

    private SimpleXYSeries redHistorySeries = null;
    private SimpleXYSeries thresholdHistorySeries = null;

    private XYPlot ppgPlot = null;

    private Vibrator vibrator;
    private float THRESHOLD;

     private final TextureView.SurfaceTextureListener mSurfaceTextureListener
            = new TextureView.SurfaceTextureListener() {

        @Override
        public void onSurfaceTextureAvailable(SurfaceTexture texture, int width,
int height) {
            openCamera(width, height);
        }

        @Override
        public void onSurfaceTextureSizeChanged(SurfaceTexture texture, int width,
int height) {
            configureTransform(width, height);
        }

        @Override
        public boolean onSurfaceTextureDestroyed(SurfaceTexture texture) {
            return true;
        }



        @Override
        public void onSurfaceTextureUpdated(SurfaceTexture texture) {
```

```
            bitmap = mTextureView.getBitmap();

            Objects.requireNonNull(getActivity()).runOnUiThread(new Runnable() {
                float redBucket = 0;
                float greenBucket = 0;
                float blueBucket = 0;
                float pixelCount = 0;
                @SuppressLint("SetTextI18n")
                @Override
                public void run() {
                    for (int y = 0; y < bitmap.getHeight(); y+=bitmap.getHeight()
/ 20) {
                        for (int x = 0; x < bitmap.getWidth();
x+=bitmap.getWidth() / 20) {
                            int pixel = bitmap.getPixel(x, y);
                            pixelCount++;

                            redBucket += Color.red(pixel);
                            greenBucket += Color.green(pixel);
                            blueBucket += Color.blue(pixel);
                        }
                    }
                    float r = redBucket/pixelCount;
                    float g = greenBucket/pixelCount;
                    float b = blueBucket/pixelCount;

                    // double Y = 0.257*r + 0.504*g + 0.098*b + 16;
                    double U = -0.148*r - 0.291*g + 0.439*b + 128;
                    // double V =  0.439*r - 0.368*g - 0.071*b + 128;

                    rgbView.setText("R:" + Math.round(r) + " G:" + Math.round(g) +
" B:" + Math.round(b));

                    float maxValue = 0, localMax = 0;
                    float minValue = 255;

                    for (int i = 0; i < redHistorySeries.size(); i++) {
                        float val = redHistorySeries.getY(i).floatValue();
                        if (val > maxValue) maxValue = val;
                        if (val < minValue) minValue = val;
                    }
                    ppgPlot.setRangeBoundaries(minValue - 0.001, maxValue + 0.001,
BoundaryMode.FIXED);

                    for (int i = redHistorySeries.size()-1; i >=
(redHistorySeries.size() - redHistorySeries.size()/10); i--) {
                        float val = redHistorySeries.getY(i).floatValue();
                        if (val > localMax) localMax = val;
                    }

                    // DC removal with high-pass filter
                    //float r_filtered = highpass.filter((float) U);
                    //float U_filtered = firFilter.doFilter((float) U);
                    float U_filtered = iirFilter.doFilter((float) U);

                    // Matched filter
                    float U_matched = isFlashON ?
matchedFilter.matched_filter(U_filtered) : 0;
```

```
                    THRESHOLD = isFlashON ? (float) 0.7 * localMax : 0;

                    // Finger ON
                    if (r >= 150 && b <= 10) {
                        fingerOnCounter++;
                        if (spinner.getAlpha() == 1) {
                            spinner.setAlpha(0);
                            Toast.makeText(getContext(), "Measuring - keep
steady", Toast.LENGTH_SHORT).show();
                        }
                        if (fingerOnCounter > AttysECG.fpsFixed.getLower() * 2) {
                            if (loadBar.getProgress() == 0) progressLoader(true);
                            addValue(U_matched, THRESHOLD);
                            heartRate_detect(U_matched, THRESHOLD);
                        } else {
                            addValue(0, 0);
                        }
                    } else {
                        // Finger OFF
                        addValue(0, 0);
                        if (loadBar.getProgress() != 0) {
                            progressLoader(false);
                            lastBPM =
Integer.parseInt(bpmView.getText().toString());
                            bpmView.setText("");
                            spinner.setAlpha(1);
                            fingerOnCounter = 0;
                        }
                    }

                    // Recorder
                    if (isRecording) {
                        U_raw_data.add((float) U);
                        U_filtered_data.add(U_filtered);
                        U_matched_data.add(U_matched);
                        THRESHOLD_data.add(THRESHOLD);
                    }
                }
            });
            loop_counter++; // used for FPS calculation
        }

    };

    @SuppressLint("SetTextI18n")
    private void heartRate_detect(float signal, float threshold) {
        // bogusStamp > 333 checks if the two detected peaks = heart-rate > 180
        if (signal >= threshold && aboveThreshold && (new Date().getTime() -
bogusStamp) > 333) {
            BeepGenerator.doBeep();
            if (bpmInitialLoop) {
                timeStamp = new Date().getTime();
                bpmInitialLoop = false;
            } else {
                double bpm = 60.0 / ((new Date().getTime() - timeStamp) / 1000.0);
                if (bpm > 40 && bpm < 180) {
                    if (spinner.getAlpha() != 0) spinner.setAlpha(0);
                    beatCounter++;
                    bpmTotal += bpm;
```

```
                    bpmView.setText(Integer.toString((int) bpmTotal/beatCounter));
                }
                bpmInitialLoop = true;
            }
            aboveThreshold = false;
        } else if (signal < threshold && !aboveThreshold) {
            bogusStamp = new Date().getTime();
            aboveThreshold = true;
        }
    }

    private void progressLoader(final boolean start) {
        if (start) {
            progressAnimator.removeAllListeners();
            progressAnimator.addListener(new Animator.AnimatorListener() {
                @Override
                public void onAnimationStart(Animator animation) {

                }

                @Override
                public void onAnimationEnd(Animator animation) {
                    loadBar.setProgress(0);
                    ppgPlot.setDrawingCacheEnabled(true);
                    ppgPlot.buildDrawingCache(true);
                    Bitmap cache = ppgPlot.getDrawingCache();
                    final Bitmap bitmap = Bitmap.createBitmap(cache);
                    ppgPlot.redraw();
                    ppgPlot.setDrawingCacheEnabled(false);

                    final String recordedDateTime = new SimpleDateFormat("EEE, d
MMM yyyy\nHH:mm:ss", Locale.getDefault()).format(new Date());

                    ImageView preview = new ImageView(getContext());
                    preview.setImageBitmap(bitmap);

                    AlertDialog.Builder builder = new
AlertDialog.Builder(Objects.requireNonNull(getContext()));
                    builder.setView(preview)
                            .setTitle(bpmView.getText() + "bpm")
                            .setIcon(R.drawable.heart)
                            .setMessage(recordedDateTime)
                            .setCancelable(false)
                            .setPositiveButton("SAVE", new
DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog, int
which) {
                                    ByteArrayOutputStream bStream = new
ByteArrayOutputStream();
                                    bitmap.compress(Bitmap.CompressFormat.PNG,
100, bStream);
                                    byte[] byteArray = bStream.toByteArray();

                                    ContentValues contentValues = new
ContentValues();
                                    contentValues.put("BPMs", lastBPM + "bpm");
                                    contentValues.put("DateTimes",
recordedDateTime);
```

```
                                    contentValues.put("Plots", byteArray);

                                    database.insert("RecordingsTable", null,
contentValues);
                                    startActivity(historyActivity);
                                }
                            })
                            .setNegativeButton("DISCARD", new
DialogInterface.OnClickListener() {
                                @Override
                                public void onClick(DialogInterface dialog, int
which) {
                                    flashON(true);
                                }
                            })
                            .show();
                    flashON(false);
                }

                @Override
                public void onAnimationCancel(Animator animation) {

                }

                @Override
                public void onAnimationRepeat(Animator animation) {

                }
            });
            progressAnimator.start();
        }
        else {
            if (progressAnimator.isRunning()) {
                progressAnimator.removeAllListeners();
                progressAnimator.end();
                loadBar.setProgress(0);
            }
        }
    }

    private void recorder() {
        vibrate(100);
        if (!isRecording) {
            // Start recording & start timer without blocking UI
            AsyncTask.execute(new Runnable() {
                @SuppressLint("SetTextI18n")
                @Override
                public void run() {
                    isRecording = true;
                    recordButton.setText("SAVE PPG/ECG");
                    chronometer.setBase(SystemClock.elapsedRealtime());
                    chronometer.setAlpha(1);
                    chronometer.start();
                }
            });

        } else {
            // Save the RGB/ECG data to .txt without blocking UI
            AsyncTask.execute(new Runnable() {
```

```
                @SuppressLint("SetTextI18n")
                @Override
                public void run() {
                    isRecording = false;
                    recordButton.setText("RECORD");
                    chronometer.setAlpha(0.5f);
                    chronometer.stop();

                    File file = new File(AttysECG.ATTYSDIR, "PPG & ECG
recordings");
                    if (!file.exists()) {
                        boolean wasSuccessful = file.mkdir();
                        if (!wasSuccessful) Log.e("File.mkdir()", "Failed");
                    }

                    try {
                        File ppgFile = new File(file, "PPG" + "_" + new
Date().getTime());
                        File ecgFile = new File(file, "ECG" + "_" + new
Date().getTime());

                        FileWriter writer_ppg = new FileWriter(ppgFile);
                        FileWriter writer_ecg = new FileWriter(ecgFile);

                        writer_ppg.append("Y[U]V_raw:
").append(U_raw_data.toString()).append("\n")
                                    .append("Y[U]V_filtered:
").append(U_filtered_data.toString()).append("\n")
                                    .append("Y[U]V_matched:
").append(U_matched_data.toString()).append("\n");
                        writer_ppg.append("threshold:
").append(THRESHOLD_data.toString());
                        writer_ecg.append("Einthoven
II").append(ECG_II_data.toString());

                        writer_ppg.flush();
                        writer_ecg.flush();
                        writer_ppg.close();
                        writer_ecg.close();

                    } catch (Exception e) {
                        Log.e("PPG/ECG recording error", "Could not save!");
                    }

                    THRESHOLD_data.clear();
                    ECG_II_data.clear(); // Clear the ECG II data
                    U_raw_data.clear();
                    U_filtered_data.clear();
                    U_matched_data.clear();
                }
            });
        }
    }

     private final CameraDevice.StateCallback mStateCallback = new
CameraDevice.StateCallback() {

        @Override
        public void onOpened(@NonNull CameraDevice cameraDevice) {
```

```
            // This method is called when the camera is opened.  We start camera
preview here.
            mCameraOpenCloseLock.release();
            mCameraDevice = cameraDevice;
            createCameraPreviewSession();
        }

        @Override
        public void onDisconnected(@NonNull CameraDevice cameraDevice) {
            mCameraOpenCloseLock.release();
            cameraDevice.close();
            mCameraDevice = null;
        }

        @Override
        public void onError(@NonNull CameraDevice cameraDevice, int error) {
            mCameraOpenCloseLock.release();
            cameraDevice.close();
            mCameraDevice = null;
            Objects.requireNonNull(getActivity()).finish();
        }

    };

    // Camera2 Related Methods
     private void openCamera(int width, int height) {
        if
(ContextCompat.checkSelfPermission(Objects.requireNonNull(getContext()),
Manifest.permission.CAMERA)
                != PackageManager.PERMISSION_GRANTED) {
            new AttysECG().requestCameraPermission();
            return;
        }
        setUpCameraOutputs(width, height);
        configureTransform(width, height);
        try {
            if (!mCameraOpenCloseLock.tryAcquire(2500, TimeUnit.MILLISECONDS)) {
                throw new RuntimeException("Time out waiting to lock camera
opening.");
            }

AttysECG.mCameraManager.openCamera(AttysECG.mCameraManager.getCameraIdList()[0],
mStateCallback, mBackgroundHandler);
        } catch (CameraAccessException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            throw new RuntimeException("Interrupted while trying to lock camera
opening.", e);
        }
    }

    // Closes the current camera
     private void closeCamera() {
        try {
            mCameraOpenCloseLock.acquire();
            if (null != mCaptureSession) {
                mCaptureSession.close();
                mCaptureSession = null;
            }
```

```
            if (null != mCameraDevice) {
                mCameraDevice.close();
                mCameraDevice = null;
            }
            if (null != mImageReader) {
                mImageReader.close();
                mImageReader = null;
            }
        } catch (InterruptedException e) {
            throw new RuntimeException("Interrupted while trying to lock camera
closing.", e);
        } finally {
            mCameraOpenCloseLock.release();
        }
    }

    // Sets up member variables related to camera

     private void setUpCameraOutputs(int width, int height) {
        int MAX_PREVIEW_WIDTH = 1920;
        int MAX_PREVIEW_HEIGHT = 1080;
        try {
            for (String cameraId : AttysECG.mCameraManager.getCameraIdList()) {
                CameraCharacteristics characteristics
                        =
AttysECG.mCameraManager.getCameraCharacteristics(cameraId);

                // We don't use a front facing camera in this sample.
                Integer facing =
characteristics.get(CameraCharacteristics.LENS_FACING);
                if (facing != null && facing ==
CameraCharacteristics.LENS_FACING_FRONT) {
                    continue;
                }

                StreamConfigurationMap map = characteristics.get(
                        CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
                if (map == null) {
                    continue;
                }

                // For still image captures, we use the largest available size.
                Size largest =
Collections.max(Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)), new
CompareSizesByArea());
                mImageReader = ImageReader.newInstance(largest.getWidth(),
largest.getHeight(), ImageFormat.JPEG, /*maxImages*/2);
                mImageReader.setOnImageAvailableListener(null,
mBackgroundHandler);

                Point displaySize = new Point();

Objects.requireNonNull(getActivity()).getWindowManager().getDefaultDisplay().getSi
ze(displaySize);

                int maxPreviewWidth = displaySize.x;
                int maxPreviewHeight = displaySize.y;

                if (maxPreviewWidth > MAX_PREVIEW_WIDTH) {
```

```
                    maxPreviewWidth = MAX_PREVIEW_WIDTH;
                }

                if (maxPreviewHeight > MAX_PREVIEW_HEIGHT) {
                    maxPreviewHeight = MAX_PREVIEW_HEIGHT;
                }

                mPreviewSize =
chooseOptimalSize(map.getOutputSizes(SurfaceTexture.class), width, height,
maxPreviewWidth, maxPreviewHeight, largest);

                AttysECG.mCameraId = cameraId;
                return;
            }
        } catch (CameraAccessException e) {
            e.printStackTrace();
        } catch (NullPointerException e) {
            Toast.makeText(getContext(), "Camera2 API not supported on this
device", Toast.LENGTH_LONG).show();
        }
    }

    // Flash related methods

    private void flashON(final boolean SWITCH) {
        AsyncTask.execute(new Runnable() {
            @Override
            public void run() {
                try {
                    mCaptureSession.stopRepeating();
                    isFlashON = SWITCH;
                    int flashMode = isFlashON ? CaptureRequest.FLASH_MODE_TORCH :
CaptureRequest.FLASH_MODE_OFF;
                    String flashButtonText = isFlashON ? flashONTag : flashOFFTag;
                    mPreviewRequestBuilder.set(CaptureRequest.FLASH_MODE,
flashMode);
                    flashButton.setText(flashButtonText);

mCaptureSession.setRepeatingRequest(mPreviewRequestBuilder.build(), null,
mBackgroundHandler);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    // Vibrator
    private void vibrate(int milliseconds) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            vibrator.vibrate(VibrationEffect.createOneShot(milliseconds,
VibrationEffect.DEFAULT_AMPLITUDE));
        } else {
            //deprecated in API 26
            vibrator.vibrate(milliseconds);
        }
    }

    // Creates camera capture session for preview
```

```
    private void createCameraPreviewSession() {
        try {
            SurfaceTexture texture = mTextureView.getSurfaceTexture();
            assert texture != null;

            // We configure the size of default buffer to be the size of camera
preview we want.
            texture.setDefaultBufferSize(mPreviewSize.getWidth(),
mPreviewSize.getHeight());

            // This is the output Surface we need to start preview.
            Surface surface = new Surface(texture);

            // We set up a CaptureRequest.Builder with the output Surface.
            mPreviewRequestBuilder =
mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_PREVIEW);
            mPreviewRequestBuilder.addTarget(surface);

            // Here, we create a CameraCaptureSession for camera preview.
            mCameraDevice.createCaptureSession(Arrays.asList(surface,
mImageReader.getSurface()), new CameraCaptureSession.StateCallback() {

                        @Override
                        public void onConfigured(@NonNull CameraCaptureSession
cameraCaptureSession) {
                            // The camera is already closed
                            if (null == mCameraDevice) {
                                return;
                            }

                            // When the session is ready, we start displaying the
preview.
                            mCaptureSession = cameraCaptureSession;
                            try {
                                // Auto focus should be continuous for camera
preview.

mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AF_MODE,
CaptureRequest.CONTROL_AF_MODE_CONTINUOUS_PICTURE);

                                // FPS range set

mPreviewRequestBuilder.set(CaptureRequest.CONTROL_AE_TARGET_FPS_RANGE,
AttysECG.fpsFixed);

                                // Finally, we start displaying the camera
preview.
                                mPreviewRequest = mPreviewRequestBuilder.build();

mCaptureSession.setRepeatingRequest(mPreviewRequest, null, mBackgroundHandler);
                            } catch (CameraAccessException e) {
                                e.printStackTrace();
                            }
                        }

                        @Override
                        public void onConfigureFailed(
                                @NonNull CameraCaptureSession
```

```
cameraCaptureSession) {
                            Toast.makeText(getContext(), "Failed",
Toast.LENGTH_SHORT).show();
                        }
                    }, null
            );
        } catch (CameraAccessException e) {
            e.printStackTrace();
        }
    }

    // Background thread / handler methods
     private void startBackgroundThread() {
        mBackgroundThread = new HandlerThread("CameraBackground");
        mBackgroundThread.start();
        mBackgroundHandler = new Handler(mBackgroundThread.getLooper());
    }

     private void stopBackgroundThread() {
        mBackgroundThread.quitSafely();
        try {
            mBackgroundThread.join();
            mBackgroundThread = null;
            mBackgroundHandler = null;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Utility methods
     private Size chooseOptimalSize(Size[] choices, int textureViewWidth,
                                    int textureViewHeight, int maxWidth, int
maxHeight, Size aspectRatio) {

        // Collect the supported resolutions that are at least as big as the
preview Surface
        List<Size> bigEnough = new ArrayList<>();
        // Collect the supported resolutions that are smaller than the preview
Surface
        List<Size> notBigEnough = new ArrayList<>();
        int w = aspectRatio.getWidth();
        int h = aspectRatio.getHeight();
        for (Size option : choices) {
            if (option.getWidth() <= maxWidth && option.getHeight() <= maxHeight
&&
                    option.getHeight() == option.getWidth() * h / w) {
                if (option.getWidth() >= textureViewWidth &&
                        option.getHeight() >= textureViewHeight) {
                    bigEnough.add(option);
                } else {
                    notBigEnough.add(option);
                }
            }
        }

        if (bigEnough.size() > 0) {
            return Collections.min(bigEnough, new CompareSizesByArea());
        } else if (notBigEnough.size() > 0) {
            return Collections.max(notBigEnough, new CompareSizesByArea());
```

```
        } else {
            Log.e("Camera2", "Couldn't find any suitable preview size");
            return choices[0];
        }
    }

     private void configureTransform(int viewWidth, int viewHeight) {
        if (null == mTextureView || null == mPreviewSize) {
            return;
        }
        int rotation =
Objects.requireNonNull(getActivity()).getWindowManager().getDefaultDisplay().getRo
tation();
        Matrix matrix = new Matrix();
        RectF viewRect = new RectF(0, 0, viewWidth, viewHeight);
        RectF bufferRect = new RectF(0, 0, mPreviewSize.getHeight(),
mPreviewSize.getWidth());
        float centerX = viewRect.centerX();
        float centerY = viewRect.centerY();
        if (Surface.ROTATION_90 == rotation || Surface.ROTATION_270 == rotation) {
            bufferRect.offset(centerX - bufferRect.centerX(), centerY -
bufferRect.centerY());
            matrix.setRectToRect(viewRect, bufferRect, Matrix.ScaleToFit.FILL);
            float scale = Math.max(
                    (float) viewHeight / mPreviewSize.getHeight(),
                    (float) viewWidth / mPreviewSize.getWidth());
            matrix.postScale(scale, scale, centerX, centerY);
            matrix.postRotate(90 * (rotation - 2), centerX, centerY);
        } else if (Surface.ROTATION_180 == rotation) {
            matrix.postRotate(180, centerX, centerY);
        }
        mTextureView.setTransform(matrix);
    }

    static class CompareSizesByArea implements Comparator<Size> {

        @Override
        public int compare(Size lhs, Size rhs) {
            // We cast here to ensure the multiplications won't overflow
            return Long.signum((long) lhs.getWidth() * lhs.getHeight() -
                    (long) rhs.getWidth() * rhs.getHeight());
        }
    }

    @Override
    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container,
                             Bundle savedInstanceState) {
        historyActivity = new Intent(getActivity(), ppgHistory.class);
        database =
Objects.requireNonNull(getActivity()).openOrCreateDatabase("database",
MODE_PRIVATE, null);
        database.execSQL("CREATE TABLE IF NOT EXISTS RecordingsTable(BPMs VARCHAR,
DateTimes VARCHAR, Plots BLOB);");

        // Init high-pass filter
        highpass = new Highpass();
        matchedFilter = new MatchedFilter();
        matchedFilter.set_coefficients(Constants.float_coefficients);
```

```
        // Init FIR Filter
        firFilter = new FIR_Filter();
        firFilter.set_coefficients(Constants.FIR_coefficients);

        // Init IIR Filter
        iirFilter = new IIRFilter();
        iirFilter.set_sos_coefficients(Constants.sos_butter_order4_bp);

        // Init beep generator
        new BeepGenerator();

        AlertDialog.Builder dlgAlert  = new
AlertDialog.Builder(Objects.requireNonNull(requireActivity()),
R.style.CustomDialog);
        dlgAlert.setIcon(R.drawable.fingertip);
        dlgAlert.setMessage("Place your index fingertip over the rear camera to
analyse your heart-rate\n\n" +
                "You can use the record PPG/ECG button to simultaneously record
PPG & ECG data to a .txt file as an array\n\n" +
                "Recordings saved to: 'Phone/attys/PPG & ECG recordings' folder");
        dlgAlert.setTitle("PPG Info");
        dlgAlert.setPositiveButton("OK", null);
        dlgAlert.setCancelable(false);
        dlgAlert.setPositiveButton("Ok",
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int which) {
                        flashON(true);
                    }
                });
        dlgAlert.create().show();

        Log.d(TAG, "onCreate, creating Fragment");
        vibrator = (Vibrator)
Objects.requireNonNull(getActivity()).getSystemService(Context.VIBRATOR_SERVICE);

        if (container == null) {
            return null;
        }

        final View view = inflater.inflate(R.layout.ppgplotfragment, container,
false);

        loadBar = view.findViewById(R.id.loadBar);
        loadBar.setMax(100 * 100);
        progressAnimator = ObjectAnimator.ofInt(loadBar, "progress", 0, 100 *
100);
        progressAnimator.setDuration(15000);
        progressAnimator.setInterpolator(new LinearInterpolator());
        spinner = view.findViewById(R.id.spinner);

        mTextureView = view.findViewById(R.id.texture);
        rgbView = view.findViewById(R.id.rgbView);
        bpmView = view.findViewById(R.id.bpmPPG);
        fps = view.findViewById(R.id.fpsView);
        chronometer = view.findViewById(R.id.chronometer);
        flashButton = view.findViewById(R.id.flashButton);
        flashButton.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                vibrate(200);
                if
(Objects.requireNonNull(getActivity()).getPackageManager().hasSystemFeature(Packag
eManager.FEATURE_CAMERA_FLASH)) {
                    if (!isFlashON) {
                        flashON(true);
                    } else {
                        flashON(false);
                    }
                } else {
                    Toast.makeText(getContext(), "No flash available on this
device", Toast.LENGTH_SHORT).show();
                }
            }
        });

        recordButton = view.findViewById(R.id.recordButton);
        recordButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                recorder();
            }
        });

        // setup the APR Levels plot:
        ppgPlot = view.findViewById(R.id.ppgPlotView);

        // Set padding & colors
        XYGraphWidget ppgGraph = ppgPlot.getGraph();
        ppgGraph.setDomainGridLinePaint(null);
        ppgGraph.setRangeGridLinePaint(null);
        ppgPlot.setBorderPaint(null);
        ppgPlot.setPlotMargins(0, 0, 0, 0);
        redHistorySeries = new SimpleXYSeries("pulse");
        thresholdHistorySeries = new SimpleXYSeries("threshold");
        redHistorySeries.useImplicitXVals();
        thresholdHistorySeries.useImplicitXVals();
        ppgPlot.setDomainBoundaries(0, HISTORY_SIZE, BoundaryMode.FIXED);
        ppgPlot.setRangeBoundaries(-maxRed/2, maxRed/2, BoundaryMode.FIXED);

        LineAndPointFormatter redLineAndPointFormatter = new
LineAndPointFormatter(Color.rgb(255, 0, 0), null, null, null);
        redLineAndPointFormatter.getLinePaint().setStrokeWidth(4);
        ppgPlot.addSeries(redHistorySeries, redLineAndPointFormatter);

        ppgPlot.addSeries(thresholdHistorySeries,
                new LineAndPointFormatter(
                        Color.argb(100, 0, 0, 0), null, null, null));

        ppgPlot.getLegend().setDrawIconBackgroundEnabled(false);
        ppgPlot.setDomainStep(StepMode.INCREMENT_BY_VAL, 10);

        return view;
    }

    private synchronized void addValue(final float v, final float t) {
        if (redHistorySeries == null) {
            if (Log.isLoggable(TAG, Log.VERBOSE)) {
```

```
                Log.v(TAG, "redHistorySeries == null");
            }
            return;
        }

        // get rid the oldest sample in history:
        if (redHistorySeries.size() > HISTORY_SIZE) {
            redHistorySeries.removeFirst();
            thresholdHistorySeries.removeFirst();
        }

        // add the latest history sample:
        redHistorySeries.addLast(null, v);
        if (thresholdAnalyserON) thresholdHistorySeries.addLast(null, t);
        else thresholdHistorySeries.addLast(null, null);

        ppgPlot.redraw();
    }

    @Override
    public void onResume() {
        super.onResume();
        startBackgroundThread();
        if (mTextureView.isAvailable()) {
            openCamera(mTextureView.getWidth(), mTextureView.getHeight());
        } else {
            mTextureView.setSurfaceTextureListener(mSurfaceTextureListener);
        }
        loopTimer = new Timer();
        loopTimer.scheduleAtFixedRate(new TimerTask() {
            @Override
            public void run() {
                Objects.requireNonNull(getActivity()).runOnUiThread(new Runnable()
{
                    @SuppressLint("SetTextI18n")
                    @Override
                    public void run() {
                        if(fpsAnalyserON) fps.setText("FPS " + loop_counter);
                        loop_counter = 0;
                    }
                });
            }
        }, 0, 1000);
        flashON(true);
    }

    @Override
    public void onPause() {
        super.onPause();
        flashON(false);
        loopTimer.cancel();
        closeCamera();
        stopBackgroundThread();
    }
}
```

8.3.1 PPG Activity Layout XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:ap="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    style="@style/APDefacto.Light"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_gravity="center"
    android:background="?attr/colorPrimary"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="25dp"
        android:baselineAligned="false"
        android:orientation="horizontal">

        <Chronometer
            android:id="@+id/chronometer"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginStart="10dp"
            android:layout_marginTop="5dp"
            android:layout_weight="0.3"
            android:alpha="0.5"
            android:textAlignment="center" />

        <TextView
            android:id="@+id/fpsView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginTop="5dp"
            android:layout_weight="0.4"
            android:textAlignment="center"
            android:textColor="#555" />

        <TextView
            android:id="@+id/rgbView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginTop="5dp"
            android:layout_marginEnd="10dp"
            android:layout_weight="0.3"
            android:textAlignment="center"
            android:textColor="#555" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/recordButton"
            android:layout_width="match_parent"
            android:layout_height="35dp"
            android:layout_marginStart="10dp"
```

```
            android:layout_marginEnd="5dp"
            android:layout_marginTop="12.5dp"
            android:layout_weight="0.5"
            android:background="@drawable/button"
            android:text="RECORD PPG/ECG" />

        <RelativeLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0.75">
            <TextureView
                android:id="@+id/texture"
                android:layout_width="100px"
                android:layout_height="100px"
                android:layout_centerInParent="true" />
            <ImageView
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_centerInParent="true"
                ap:srcCompat="@drawable/ring"/>
        </RelativeLayout>

        <Button
            android:id="@+id/flashButton"
            android:layout_width="match_parent"
            android:layout_height="35dp"
            android:layout_marginEnd="10dp"
            android:layout_marginStart="5dp"
            android:layout_marginTop="12.5dp"
            android:background="@drawable/button"
            android:layout_weight="0.5"
            android:text="FLASH ON" />

    </LinearLayout>

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/bpmPPG"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="serif"
            android:paddingBottom="10dp"
            android:layout_centerInParent="true"
            android:textAlignment="center"
            android:textSize="40sp" />
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="serif"
            android:textSize="16sp"
            android:layout_centerInParent="true"
            android:paddingTop="50dp"
            android:text="bpm"
            android:textAlignment="center"/>
        <ProgressBar
            android:id="@+id/spinner"
            android:layout_width="40dp"
```

```
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:paddingBottom="10dp"/>
        <ProgressBar
            android:id="@+id/loadBar"
            android:layout_marginTop="20dp"
            android:layout_width="160dp"
            android:layout_height="160dp"
            android:layout_gravity="center"
            android:progressDrawable="@drawable/progress"
            android:layout_centerInParent="true"
            style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>
    </RelativeLayout>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <com.androidplot.xy.XYPlot
            android:id="@+id/ppgPlotView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            ap:backgroundColor="@color/ap_transparent"
            ap:domainOriginLineColor="@color/ap_transparent"
            ap:graphBackgroundColor="@color/ap_transparent"
            ap:gridBackgroundColor="@color/ap_transparent"
            ap:graphMarginBottom="50dp"
            ap:graphMarginLeft="0dp"
            ap:graphMarginRight="0dp"
            ap:graphWidth="-25dp"
            ap:layout_constraintBottom_toBottomOf="parent"
            ap:layout_constraintEnd_toEndOf="parent"
            ap:layout_constraintStart_toStartOf="parent"
            ap:layout_constraintTop_toTopOf="parent"
            ap:legendAnchor="bottom_middle"
            ap:legendHeight="60dp"
            ap:legendTextSize="14sp"
            ap:legendHorizontalPositioning="absolute_from_center"
            ap:legendVisible="true"
            ap:lineLabels="none"
            ap:rangeLineColor="@color/ap_transparent"
            ap:rangeOriginLineColor="@color/ap_transparent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
</LinearLayout>
```

8.3.1.1 Custom Button XML

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient android:startColor="#eee"
        android:endColor="#ccc"
        android:angle="270" />
    <corners android:radius="5dp" />
</shape>
```

8.3.1.2 Progress Ring XML

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="ring"
    android:thickness="15dp"
    android:useLevel="true">
    <gradient android:angle="-90" android:startColor="#C02425"
android:endColor="#F0CB35" />
</shape>
```

8.4 Main Menu [Extensions]

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
      <item
          android:title="PPG analysis"
          app:showAsAction="collapseActionView">
          <menu>
              <item
                  app:showAsAction="never"
                  android:id="@+id/plotWindowPPG"
                  android:title="PPG plotter" />
              <item
                  app:showAsAction="never"
                  android:id="@+id/ppgHistory"
                  android:title="PPG history" />
              <item
                  app:showAsAction="never"
                  android:id="@+id/fpsAnalyser"
                  android:checked="true"
                  android:checkable="true"
                  android:title="FPS analyser"/>
              <item
                  app:showAsAction="never"
                  android:id="@+id/thresholdAnalyser"
                  android:checked="false"
                  android:checkable="true"
                  android:title="Plot threshold"/>
          </menu>
      </item>

</menu>
```

8.5 AttysECG Activity [Extensions]

8.5.1 Declarations

```
PPGPlotFragment ppgPlotFragment = null;
static String mCameraId;
static CameraManager mCameraManager;
private CameraCharacteristics characteristics;
private static final int REQUEST_CAMERA_PERMISSION = 1;

static Range<Integer>[] fpsRanges;
static Range<Integer> fpsFixed;
```

8.5.2 Simultaneous ECG Recording

```
if (PPGPlotFragment.isRecording) PPGPlotFragment.ECG_II_data.add(II);
```

8.5.3 OnCreate Methods

```
mCameraManager = (CameraManager) getSystemService(Context.CAMERA_SERVICE);

try {
    mCameraId = mCameraManager.getCameraIdList()[0];
    characteristics = mCameraManager.getCameraCharacteristics(mCameraId);
} catch (Exception e) {
    e.printStackTrace();
}

fpsRanges =
characteristics.get(CameraCharacteristics.CONTROL_AE_AVAILABLE_TARGET_FPS_RANGES);
```

8.5.4 Camera Permissions

```
void requestCameraPermission() {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
Manifest.permission.CAMERA)) {
        new AlertDialog.Builder(AttysECG.this)
                .setMessage("R string request permission")
                .setPositiveButton(android.R.string.ok, new
DialogInterface.OnClickListener() {
                    @Override
                    public void onClick(DialogInterface dialog, int which) {
                        ActivityCompat.requestPermissions(AttysECG.this,
                                new String[]{Manifest.permission.CAMERA},
                                REQUEST_CAMERA_PERMISSION);
                    }
                })
                .setNegativeButton(android.R.string.cancel,
                        new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which)
{
                                finish();

                            }
                        })
                .create();
```

```
    } else {
        ActivityCompat.requestPermissions(AttysECG.this, new
String[]{Manifest.permission.CAMERA},
                REQUEST_CAMERA_PERMISSION);
    }
}
```

8.5.5 Add PPG Window to Main Activity

```
private void openWindowPPG() {
    deletePlotWindow();
    ppgPlotFragment = new PPGPlotFragment();
    getSupportFragmentManager().beginTransaction()
            .add(R.id.fragment_plot_container,
                    ppgPlotFragment,
                    "ppgPlotFragment")
            .commitAllowingStateLoss();
    showPlotFragment();
}
```

8.5.6 Added MenuItem Methods

```
case R.id.plotWindowPPG:
    openWindowPPG();
    return true;

case R.id.ppgHistory:
    Intent historyActivity = new Intent(getApplicationContext(),
ppgHistory.class);
    startActivity(historyActivity);

case R.id.fpsAnalyser:
    if (item.isChecked()) {
        PPGPlotFragment.fpsAnalyserON = false;
        item.setChecked(false);
        prefs.edit().putBoolean("fpsAnalyser", false).apply();
        if (PPGPlotFragment.fps != null) {
            PPGPlotFragment.fps.setEnabled(false);
            PPGPlotFragment.fps.setTextColor(Color.TRANSPARENT);
        }
    } else {
        PPGPlotFragment.fpsAnalyserON = true;
        item.setChecked(true);
        prefs.edit().putBoolean("fpsAnalyser", true).apply();
        if (PPGPlotFragment.fps != null) {
            PPGPlotFragment.fps.setEnabled(true);
            PPGPlotFragment.fps.setTextColor(Color.parseColor("#555555"));
        }
    }
    return true;

case R.id.thresholdAnalyser:
    if (item.isChecked()) {
        PPGPlotFragment.thresholdAnalyserON = false;
        item.setChecked(false);
        prefs.edit().putBoolean("thresholdAnalyser", false).apply();
    } else {
```

```
            PPGPlotFragment.thresholdAnalyserON = true;
            item.setChecked(true);
            prefs.edit().putBoolean("thresholdAnalyser", true).apply();
        }
        return true;
```

8.5.7 Shared Preferences

```
if (prefs.getBoolean("fpsAnalyser", true)) {
    menuItemfpsAnalyser.setChecked(true);
    PPGPlotFragment.fpsAnalyserON = true;
} else {
    menuItemfpsAnalyser.setChecked(false);
    PPGPlotFragment.fpsAnalyserON = false;
}

if (prefs.getBoolean("thresholdAnalyser", true)) {
    menuItemthresholdAnalyser.setChecked(true);
    PPGPlotFragment.thresholdAnalyserON = true;
} else {
    menuItemthresholdAnalyser.setChecked(false);
    PPGPlotFragment.thresholdAnalyserON = false;
}
```

8.5.8 Camera FPS Preferences

```
int cameraFPS = Integer.parseInt(prefs.getString("fps", "24"));
fpsFixed = Range.create(cameraFPS, cameraFPS);
```

8.6 preferences.xml [Extensions]

```
<ListPreference
    android:defaultValue="24"
    android:entries="@array/fpsOptions"
    android:entryValues="@array/fpsValues"
    android:key="fps"
    android:summary="Frames per second for video (Higher values require more
processing power)"
    android:title="Camera FPS" />
```

8.6.1 array.xml [Extensions]

```
<string-array name="fpsOptions">
    <item>20fps</item>
    <item>24fps</item>
    <item>30fps</item>
</string-array>
<string-array name="fpsValues">
    <item>20</item>
    <item>24</item>
    <item>30</item>
</string-array>
```