Novel Human Computer Interaction Principles for Cardiac Feedback Using Google Glass and Android Wear

Bachelor Thesis in Medical Engineering

submitted
by

Robert Richer

born 04.06.1994 in Bamberg

Written at

Lehrstuhl für Mustererkennung (Informatik 5)
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg.

Advisor: Prof. Dr. Bjoern Eskofier, Prof. Dr. Bernhard Hensel, PD Dr. med. Christian Stumpf

Started: 15.08.2014

Finished: 15.02.2015

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien des Lehrstuhls für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Erlangen, den 15th February 2015

Übersicht

Zu den Hauptursachen eines frühzeitigen Todes gehören Erkrankungen des Herz-Kreislauf-Systems, vor allem aufgrund demographischen Wandels, weswegen die Nachfrage nach tragbaren Systemen zur Gesundheitswachung stetig ansteigt. Mehrere Forschergruppen haben sich bereits mit der Entwicklung mobiler Lösungen zur Überwachung von Vitalparametern befasst. Allerdings besteht immer noch ein Forschungsbedarf in der Entwicklung eines Komplettsystems, welches das komplette Spektrum von der Datenaufnahme, über die -verarbeitung, bis hin zur Visualisierung, abdeckt. Aus diesem Grunde wurde im Rahmen dieser Arbeit eine Lösung entwickelt, die neue Interaktionskonzepte mit einer Smartphone-Applikation für kardiales Feedback verbindet. Dabei wird ein neuentwickelter EKG-Sensor (genannt *BLE-ECG Stamp*) als Aufnahmegerät benutzt, der das Signal über *Bluetooth Low Energy* an ein mobiles Endgerät sendet. Des weiteren bietet die Applikation dem Benutzer verschiedene Möglichkeiten, seinen aktuellen Herzstatus abzufragen sowie ihn über einen längeren Zeitintervall hinweg zu überwachen, ohne dass der Benutzer in seinen alltäglichen Tätigkeiten unterbrochen wird. Deswegen hat sich diese Arbeit nicht nur auf die Entwicklung einer reinen Smartphone-Applikation konzentriert, sondern integriert auch tragbare Geräte (sogenannte *wearables*) wie *Google Glass* (eine Brille mit einem integrierten Display, das Informationen in das Sichtfeld des Benutzers einblenden kann) und *Android Wear*-basierten Smartwatches (Armbanduhren mit integriertem Display und Sensoren). Dieses System wurde im Rahmen zweier Studien evaluiert: Zum einen wurde die Qualität des neuen Sensors validiert. Als Referenzsystem diente ein tragbarer EKG-Sensor von *Shimmer*, der bereits in mehreren wissenschaftlichen Arbeiten zum Einsatz kam. Das Auswertung der Studie ergibt einen mittleren absoluten Fehler des *BLE-ECG Stamps* von 4.83 bpm (Schläge pro Minute) im Vergleich zum *Shimmer*, bei einer Standardabweichung von 4.46 bpm. Außerdem wurde das Gesamtsystem, welches in dieser Arbeit verwirklicht wurde, in einer Studie getestet und dessen Alltagstauglichkeit mittels eines Fragebogens ausgewertet. Die Bedienbarkeit der Applikation in Kombination mit einer *Google Glass* erreichte eine durchschnittliche Bewertung von 2.8/5 Punkten, die Kombination mit einer Smartwatch eine Bewertung von 4.2/5 Punkten. Im Vergleich zur *Google Glass* erzielten Smartwatches über alle Fragen hinweg bessere Ergebnisse. Beide Studien kamen zu dem Schluss, dass das entwickelte und getestete Gesamtsystem dem Benutzer eine zuverlässige Plattform für die Überwachung seines Herzstatus bietet, sowohl auf der hardwaretechnischen Seite des Sensors, als auch seitens der Bedienbarkeit der entwickelten Applikation.

Abstract

One of the main reasons for early fatality are diseases of the cardiovascular system. This problem is magnified by the current changes in the demographic structure which makes a wearable health monitoring of individuals more and more important. Several research groups have developed mobile solutions for providing vital sign monitoring. Nevertheless a lack of research still remains in the development of a system that addresses the whole pipeline from data acquisition over data processing, with a special respect to proper data visualization and interaction concept between the user and the system. Therefore the present work presents a solution that offers novel human computer interaction principles for cardiac feedback. It is based on an application for smartphones or tables running on an *Android* operating system and uses a novel wearable ECG sensor as recording device which transmits the acquired data via the *Bluetooth Low Energy* communication standard. Furthermore the application offers various features for allowing the user to checking and monitoring his heart status in a unobtrusive and comfortable way. The range of functions was enhanced by including novel wearable technology like *Google Glass* (a wearable with an included optical head-mounted display) and smartwatches running on the *Android Wear* operating system. This setup was evaluated in two studies: The first study validated the signal quality of the sensor compared to another wearable ECG sensor by *Shimmer*. It has already been used in several studies and hence was considered as gold standard. The results showed that the heart rate of the *BLE-ECG Stamp* has an absolute error of 4.83 bpm compared to the *Shimmer* with a standard deviation of 4.46 bpm. Furthermore, the developed application was evaluated in a questionnaire with respect to the daily usability. The usability of the *DailyHeart* application in combination with *Google Glass* was rated 2.8/5, whereas the combination with a smartwatch was rated 4.2/5. Compared to *Glass*, smartwatches running on *Android Wear* achieved higher ratings throughout all questions.

Both evaluations proved that the newly developed system provides a solid platform for monitoring the user's cardiac activities in daily life, on the sensor side as well as on the user interface side.

Contents

Chapter 1

Introduction

Designing wearable measurement units, especially for measuring physiological signals of the human body, has been a growing research interest in the last decade, creating lightweight, miniaturized and affordable systems with possible applications in medicine, health-care, sports and security [Yil10]. These solutions allow a ubiquitous and pervasive monitoring of vital signs in various environments without any restrictions in activity or modification in behavior. According to Yang [Yan14], the aim of ***Body Sensor Networks (BSN)***[1] is to provide a personalized monitoring platform that is "pervasive, intelligent and context-aware, yet 'invisible'". Some of main challenges that have to be dealt with are:

- intelligent sensor design,
- integration of ***micro electro mechanical systems (MEMS)***,
- power source miniaturization,
- low power wireless transmission, and
- integration with smart diagnostic and therapeutic systems.

With the changes in the demographic structure, characterized by a decrease in reproduction rates, a decrease in mortality and an increase of life expectancy [Car06], and resulting in an increase of chronic diseases, a need to monitor the individual's health status for preventing fatal disorders becomes more and more important.

[1] Definitions for abbreviations and terms that are highlighted can be found in the Glossary

Wearable health-care technology is a promising way to increase the quality of life for chronic disease patients and elderly people, as well as healthy individuals in any environment. The environment has an impact on the patient's physiology which is not negligible. The best example for this impact is the *"white coat syndrome"*, which is known of the patients' blood pressure being much higher in a clinical as it is in a normal environment [DH03]. For patients with chronic diseases, the key is to prevent unnecessary hospitalizations and enable out-patient care by monitoring their vital signs in a daily environment, being at home, at work, during physical activity or at rest.

The function of the cardiovascular system can be monitored using different types of physiological signals. The most common known modality is the **Electrocardiogram (ECG)**. It is a interpretation of the electrical activity of the heart cells over a period of time and is measured by electrodes attached to the skin surface of the patient. Apart from calculating the heart rate from the ECG, it is furthermore used as indication for several abnormalities, including *coronary heart diseases*, *heart arrhythmia* (an irregular heart beat pattern), *cardiomyopathy* (a lack of contraction ability within the heart muscles), *congenital heart defects* (birth defects in the heart), *heart valve diseases*, or *pericarditis* (an inflammation of the sac that surrounds the heart) [Sut14].

Another physiological signal to monitor the cardiac function is the **Photoplethysmogram (PPG)**. It measures the change in blood volume in the skin caused by the cardiac cycle. The PPG signal is gained by illuminating the skin with a light-emitting-diode (LED) and detecting the changes in light absorption being either transmitted or reflected. PPG can be used to determine the individual's heart rate as well as measuring the saturation of peripheral oxygen (also known as SpO_2) [All07].

Several papers have been published that focus on the development of mobile applications measuring the user's vital signs for different purposes in health-care and sports. In 2010, Suh et al. [Suh10] presented a system for an automated vital sign monitoring for congestive heart failure patients. The system incorporates wireless data transmission of several devices – like weight scales, blood pressure monitors, blood glucose monitors, or personal activity monitors – to web databases, as well as mobile and web applications with functions for checking the patients' vital signs. Gradl et al. [Gra12] developed an application called "Hearty" for real-time ECG monitoring with a *Shimmer*™ (Dublin, Ireland) sensor node. They furthermore provided an algorithm for real-time arrhythmia detection by segmenting the ECG's QRS complexes and automatically classifying the heart beat into normal and abnormal beats.

Because cardiac monitoring is a very important topic, several inventions related to the present work have been patented:

- The three US patents **US8509882** (Appendix A.1), **US20130338460** (App. A.2), and **US20140100432** (App. A.3) all present wearable devices for a continuous cardiac monitoring by recording the user's ECG signal. The last patent even combines the ECG recording with a simultaneous PPG recording. The acquired data might be processed on the wearable and afterwards transmitted wirelessly to a computer or a smartphone.

- The US patents **US8233972** (App. A.4), **US7120485** (App. A.5), and the European patent **EP2676604** (App. A.6) have a claim on the development of methods that are able to detect and classify cardiac arrhythmia.

- US patent **US6102856** (App. A.7) provided a wearable vital sign monitoring system that does not only consist of an ECG sensor but furthermore uses a respiration sensor, a temperature sensor, and an O_2 sensor in order to measure several vital sign parameters and determine a deviation from the wearer's normal condition.

- US patent **US20140273858** (App. A.8) presents a biometric monitoring device that sends physiological data utilizing two different *Bluetooth* communication interfaces in a common device.

Other examples of applications for detecting cardiovascular abnormalities have been proposed by Oresko et al. [Ore10] or Makki et al. [Mak14]. Abtahi et al. [Abt14] developed a ***heart rate variability (HRV)*** biofeedback application for *Android*^{TM}-based smartphones by recording the heart rate via ECG and the respiration rate via bioimpedance measurement. An application that uses ECG data in a context of sports has been presented by Richer et al. [Ric14]. It is an application that is specialized for biking and calculates the heart rate out of the ECG in combination with the cadence obtained from the ***electromyogram (EMG)***.

The focus of this thesis lies on the implementation and evaluation of an application that is able to monitor the user's cardiac functions by recording and processing its ECG in daily life. Furthermore, the application is also capable for simulating pre-recorded data from a *Bluetooth® Low Energy* sensor or from the MIT-BIH Arrhythmia database [Mar97]. Therefore, it is necessary to develop an application that includes a proper visualization of the acquired information, as well as keeping interaction between the user and the mobile device as simple and intuitive as possible.

With the rapid development of ***wearable computers*** (also known as ***wearables***), especially in the past years, it is possible to include these techniques for displaying information about the user's vital signs as well as interacting between wearable and mobile phone. A wearable that can be used for this purpose (and is used for the present work) is *Google Glass* (Google Inc., Mountain View, CA, USA), which offers an optical head-mounted display to interact with the user. Hernandez et al. have used Google Glass and its internal sensors to estimate physiological parameters such as heart rate and respiratory rate [Her14a]. They furthermore presented an application to measure and visualize daily life emotions [Her14b] with Glass. Another type of wearable devices that are used in this thesis is the technology of smart watches, which are computerized wristwatches with functionalities that are beyond traditional timekeeping.

The structure of this thesis is organized as follows: Chapter 2 describes the methods used for the development of an application that provides cardiac monitoring in daily life with the incorporation of novel wearable technology. Beginning with basic information about the ECG signal, the sensor device used for data acquisition is described afterwards. It is followed by a section about the algorithm used for the ECG analysis and heart beat classification. The chapter is concluded by presenting the implementation of the application on the mobile phone, Google Glass, and smartwatches and by the evaluation of the sensor device as well as the the application itself, with special focus on daily life usability and the combination with wearables. Chapter 3 presents the results of the work and the evaluation, which is followed by the discussion in Chapter 4. Chapter 5 concludes the work and provides an outlook for possible future work.

Chapter 2

Methods

The present work shows the development of an Android application that incorporates novel human-computer-interaction concepts under the aspect of cardiac monitoring in daily life. Therefore, the whole pipeline from data acquisition over data processing to data visualization has to be addressed in order to provide a solid, reliable system. A detailed description of the pipeline and the methods used to implement this work are presented in the following chapter.

2.1 Data Generation – ECG Basics

As mentioned in the Introduction, the cardiovascular activity can be measured by different modalities. The present work uses the **electrocardiogram (ECG)** since it is a very common and reliable technique for cardiac monitoring. A typical ECG recording of a single heart beat is shown in Figure 2.1. As the wave of myocardial depolarization that is initiated in the sinoatrial node (SA node) travels through the heart, the signal can be divided in the following phases:

- **P wave**: Slow depolarization of the atrial cells
  Duration: 60 - 80 ms
- **PQ segment**: Interval between the end of the P wave and the beginning of the QRS complex; represents the time the wave travels from the atrioventricular (AV) node through the Purkinje fibers over the bundle branches towards the ventricles
  Duration: 60 - 80 ms
- **QRS complex**: Rapid depolarization of the ventricles
  Duration: 80 - 100 ms

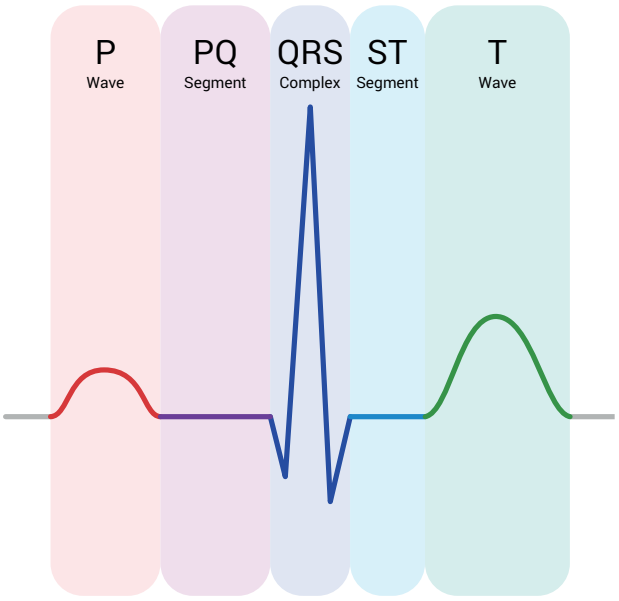


Figure 2.1: ECG complex. Schematic representation of a normal ECG complex

- **ST segment**: Interval between the end of the QRS complex and the beginning of the T wave; represents the time the ventricular cells are depolarized
  Duration: 100 - 120 ms
- **T wave**: Slow repolarization and relaxation of the ventricles
  Duration: 120 - 160 ms

Various diseases can be diagnosed by observing pathological changes in the ECG signal. They manifest in abnormal waveforms and arrhythmic pace. The algorithm used in this work, proposed by Gradl et al. [Gra12], is able to classify and differentiate between the following abnormalities:

- **Premature ventricular contraction (PVC)**: Heart beats that are initiated by the Purkinje fibres rather than the SA node (the normal initiator). The ventricles contract before the atria have been filled with blood, leading to inefficient blood circulation. While single PVCs can also occur in healthy patients and usually do not pose a danger, longer periods of PVCs can be caused by myocardial infarction, myocarditis, or thyroid problems (2.2a) [Sut14].
- **Atrial premature contraction (APC)**: Depolarization of the atria which is not triggered by the SA node. APC beats are characterized by an abnormal shape of the P wave, but not considered malign (2.2b) [Sur08].
- **Bundle branch block (BB block)**: A defect in the wave conduction due to interruptions in one or more bundle branches. Hence, ventricular depolarization has to

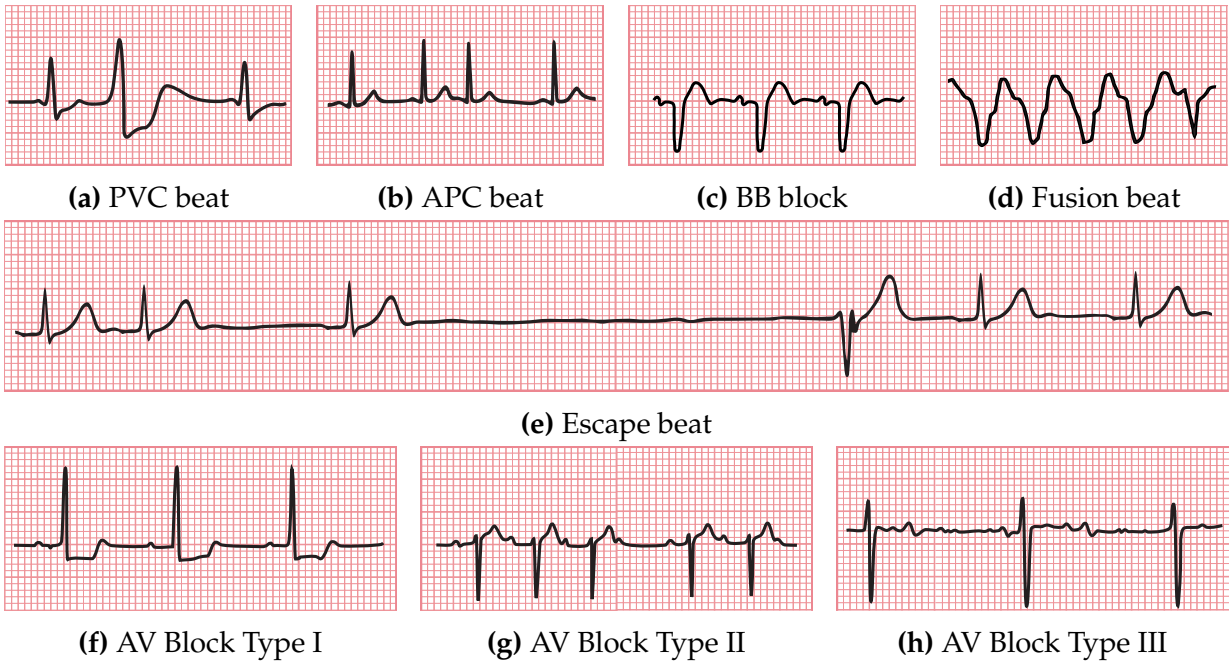


Figure 2.2: Cardiac abnormalities in an ECG. The algorithm used in this work is capable of analyzing the ECG and classifying the heart beats into normal and abnormal; *(a) PVC beats, (b) APC beats, (c) BB blocks, (d) Fusion beats, (e) Escape beats, (f) AV Block Type I, (g) AV block Type II, (h) AV block Type III*

find alternative pathways, resulting in a longer QRS complex. BB blocks can be a sign for myocardial infarction, or the result of injuries during cardiac surgery (2.2c) [Sur08].

- **Fusion beats**: A region of the heart is stimulated by regular triggering from the SA node and abnormal sources at the same time. The result is a different waveform of the QRS complex (2.2d) [Sut14].
- **Escape beats**: A self-generated depolarization in the ventricles of the heart, causing a contraction that is not initiated by the SA node. Depolarization is triggered by the ventricles after a long pause of contraction in order to prevent cardiac arrest. Escape beats in the ECG indicate a failure in the heart's electrical conduction system (2.2e) [Sut14].
- **Atrioventricular block (AV block)**: An abnormality in the electrical conduction between the atria and the ventricles. It is usually caused by a delay or interruption of the stimulating AV node, so that the triggering of atria and ventricles is independent. AV blocks might indicate a myocarditis or various infections like endocarditis or Lyme borreliosis (2.2f, 2.2g, 2.2h) [Sur08].

- **Bradycardia**: A resting heart rate of under 60 beats per minute (bpm) over a longer period of time. Pathological bradycardia might be the result of various reasons, such as hypothyroidism, poisoning, or infections, and can lead to dizziness, lightheadness, or syncope [Sut14].
- **Tachycardia**: A resting heart rate of more than 100 bpm over a longer period of time. Pathological tachycardia can be caused by fever, hyperthyroidism, anemia, or coronary/myocardial infarcts [Sut14].

2.2 Data Acquisition – "BLE-ECG Stamp"

Traditionally, continuous ECG monitoring is performed using a ***Holter monitor***. The setup seen in Figure 2.3 already shows the major constraint of this device: The bulky central unit interrupts the patient in its daily routine and thus is not very feasible for an unobtrusive, continuous monitoring. Miniaturizing the concept of Holter monitoring is one of the key challenges in the development of Body Sensor Networks for cardiac monitoring. A promising approach has been accomplished in the lab of the *Max Schaldach endowed professorship for Biomedical Engineering* at the Friedrich-Alexander-Universität Erlangen-Nürnberg, led by Prof. Dr. Bernhard Hensel. The bachelor's thesis *Development of a Platform for Wearable Biosensor Networks with Bluetooth®Low Energy* by Tim Maiwald [Mai14] written at Prof. Hensel's lab presents the results of this project. The platform was used to acquire the user's ECG signal unobtrusively. Its components will be outlined in this section.

2.2.1 Setup of Sensor Components

The sensor system (further referred to as *BLE-ECG Stamp*) basically consists of two main components, as shown schematically in Figure 2.4: the **Sensor/Amplifier Unit**, and the **Controller/Transmission Unit**.

The Sensor/Amplifier Unit measures a physical quantity X and converts it to an amplified analog output voltage U. In case of electrocardiography, the measured physical quantity is the electrical change on the skin, caused by the depolarization of the heart muscles during every heart beat.

The Controller/Transmission Unit consists of a microcontroller for data sampling and a ***System-on-a-chip (SoC)*** for data transmission. The microcontroller used in this sensor is a MSP430-FR5969 (Texas Instruments, Dallas, TX, USA), an *ultra-low-power* processor

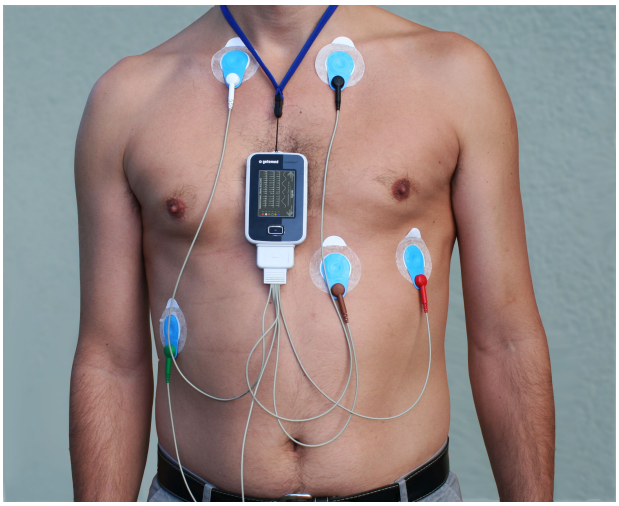


Figure 2.3: Setup of a Holter monitor. It consists of a central unit and several electrodes connected by wires
Source: Wikimedia Commons; Author: *Misscurry*, licensed under *CC BY 2.0* [Mis]

with a maximum frequency of 16 MHz and an ***FRAM*** memory with a capacity of 64 kB. Due to energy saving aspects, the clock rate of the microcontroller for sampling the ECG was set to 1 MHz. It is also possible to increase the clock rate and perform more complex computations on the MSP430, like the calculation of the current heart rate. The SoC is a CC2541 (Texas Instruments, Dalls, TX, USA), which is a integrated solution for Bluetooth Low Energy applications, optimized for maximal performance. It combines a high power Bluetooth antenna with an improved 8051 microcontroller.

2.2.2 Sampling of the ECG

The MSP430-FR5969 microcontroller uses an ***Analog-to-digital-converter (ADC)*** to sample a physical quantity to a digital quantization with a resolution of 12-bit at a specific sampling frequency. This frequency has to be chosen carefully according to the Nyquist-Shannon sampling theorem [Sha49]:

$$f_s \geq 2 f_{max}. \tag{2.1}$$

According to this theorem, avoiding aliasing artifacts requires the sampling frequency f_s to be equal or greater than twice the maximum frequency f_{max} in the signal. Regarding the fact that the ECG signal is not a signal with band-pass characteristic and also contains components in higher frequency ranges, a trade-off between the accuracy of the sampled

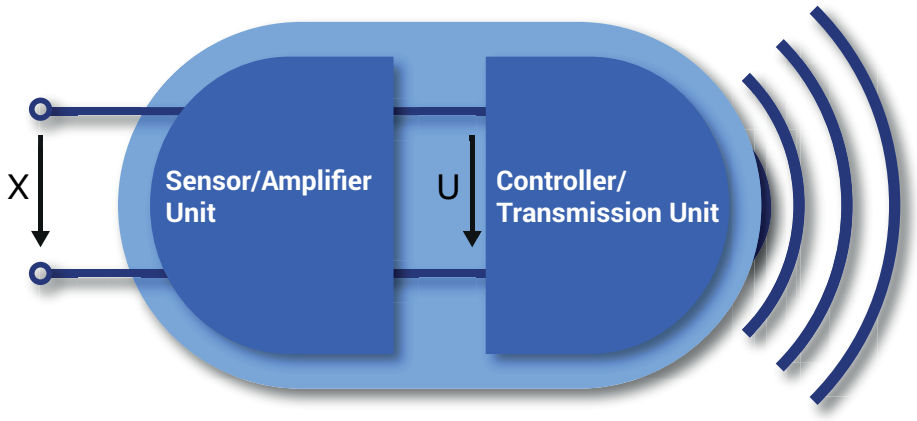


Figure 2.4: Setup of the BLE-ECG Stamp. It consists of the ***Sensor/Amplifier Unit*** (left), which measures the data, and the ***Controller/Transmission Unit*** (right), which transmits the data via *Bluetooth Low Energy*

signal and the amount of acquired data has to be found. The chosen sampling frequency of 250 Hz was chosen according to the results of Pizzuti et al. [Piz85]: They observed that a sampling frequency of 125 Hz shows significant reductions in the signal quality, whereas a sampling frequency between 250 Hz and 500 Hz does not show any significant changes in the signal quality. The analog output voltage of the sensor $y(t)$ is generated by the convolution of the raw ECG signal $x(t)$ tapped by the electrodes with the internal impulse response of the sensor $h(t)$:

$$y(t) = x(t) * h(t) = \int_0^t x(\tau)h(t-\tau)d\tau. \tag{2.2}$$

2.2.3 Lead Positioning

The *BLE-ECG Stamp* records a 1-channel-ECG with 3 electrodes. By using a different ECG amplifier, the number of channels can be increased so that even 12-channel-ECG recordings are possible. Lead II according to Einthoven's triangle [Con03] was used for data acquisition. The setup and the lead positioning can be seen in Figure 2.5. The negative electrode is placed on the sternum, whereas the positive electrode is placed on the left hand costal arch. The third electrode is used for the *Driven Right Leg (DRL) Circuit*. It is a circuit that is often added to physiological signal amplifiers in order to eliminate electromagnetic interferences, especially noise from surrounding electrical power lines at a frequency of 50/60 Hz [Win83]. This electrode can be placed either on the leg or on the breast.

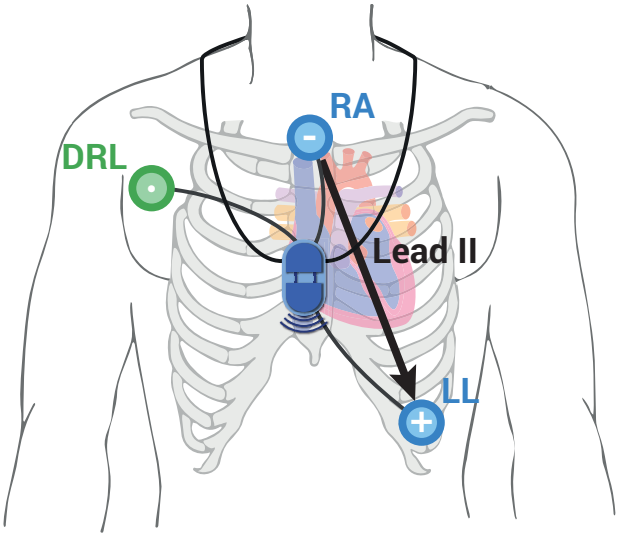


Figure 2.5: Lead Positioning of the BLE-ECG Stamp. Schematic drawing of the sensor placement and the lead positioning; ***RA:*** *Right Arm*, ***LL:*** *Left Leg*, ***DRL:*** *Driven Right Leg*

2.2.4 Data Transmission via Bluetooth® LE

The CC2541 SoC receives the sampled signal values from the MSP430 microcontroller over a ***serial port*** and transmits the data via *Bluetooth Low Energy*. Bluetooth® is a wireless technology standard developed by the Bluetooth Special Interest Group (SIG) for exchanging data over short distances at a frequency of approximately 2.4 GHz. It allows the connection of several devices, which makes it especially suitable for mobile devices and the setup of a Body Sensor Network.

With the release of Bluetooth version 4.0 in 2010, Bluetooth Low Energy, also known as *Bluetooth LE*, *BLE*, or *Bluetooth Smart*, was unveiled. *Bluetooth LE* features a completely new protocol stack. This enables a rapid build-up of links between devices, which results in a significantly reduced power consumption compared to standard Bluetooth or other wireless standards like *ZigBee* or *ANT* [Gom12], [Dem13]. One *master device* (also referred to as Bluetooth Client) can establish connections to multiple *slave devices* simultaneously. In the setup of the present work, the *BLE-ECG Stamp* plays the role of a slave device, whereas the smartphone or tablet acts as the master device.

The *Bluetooth LE* architecture consists of three layers, divided into several sublayers, as shown in Figure 2.6. The lowest layer is the **Controller Layer**, which represents the interface to the environment, followed by the **Host Layer**, which takes care of handling the processes and functions of BLE. Among others, the Host Layer contains the *Generic Attribute Profile (GATT)* and the *Generic Access Profile (GAP)* sublayers. The parameters of

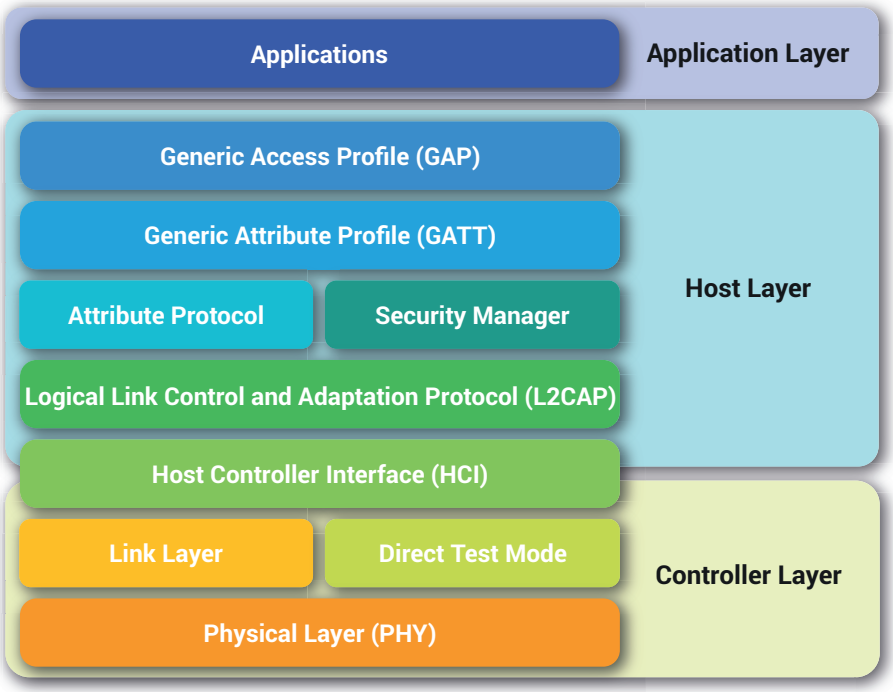


Figure 2.6: BLE Protocol Stack. It consists of three layers, each of them divided into different sublayers: the *Controller Layer*, the *Host Layer*, and the *Application Layer*

GATT and GAP are passed to the **Application Layer**. They are specific for one application, because they define the necessary *Services*, *Characteristics*, and *Descriptors*.

A Service describes the behavior of the device. Each device can handle different Services, which are identified by ***universally unique identifiers (UUID)***. The Services on the *BLE-ECG Stamp* (the slave device) can be activated and deactivated by the smartphone (the master device). The values acquired by the sensor are stored in Characteristics defined by the Service and transmitted to the client. Similar to Services, Characteristics are also identified by a UUID in order be able to uniquely assign them on the client side. Descriptors are used to describe a specific Characteristic and can be read as well as written by the client.

The *Bluetooth Application* on the BLE-ECG stamp was developed for the transmission of values acquired by the sensor to the connected client that supports *Bluetooth LE*. In the present work, the client is a smartphone or tablet running on Android software version 4.3 (***API*** level 18) or higher. The Application consists of two Bluetooth Services: The *EcgService* for streaming the sampled raw ECG signal, and the *HeartrateService*. The latter Service reduces the data rate clearly by performing a heart rate calculation on the sensor

side and only transmitting the current heart rate to the client. This leads to a significantly lower power consumption compared to streaming the raw ECG signal by the *EcgService*. Because the smartphone application developed in this thesis was designed to use the raw ECG signal for performing further processing on the mobile device, the *HeartrateService* was not feasible for that specific purpose. However, by buffering the raw ECG values and sending packages consisting of 10 values, the Bluetooth transmission under real-time constraints has been designed more energy-efficiently.

2.3 Data Processing and Classification – "Hearty Algorithm"

In order to calculate the user's heart rate and classify the heart beats, the received raw ECG data needs to be processed using efficient real-time algorithms that are also suitable for the processing on mobile devices.

The algorithm used in the present work was presented by Gradl et al. in their work *Real-time ECG Monitoring and Arrhythmia Detection using Android-based Mobile Devices* [Gra12]. This algorithm is able to perform a real-time detection of QRS complexes in an ECG signal, followed by an automated heart beat classification into normal or abnormal heart beats, which extends well-known analysis methods [Kra07]. The pipeline is visualized in Figure 2.7 and consists of four stages: QRS detection (2.3.1); template formation and adaption (2.3.2); feature extraction (2.3.3) and beat classification (2.3.4).

2.3.1 QRS Detection

First, the raw ECG signal is processed with an algorithm that was proposed by Pan & Tompinks [Pan85] in 1985. It uses a pipeline of digital filters for noise reduction and QRS detection.

Bandpass filter The bandpass filter is a recursive filter with integer coefficients that reduces the influences of muscle artifacts, 50 Hz interferences from the electrical power lines, baseline wander, and T-wave interference. It is cascaded into a lowpass and a

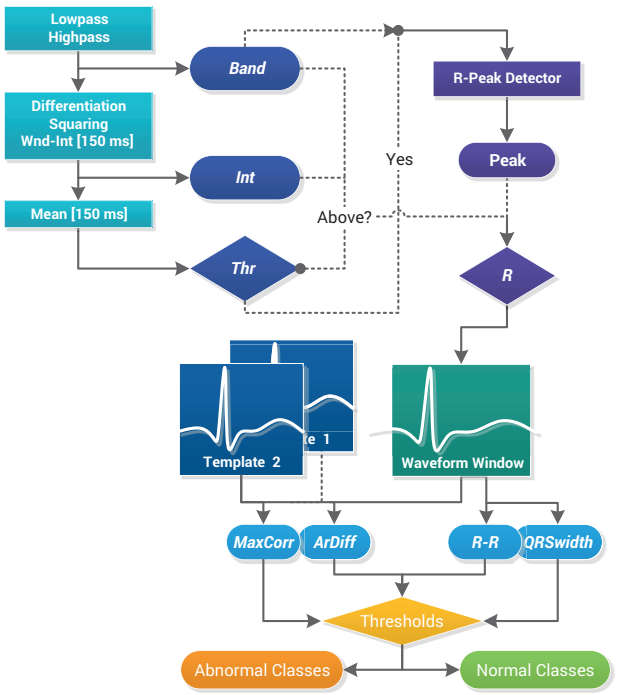


Figure 2.7: Algorithm pipeline. Overview of the pipeline that is used for QRS detection and heart beat classification [Gra12] (modified with kind permission of the author)

highpass filter. The transfer function of the *lowpass filter* is

$$H(z) = \frac{1}{32}\frac{(1-z^{-6})^2}{(1-z^{-1})^2}, \tag{2.3}$$

with the following difference equation:

$$y_(n) = 2y(n-1) - y(n-2) + \frac{1}{32}\left[x(n) - 2x(n-6) + x(n-12)\right]. \tag{2.4}$$

The lowpass filter has a cutoff frequency of 11 Hz with a delay of 6 samples. The transfer function of the *highpass filter* is

$$H(z) = \frac{1}{32}\frac{z^{-32} - 32z^{-17} + 32z^{-16} - 1}{1 - z^{-1}}, \tag{2.5}$$

with the following difference equation:

$$y(n) = y(n-1) - \frac{1}{32}x(n) + x(n-16) - x(n-17) + \frac{1}{32}x(n-32). \tag{2.6}$$

The designed highpass filter has a cutoff frequency of 5 Hz with a delay of 16 samples. The frequency response of the resulting bandpass filter is plotted in Figure 2.8a.

Derivation The bandpassed signal is afterwards processed using a five-point-derivation filter in order to suppress the low frequency components of the P and T wave and to amplify the high frequency parts of the QRS complex due to its strong slope. The transfer function of the derivation is

$$H(z) = \frac{1}{8}\left[2z + z^{-1} - z^{-3} - 2z^{-4}\right], \tag{2.7}$$

with the following difference equation:

$$y(n) = \frac{1}{8}\left[2x(n) + x(n-1) - x(n-3) - 2x(n-4)\right]. \tag{2.8}$$

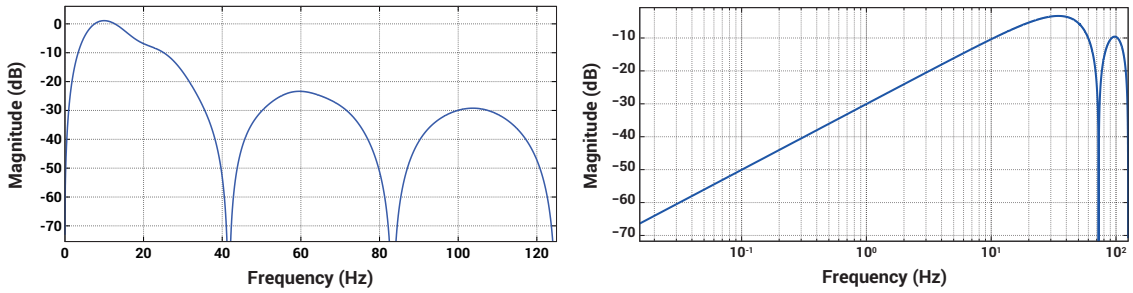
As seen in Figure 2.8b, the frequency response of this derivative is almost linear until 30 Hz, making it an ideal differentiator in this interval, with only two samples delay.

Squaring It ensures all sample values are positive and performs a nonlinear amplification of the QRS complex peak with a simultaneous suppression of the P and T wave peak:

$$y(n) = \left[x(n)\right]^2. \tag{2.9}$$

Moving window integration The moving window integrator smoothes the output of the derivation, because multiple peaks can occur in the interval of one QRS complex. The window length has to be approximately as long as the widest possible QRS width, and has been set to 150 ms by [Pan85] empirically.

Peak detection The QRS complexes are isolated using a threshold (denoted as *Thr*). It is computed by applying a moving-average filter with a window size of 150 ms on the output of the moving window integrator (denoted as *Int*). If either the output of the bandpass (denoted as *Band*) or *Int* reaches the threshold, the R deflection is searched by using a 3-point peak detector on the *Band* output. Each peak candidate is again compared to *Thr* in order to assure it is a valid R deflection.



(a) Filter response of the bandpass filter (linear-log scale)

(b) Filter respone of the derivation filter (log-log scale)

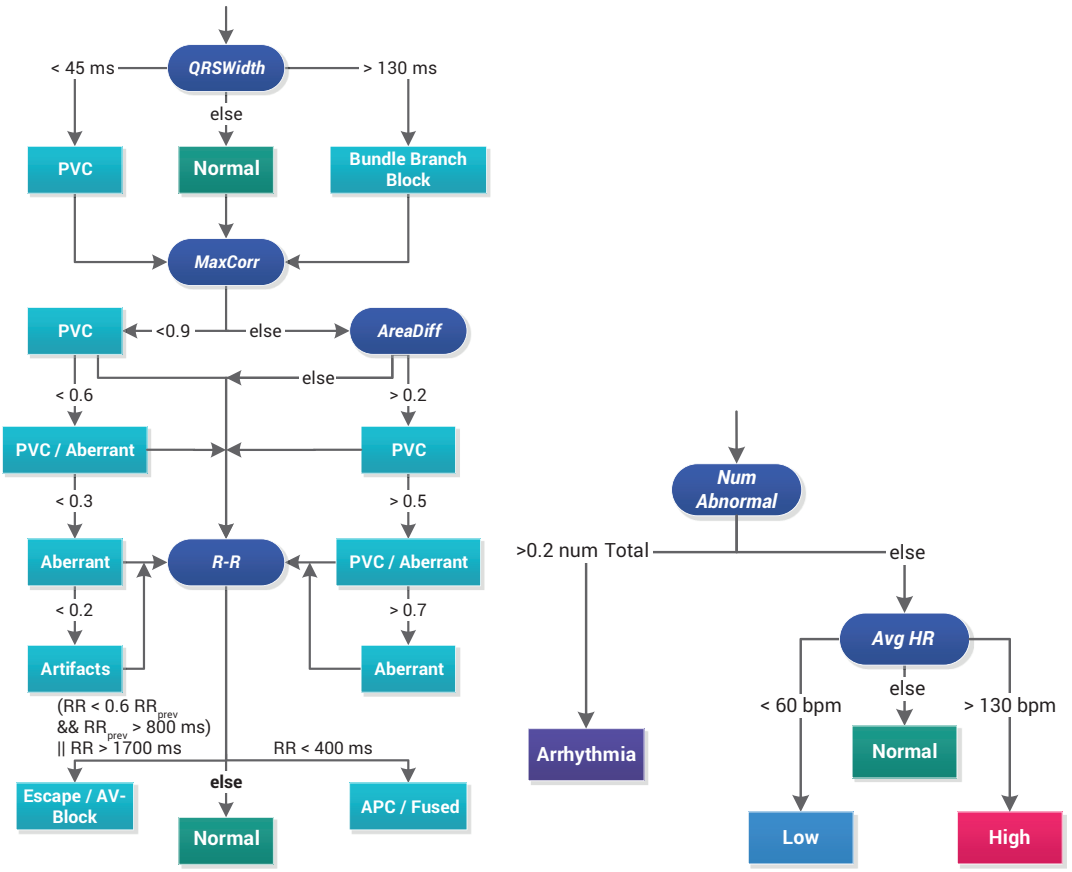
Figure 2.8: Bandpass and derivation magnitude responses of the Pan-Tompkins-Algorithm. The bandpass filter is the first step in the processing pipeline of the QRS complex detection algorithm, followed by the derivation filter

2.3.2 Template Formation and Adaptation

For enabling an automatic classification of heart beats, an adequate feature computation using two QRS complex templates has to be implemented. Gradl et al. [Gra12] proposed a method in which QRS templates were automatically found in the ECG signal and adapted over time. The first templates are generated by finding the first six valid R peaks. A window of 400 ms was centered around each R peak in order to extract the QRS complexes. For determining the two QRS complexes that are most suitable as templates, two criteria have been used: the smallest difference of the individual waveform area to the average waveform area of the six QRS complexes (1), and a cross-correlation between the QRS complexes of more than 0.95 (2). The template candidates were sorted according to (1) in ascending order, meaning that QRS complexes with an area lower than the average area were considered first. The first two consecutive candidates in this ranking that fulfilled (2) were assigned to the template slots. If such a pair could not be determined, the first two QRS complexes in the ranking were assigned. In order to adapt the templates over time, they were updated during the measurement by assigning the current heart beat classified as normal to the template slot with the higher correlation between the current beat and both template beats.

2.3.3 Feature Extraction

The features used for the heart beat classification were again extracted from 400 ms windows centered on every valid R peak:



(a) Decision tree for heart beat classification [Gra12] (modified with kind permission of the author).

(b) Decision tree for heart status determination

Figure 2.9: Decision trees for ECG classification

1. **AreaDiff:** The difference in absolute area between the templates and the current heart beat using normalized waveform areas.

2. **MaxCorr:** Maximal Pearson correlation between the templates and the current heart beat.

3. **QrsWidth:** The width of the detected QRS complex, obtained from the *Int* output of the *Pan-Tompkins-Algorithm*.

4. **RR:** The R-R interval between the last two detected QRS complexes, also obtained from the *Pan-Tompkins-Algorithm*.

2.3.4 Beat Classification

The different heart beats were classified according to the decision tree in Figure 2.9a. Specific characteristics in the waveform of the heart beats were analyzed as well as changes in the pace or rhythm of succeeding beats. Abnormalities in the waveform lead to *PVC beats*, *BB Block beats*, *Escape beats*, and *APC beats*, whereas abnormalities in rhythm/pace manifest in *Fusion beats*, *AV blocks*, *Tachycardia*, or *Bradycardia*. A characterization of these cardiac abnormalities is provided in Section 2.1.

The present work uses the results from the heart beat classification algorithm to classify the user's cardiac status into four classes:

- **OK**: No abnormalities in the ECG
- **Low**: The ECG shows potential signs of bradycardia
- **High**: The ECG shows potential signs of tachycardia
- **Arrhythmia**: The ECG shows potential signs of arrhythmia

Figure 2.9b shows the decision tree to determine the current heart status. It has to be mentioned that the classification of the heart status has not been clinically validated and was determined empirically for this work.

2.4 Data Visualization – "DailyHeart"

The ECG data acquired from the *BLE-ECG Stamp* was transmitted to an Android-based smartphone. The result of this work is an application called *DailyHeart* that is capable of receiving, processing and visualizing this data. Before the implementation of the application, the following design considerations were defined:

- **Measuring** the user's current heart rate (*CurrentHR* mode)
- **Analyzing** the user's current ECG (*AnalyzeECG* mode)
- **Monitoring** the user's ECG for a period of time (*DailyMonitor* mode)
- **Simulating** previously recorded ECG data from *DailyHeart* and the *MIT-BIH Arrhythmia Database* (*DailySimulate* mode)
- **Displaying** the results in an adequate way

- **Storing** the results in a database
- **Uploading** the results to *Google Fit*
- **Interacting** with *Google Glass* and smartwatches running on *Android Wear*

2.4.1 Android Basics

Android™ is a mobile operating system by Google Inc. (Mountain View, CA, USA) primarily developed for mobile devices, such as smartphones and tablets. It was originally created by a company called *Android Inc.* that was founded in 2003 and acquired by Google in 2005. The main intention of Android was providing a mobile device platform powered by a Linux ***kernel***. In 2007, the *Open Handset Alliance* was founded. It is a consortium of technology companies like Google, device manufactures (e.g. LG, HTC, Samsung), mobile carriers (e.g. T-Mobile, Vodafone, Telefónica), and chipset makers (e.g. Intel, Qualcomm, Texas Instruments) with the common goal to develop and advance open standards for mobile devices [oha07]. The source code of the Android operating system has been released under the *Apache License version 2.0*, which makes it an open source software that allows modification and redistribution[Bou08]. The Linux kernel is mostly written in C and C++, whereas the programming language for developing applications is Java. The first commercial version of Android – Android 1.0 (API level 1) – was released on September 23, 2008. With the release of version 1.5 (API level 3) in April 2009, each major release is named in alphabetical order after a dessert or sweet, beginning with "Cupcake". The newest software version (Q1/2015) is Android 5.0.2 "Lollipop" (API level 21).

Table 2.1: Worldwide Market Share of Mobile Phone Operating Systems [Riv14]

Operating System	**Market Share Q3, 2014 (in %)**	**Market Share Q3, 2013 (in %)**
Android	83.1	82.0
iOS	12.7	12.1
Windows	3.0	3.6
Blackberry	0.8	1.8
Other OS	0.4	0.6

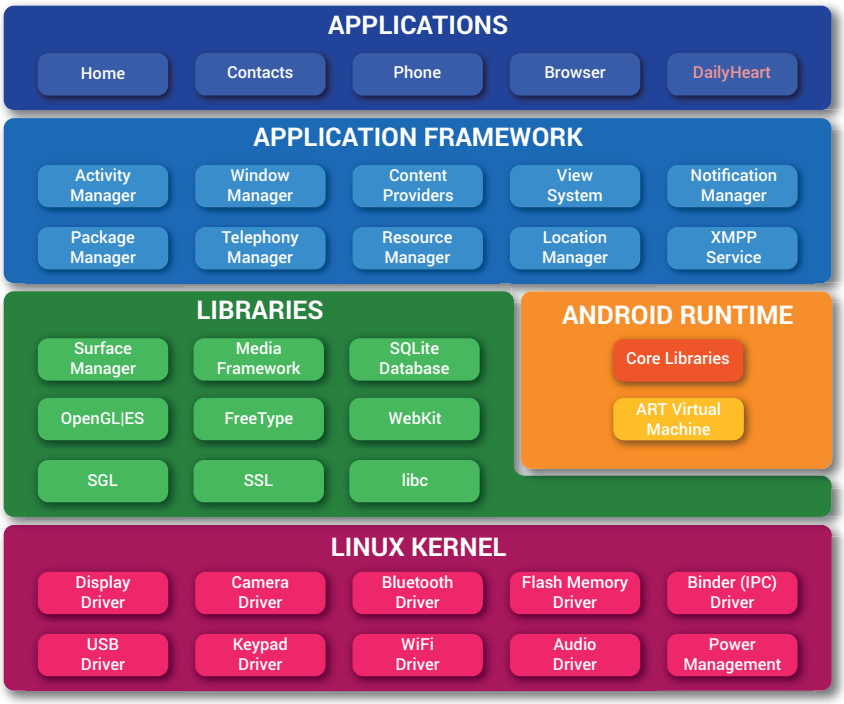


Figure 2.10: Android software architecture. The *Linux Kernel* forms the basis for the Android operating system, together with the *Libraries*, the *Android Runtime*, and the *Android Framework*.

The application in this work was developed for Android because of its open source characteristic, the portability of the Java code, and the support of *Bluetooth LE* since software version 4.3 (API level 18). Furthermore – as seen in Table 2.1 – Android has a market share of 83.1% (Q3, 2014) and hence is currently the most widely spread mobile operating system.

Software Architecture

Figure 2.10 shows the architecture of the Android operating system. The modified **Linux kernel** contains all drivers for hardware components, such as audio drivers, camera drivers, USB drivers, WiFi or Bluetooth drivers and provides the level of abstraction between the hardware and the software stack. **Android Runtime (ART)** is the runtime environment for the operating system. It includes a virtual machine, and the Android **Core Libraries**. Each application that runs on Android is executed in its own instance of the ART virtual machine. The **Application Framework** that sits on top of the ART is a collection of services forming the environment in which Android applications are

executed. For instance, it takes care of the application lifecycle and resources, manages telephony and location services, and handles the communication between different applications. The highest layer of the Android software stack consists of the several applications on the device.

Application Anatomy

The main goal of Android applications is to extend the basic functionalities of the operating system. The applications are primarily written in Java, using the Android Software Development Kit (SDK). It includes all necessary development tools, software libraries, a debugger, sample code and the documentation for the Android Framework. The two most common ***integrated development environments (IDEs)*** for developing Android applications are Eclipse (Eclipse Foundation, Ottawa, Canada) with the Android Development Tools (ADT) plugin, and Android Studio (Google Inc.) which is the officially supported Android IDE since the first stable release in December 2014 [Eas14].

The three most important components of an Android application are *Java* files, *resource* files, and the *Manifest* file. Java files define the behavior of the application, such as the interaction between user and user interface, or background processing. Resource files can be images or XML files that define the layout structure or store values such as colors, text, animation behaviors, etc. The Manifest file (`AndroidManifest.xml`) provides essential information about the application to the Android system so that it can run the application code. Every Android application consists of one or more of the of the following components, which all have to be declared in the Manifest file:

- **Activity:**[1] An activity represents a single screen with a graphical user interface to interact with the user. Activities are customized for each purpose and all implemented as a subclass of `android.app.Activity`. Although all activities are implemented separately from each other, they work together in an application for providing a cohesive user experience.

- **Service:**[2] A service runs in the background of an application and performs longer-running tasks such as downloading files or receiving data from a Bluetooth device. Services are implemented as a subclass of `android.app.Service`.

[1] http://developer.android.com/reference/android/app/Activity.html
[2] http://developer.android.com/reference/android/app/Service.html

- **Content Provider:**[3] Content providers manage the sharing and modification of data between applications. They are implemented as a subclass of `android.content.ContentProvider`.
- **Broadcast Receiver:**[4] A broadcast receiver is a component that responds to *Broadcasts*. They can be announced by the system or applications themselves. For example, a broadcast can be triggered by the system when the device was plugged in for charging or the screen has turned on. Broadcast receivers are implemented as a subclass of `android.content.BroadcastReceiver`.

Not all components are mandatory, but applications that provide a graphical user interface require at least one activity.

When an application is started by the user or by other applications, an `Intent`[5] is triggered. The *Intent Manager* of the Android Framework handles all incoming intents and starts the corresponding applications or components. If an activity should become capable of receiving a specific intent, an `IntentFilter`[6] has to be declared in the `AndroidManifest.xml` file. An application listed in the system's application launcher has to have one *launcher activity*. The activity (typically called *MainActivity*) is declared as launcher activity by assigning the `IntentFilter` in Listing 2.1 to the activity in the Manifest file.

Listing 2.1: AndroidManifest.xml

```
1 <intent-filter>
2     <action android:name="android.intent.action.MAIN"/>
3     <category android:name="android.intent.category.LAUNCHER"/>
4 </intent-filter>
```

Activity Lifecycle

Each activity is managed by the *Activity stack* of the Android system. When an application is launched, a new activity instance will be placed on top of this stack. All previous activities remain in the stack, below the newly added one, until the current activity has been exited. This process – also known as *Activity Lifecycle* – is one of the major

[3] http://developer.android.com/reference/android/content/ContentProvider.html
[4] http://developer.android.com/reference/android/content/BroadcastReceiver.html
[5] http://developer.android.com/reference/android/content/Intent.html
[6] http://developer.android.com/reference/android/content/IntentFilter.html

characteristics of the Android system. Its understanding is essential for developing useful applications. The lifecycle of an activity is visualized in Figure 2.11 and has the following four essential states:

- **Active/Running**: The activity is shown in the foreground and is able to interact with the user.
- **Paused**: The activity has lost focus (e.g. by a dialog window that overlaps the activity) but is still visible and maintains all stored information. It can be killed by the system in extreme low memory situations.
- **Stopped**: The activity is completely covered by another activty and is not visible anymore, but still maintains all stored information. It can be killed by the system in memory demanding situations.
- **Killed**: If an activity in the state *Paused* or *Stopped* was killed by the system, it has to be completely restarted and restored if it is displayed to the user again.

Every time the activity moves between states, callback methods are invoked. Each callback method can be overwritten and should be used for different purposes:

- `onCreate()`: Called when the activity is initially created. This method is used for the initialization of all necessary components of the activity, like UI components, adapter for accessing hardware features (e.g. sensor manager or Bluetooth manager), databases, etc. *onCreate()* is always followed by the call of *onStart()*.
- `onRestart()`: Called after the activity has been stopped and before *onStart()* is called.
- `onStart()`: Called when the activity is becoming visible to the user. If the activity comes to the foreground *onResume()* is called, if it is hidden *onStop()* is called. This method can for example be used for connecting clients that access APIs or for establishing a connection to a web server or a database.
- `onResume()`: Called when the activity is starting to interact with the user. At this point, the activity is on top of the activity stack and is able to receive user input. This method is used to register *event listeners* or to check if any important properties required for the correct function of the activity were changed (e.g. if *Bluetooth* was disabled). If the activity looses focus, *onPause()* is called.

- **`onPause()`**: Called when the activity looses focus of the user input, because the system is about to change to another activity. The activity is going to the background, but will not (yet) be killed. This method is used for unregistering *event listeners*, stopping actions that require a certain amount of computational load, or closing resources with exclusive access that other applications might require. The implementations in *onPause()* have to be very quick because the other activity will not resumed until this method returns. If the activity returns back to the front *onResume()* is called, if it becomes invisible to the user *onStop()* is called.

- **`onStop()`**: Called when the activity is no longer visible to the user because another activity is on top of the activity stack. This method is used for disconnecting from API clients or terminating a connection to a web server or a database. If the activity is about to be restarted *onRestart()* is called, if it is killed by the system a call for *onDestroy()* will follow.

- **`onDestroy()`**: Called right before the activity is destroyed, either when the application is finished by a calling `finish()` or because the system is killing the activity to gain memory.

As mentioned above, activities stop their connections once they loose focus. Long-running operations or background work should therefore be implemented as a `Service`. The lifecycle of a service can be completely independent from any activity as well as bound to one activity lifecycle.

Views and Listeners

The Android Framework offers a variety of UI components that can be included in an application. All widgets, such as buttons, text fields, images etc. are subclasses of `View`[7]. Its subclass `ViewGroup`[8] is the base class for every `Layout` class. Layouts are invisible containers that can hold other views (or other *ViewGroups*) and define their layout properties. All subclasses of `View` (widgets as well as layouts) are able to react on user inputs if the corresponding `Listener` is registered to the view. Listeners are nested interfaces of `View` and get notified in case of anything specific happening to the view. The most common listener is `OnClickListener`[9]. If this listener is registered to a view, a call

[7] http://developer.android.com/reference/android/view/View.html
[8] http://developer.android.com/reference/android/view/ViewGroup.html
[9] http://developer.android.com/reference/android/view/View.OnClickListener.html

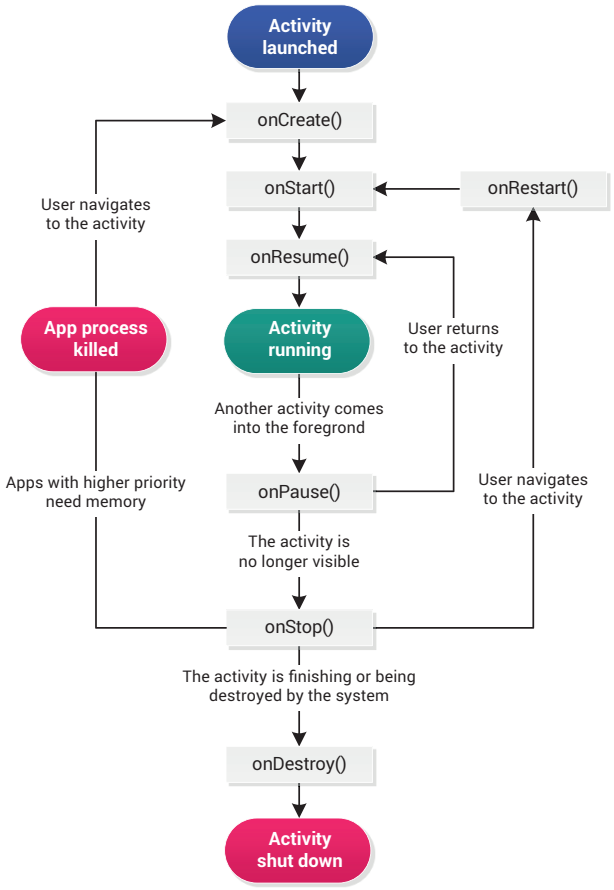


Figure 2.11: Activity Lifecycle. If an activity moves between stages, the user is notified due to a call to the corresponding callback method

to the `onClick(View v)` callback method is invoked every time a view has been tapped by the user. Listing 2.2 and Listing 2.3 demonstrate two methods of how a listener can be registered to a view. The latter method was used in this work because it provides a better overview for the implementation of multiple buttons.

Listing 2.2: Method 1: Each view implements its own OnClickListener as an anonymous class

```
1 public static class MyActivity extends Activity {
2     private Button button;
3
4     protected void onCreate(Bundle savedInstanceState) {
5         // set OnClickListener
6         button.setOnClickListener(new OnClickListener() {
```

```
            public void onClick(View v) {
            // handle click event
            }
        });
    }
}
```

Listing 2.3: Method 2: The activity implements the OnClickListener interface and all views share the same onClick() method

```
public static class MyActivity extends Activity implements
   OnClickListener {
    private Button button1;
    private Button button2;

    protected void onCreate(Bundle savedInstanceState) {
        // set OnClickListener
        button.setOnClickListener(this);
    }

    public void onClick(View v) {
        switch (v.getId()) {
            case  R.id.button1 :
                // handle click event for button1
                break;
            case  R.id.button2 :
                // handle click event for button2
                break;
        }
    }
}
```

Android 5.0 Lollipop + Material Design

The latest version of the Android operating system was unveiled at the *Google I/O* developers conference on June 25, 2014 and became publicly available on November 12,

2014 as software version 5.0 "Lollipop" (API level 21). With over 5,000 new APIs added to the system, it was the largest software change since the release of Android [Goo14b]. One of the most notable changes is the introduction of a completely new design language called *material design*. It is described by Google as a "bold, colorful, and responsive UI design for consistent intuitive experiences across all devices" [Goo14a]. The intention of material design is to use the physical surfaces and edges of materials as a metaphor and to transfer its characteristics beyond the screen following the three design principles [Goo14d]:

- **Material is the metaphor**: Transferring the fundamentals of physical material behavior like light, surface, and movement to the screen.
- **Bold, graphic, intentional**: Use properties of print-based design, like typography, grids, space, colors, and scale for creating hierarchy, meaning, and focus.
- **Motion provides meaning**: All actions take place within the same frame. Therefore, attract the primary user attention by applying motion without breaking the continuity in the user interaction.

This was accomplished consequently by the use of grid- and card-based layouts, responsive animations and transitions, print-based typography or color, and depth effects simulating light and shadow. The user interface in material design is built up in layers and visualized in Figure 2.12. The lowest layer is the *background layer* with the *content cards* right above. The *Floating Action Button*, which is a round button used for special types of actions, sits on top of the content. The topmost layer contains system elements like the *Status Bar* and the *Navigation Bar*.

2.4.2 DailyHeart – Bluetooth Connection

For a continuous monitoring of the user's ECG, the data transmitted from the *BLE-ECG Stamp* should be received and processed in the background. Because activities are not feasible for this task, a *Service* called `BLEService` was implemented for taking on this work.

Similar to activities, services have one of the two lifecycles shown in Figure 2.13. The first lifecycle desribes an unbound service. If the client calls `Context.startService(Intent)`, the system will create the service and call its callback methods `onCreate()` and `onStartCommand(Intent, int, int)` which contains the arguments from the client. At this

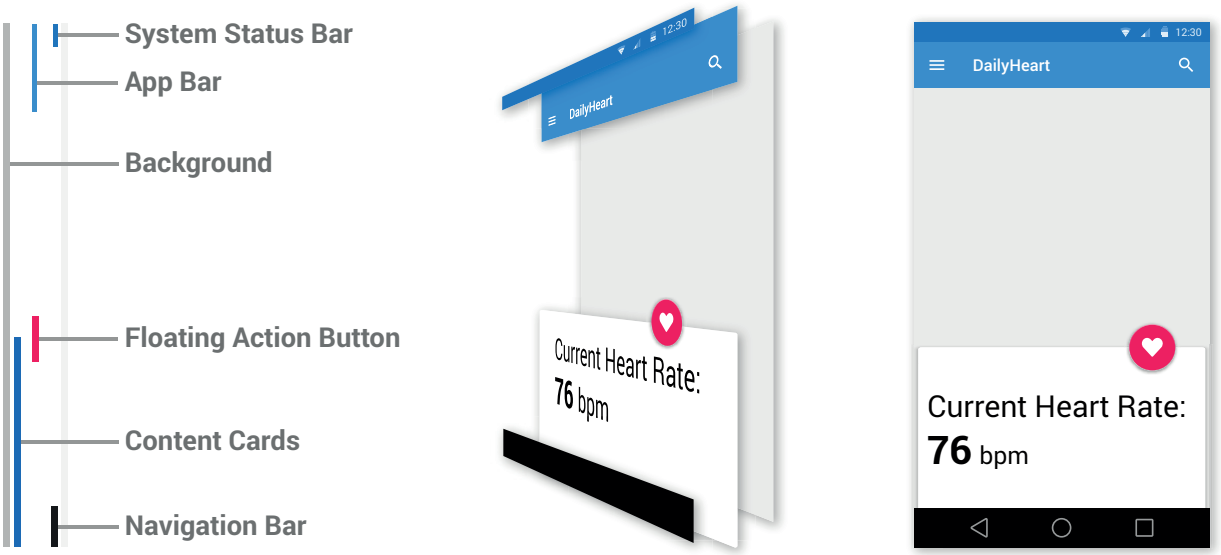


Figure 2.12: Material design layout. The user interface is built up in layers for creating a clearly arranged and natural layout.

point the service will continue running until `Context.stopService(Intent)` is called by the client or `stopSelf()` is called by the service itself. A client can also start a service by calling `Context.bindService(Intent, ServiceConnection, int)`. This will result in creating the service by calling the service's methods `onCreate()` and `onBind(Intent)` which will return an `IBinder` object for binding the service to the client. The service will be maintained alive as long as the connection is established and can be stopped by calling `unbindService(ServiceConnection)` or by finishing the client, leading to calling `onUnbind(Intent)`. Because a bound service terminates itself when the activity to which it is bounded to is finished, this solution was chosen for the implementation of the *BLEService*. The communication between service and activity is enabled by using *Handlers*. A `Handler`[10] allows to send `Message`[11] and `Runnable`[12] objects associated with the `MessageQueue` of a `Thread`. Each Handler instance is bound to the message queue in the thread is was created and will always deliver messages and runnables to that thread. This means that Handlers are able to send messages between components executed in separate threads by passing a reference of the Handler to the object in the other thread. Furthermore, Handlers can be used for scheduling messages and runnables in the same thread to be executed at a point in the future. In order to handle the messages being

[10]http://developer.android.com/reference/android/os/Handler.html
[11]http://developer.android.com/reference/android/os/Message.html
[12]http://developer.android.com/reference/java/lang/Runnable.html

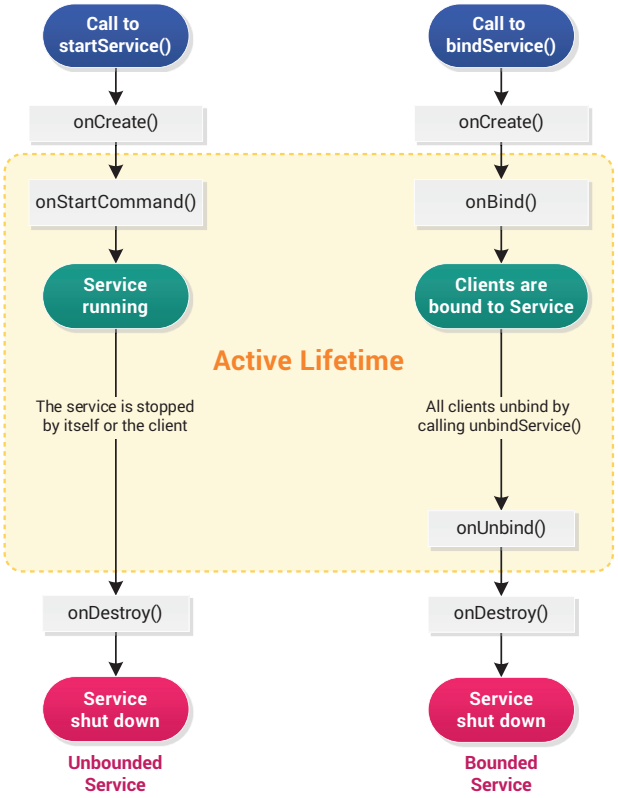


Figure 2.13: Service Lifecycles. Every service passes one of the both lifecycles, depending on if the service should be bounded to an activity or not.

sent to the Handler, the `handleMessage(Message)` method has to be overwritten when creating the Handler object, as shown in Listing 2.4. The *DailyHeart* application uses three different handlers for communication:

- **`uiHandler`** The *uiHandler* is created in the activity that starts the service and then passed to the service in order to send messages from service to UI.
- **`serviceHandler`** This Handler is used to send messages from the `BLEDeviceManager`) concerning the connection state or from the `DataProcessor` for delivering new data to the service.
- **`bleHandler`** This Handler is used for the vice-versa-messaging from all components (activities, service, etc.) to the `BLEDeviceManager`, for example to start the scan for available devices or to terminate the connection.

Listing 2.4: Handling the received messages

```
mHandler = new Handler(Message msg) {
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case MSG_CONNECT:
                // connect device
                break;
            case MSG_UPDATE:
                // update the ui
                break;
            [...]
        }
    }
};
```

Besides `BLEService`, the classes `BLEDeviceManager` and `DataProcessor` are needed for establishing a connection to the *BLE-ECG Stamp*.

BLEService

The service is started from the activity by calling `bindService(Intent, ServiceConnection, int)`. By letting the activity implement the `ServiceConnection`[13] interface, the activity is notified by the callback methods. They are called when a connection to the service has been successfully established (`public void onServiceConnected(IBinder)`) or the connection was lost (`public void onServiceDisconnected()`). The former method contains a reference to the connected service, so that methods to set the *uiHandler* (`public void BLEService.connectUiHandler(Handler)`), get the *bleHandler* (`public Handler BLEService.getBleHandler()`), and start the *Bluetooth LE* scan (`public void BLEService.startBLE()`) can be called within the activity. After a successful connection, the service can send messages to the activity with the help of the *uiHandler*. The most important messages that are used in the *DailyHeart* application are listed in Appendix C.1.

[13] http://developer.android.com/reference/android/content/ServiceConnection.html

BLEDeviceManager

By calling `BLEService.startBLE()`, the *BLEDeviceManager* is initialized. The *DeviceManager* manages any *Bluetooth* functions and can receive instructions from the activities and the service in the form of `Message` objects sent to the *bleHandler*, such as starting a scan for available *Bluetooth LE* devices, initiating a connection to a device, or terminating an existing connection to a device.

Since software version 4.3 (API level 18), the Android system offers a built-in platform for *Bluetooth LE* and provides APIs for discovering devices, query for Bluetooth services, and read/write descriptors and characteristics. Therefore, the interface `ScanCallback`[14] has to be implemented by the *DeviceManager* as well as an object of the abstract class `BluetoothGattCallback`[15]. By implementing the interface, an object of type `BluetoothLeScanner`[16] can start scanning for available *Bluetooth LE* devices and delivers the result to the callback method `public void onScanResult(int, ScanResult)`. If no device was found after five seconds, the scan is terminated automatically. If the scan result delivers a *BLE-ECG Stamp* device, it will be registered in the *DeviceManager*. Afterwards, a connection attempt is performed automatically by calling the method `public BluetoothGatt BluetoothDevice.connectGatt(Context, boolean, BluetoothGattCallback)`. This method connects to the *GATT server* that is hosted by the *BLE-ECG Stamp*, whereas the caller acts as client. The callback object is used to deliver results to the caller, such as the connection status as well as any further *GATT* client operations. Because `BluetoothGattCallback` is an abstract class, the following methods had to be implemented as seen in Listing C.2

- **`onConnectionStateChange(...)`** Method that is called if the connection state of the *Bluetooth LE* device changes. If a state change occurs the *BLEService* is notified via the *serviceHandler*.

- **`onServicesDiscovered(...)`** A call to this method is invoked if the *GATT* client is connected to the server and when the list of *services*, *characteristics* and *descriptors* for the *Bluetooth LE* device have been updated. As soon as the services were discovered, the *EcgService* is started on the *BLE-ECG Stamp*.

[14] http://developer.android.com/reference/android/bluetooth/le/ScanCallback.html
[15] http://developer.android.com/reference/android/bluetooth/BluetoothGattCallback.html
[16] http://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner.html

- **onDescriptorWrite(...)** This method indicates the result of a descriptor write operation. It is used to notify the application that the connection has been established completely.
- **onCharacteristicChanged(...)** The characteristics that are delivered as a parameter of this callback method contain the values that are transmitted from the sensor device. They are passed to the *DataProcessor* for extracting the raw data out of the characteristics.

DataProcessor

The `DataProcessor` class handles the data extraction and uses the *serviceHandler* to send the raw ECG values to the *BLEService*. The data in the characteristic was stored in a `byte[]` array with 20 entries. As mentioned in 2.2, the resolution of the sampled ECG is 12-bit, and ten samples are packaged into one characteristic. Because `byte` is an 8-bit type, the ECG samples have to be split in two `bytes`. The data extraction was implemented as described in Listing 2.5.

Listing 2.5: Extraction of ECG data from characteristics

```
1  public RawECGData[] extractData(BluetoothGattCharacteristic
       characteristic) {
2      byte[] values = characteristic.getValues();
3      RawECGData[] data = new RawECGData[values.length / 2];
4      int i = 0;
5      while (i != values.length) {
6          int tmp1 = ((values[i] & 0xFF) << 6) - 128;
7          int tmp2 = (int) values[i + 1] & 0xFF;
8          data[i / 2] = new RawECGData(timeStamp++, tmp1 + tmp2);
9          i += 2;
10     }
11 }
12 return data;
```

The extracted ECG data is processed in the *BLEService* and sent to the connected activity by the *uiHandler*. For providing a consistent way of handling the received data, all activities that receive ECG data have to implement an *interface* called `IDailyHeart`. This interface provides the following methods:

- **public void onStartStreaming()**: This method is called when the BLEDeviceManager sends a message that the *Bluetooth LE* device is ready and starts the streaming process.
- **public void onSegmentationFinished()**: This method is called when the BLEService sends a message indicating the end of a heart beat segmentation.
- **public void onDataReceived(RawECGData data)**: This method receives the incoming ECG data from the *BLEService* and handles the plotting event as well as activity-specific actions.
- **public void startDailyHeart()**: This method initializes the measurement as it binds the BLEService.
- **public void stopDailyHeart()**: This method terminates the measurement by requesting the BLEDeviceManager to disconnect from the *BLE-ECG Stamp*, as well as saves all acquired data and starts the activity that display the results.

BLEDataSimulator

The BLEDataSimulator class was designed to fit into the existing Bluetooth architecture of the application. It also uses the *BLEService* and the *BLEDeviceManager*, but instead of scanning for available *BLE* devices the file to be simulated is chosen and is passed to the data simulator. Furthermore, instead of calling connectECGDevice() for connecting to a *BLE-ECG Stamp*, connectSimDevice() is called in order to connect to the simulator and start the simulation. The file is loaded from the external storage by passing the path to the selected file to an object from the type InputStream[17]. The BufferedReader[18] wraps the existing InputStreamReader[19] (that turns the byte stream from the InputStream into a character stream) and buffers the input. The method BufferedReader.readLine() returns the next line of text available from the reader. Every line contains of the current timestamp and the ECG value that are delimited by a character. Listing 2.6 illustrates how the BufferedReader is initialized and how the lines are read consecutively. A SimpleStringSplitter[20] is used to split the current line at the delimiter.

[17] http://developer.android.com/reference/java/io/InputStream.html
[18] http://developer.android.com/reference/java/io/BufferedReader.html
[19] http://developer.android.com/reference/java/io/InputStreamReader.html
[20] http://developer.android.com/reference/android/text/TextUtils.SimpleStringSplitter.html

Listing 2.6: Loading the file from the external storage and extracting the data

```
// initialize BufferedReader
BufferedReader buff = new BufferedReader(new
   InputStreamReader(new InputStream(selectedFile)));

// read the file to the end
private void receiveData() {
    SimpleStringSplitter splitter = new
       SimpleStringSplitter(delim);
    String line;

    while ((line = bufferedReader.readLine()) != null) {
        RawECGData data = new RawECGData();
        splitter.setString(line);
        if (splitter.hasNext()) {
            data.timeStamp = splitter.next();
        }
        if (splitter.hasNext()) {
            data.ecgRaw = splitter.next();
        }

        bleHandler.obtainMessage(MSG_DATA, data).sendToTarget();
        // send thread to sleep in order to simulate live
           conditions
        Thread.sleep(1000 / samplingRate);
    }
}
```

2.4.3 DailyHeart – User Interface

The new design principles of material design were incorporated for designing the user interface of the *DailyHeart* application. The application consists of several activities, each of which has its own function. All implemented activities are subclasses of an abstract class named `BaseActivity`, which in turn inherits from `Activity`. *BaseActivity* handles

the common functionality of the application. It sets up the *navigation drawer* that allows the user to navigate between activities, establishes and terminates connections to API clients, and connects to the *DailyHeart database* that stores the results of the recordings. The activities that provide a user interface for this application are described in the following subsections.

MainActivity

This activity represents the main menu of *DailyHeart*. Its layout consists of *card views* that represent the different features of the application and will be started by clicking on the cards (see Figure2.14a).

CurrentHRActivity

This activity implements the functionalities to measure the user's current heart rate. By tapping the *Floating Action Button*, the *BLEService* is bound, the application automatically starts scanning for available devices, connects if possible, and starts the measurement after a successful connection has been established. It is possible to customize the duration of the measurement between 5 seconds and 20 seconds. The progress of the measurement is visualized by a circular widget called `CircularProgressBar` that updates the progress continuously. It surrounds an image of a heart, animated in order to create a *pulsating effect* (see Figure 2.14b).

In general, views can be animated in different ways. A possible way is to define the behavior of the animation as XML file, load the file with the `AnimationUtils`[21] class and start the animation.

Listing 2.7: XML file defining the behavior of the animation

```
1 <scale
2     android:fromXScale="1.0"
3     android:toXScale="1.1"
4     android:fromYScale="1.0"
5     android:toYScale="1.1"
6     <!-- The point in the View where the animation starts -->
7     android:pivotX="50%"
8     android:pivotY="50%"
```

[21] http://developer.android.com/reference/android/view/animation/AnimationUtils.html

```
    android:fillAfter="true"
    android:duration="400"
/>
```

Listing 2.8: Animating a View

```
Animation animHeart = AnimationUtils.loadAnimation(this, R.anim.
    anim_heart);
heartImage.startAnimation(animHeart);
```

At the end of the measurement, the activity is finished automatically and a new activity from the type `CurrentHRResultActivity` is launched. Therefore, all relevant data is added as key/value pairs (*Extras*) to the *Intent* that starts the activity as described in Listing 2.9.

Listing 2.9: Adding Extras and starting a new Activity

```
Intent intent = new Intent(this, CurrentHRResultActivity.class);
intent.putExtra(Constants.EXTRA_HEART_RATE_CURR, heartRate)
    .putExtra(Constants.EXTRA_START_TIME, startTime)
    .putExtra(Constants.EXTRA_ECG_GSON, ecgGsonString);
startActivity(intent);
```

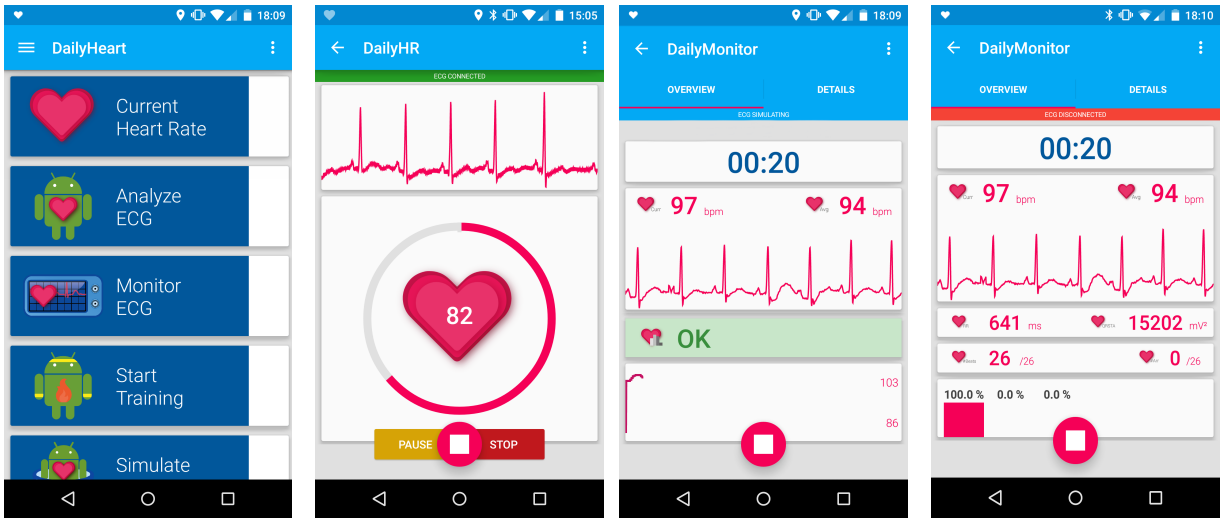
The Extras being added are an Integer value containing the measured heart rate, a Long value containing the current time, and a ***JSON***-String storing the recorded ECG raw values with its associated time stamps.

CurrentHRResultActivity

This activity displays the result of the measurement acquired by the *CurrentHRActivity*. It also provides buttons to save the results in the *DailyHeart database* and *Google Fit* or to restart the measurement. These features will be described in later sections.

AnalyzeECGActivity

The structure of this activity is similar to *CurrentHRActivity*. Its function is to analyze and classify the user's ECG over a defined period of time. For visualizing the results of the recording, they are again passed to another activity called *AnalyzeECGResultActivity*.



(a) MainActivity **(b)** CurrentHRActivity **(c)** DailyMonitor Activity – Tab 1 **(d)** DailyMonitor Activity – Tab 2

Figure 2.14: User interface of *DailyHeart*. The layout of *DailyHeart* was designed with respect to the material design guidelines in order to provide an appealing user interface; *(a) MainActivity, (b) CurrentHRActivity, (c) DailyMonitorActivity – Tab 1, (d) DailyMonitorActivity – Tab 2*

AnalyzeECGResultActivity

The result activity for the *AnalyzeECG* mode. The most important information that was acquired during the measurement (average heart rate, average R-R interval, average QRS complex area, number of normal beats, and number of abnormal beats) is displayed to the user. This activity also provides buttons for saving and remeasuring.

DailyMonitorActivity

In contrast to the *CurrentHR* and the *AnalyzeECG* mode the measurement in the *DailyMonitor* mode does not stop after a certain period of time, but is able to monitor the user's ECG continuously. In order to provide more information during and after the measurement, two tabs the user can switch between were implemented. The first tab displays a `PlotView` showing the evolution of the heart rate during the measurement, and a `CardView`[22] showing the current heart status. The second tab periodically updates the information obtained from the *Pan-Tompkins-Algorithm* and contains bar charts that visualize the relation between the different classes of heart beats. These bar charts are

[22]http://developer.android.com/reference/android/support/v7/widget/CardView.html

implemented as views whose height is changed dynamically according to the ratio of the heart beat classes as described in Listing 2.10 (see Figure 2.14c and Figure 2.14d).

Listing 2.10: Dynamically changing the height of Views

```
// Compute the ratio of normal classified beats in percent
double percentNormal = (((double) normalBeats) / totalBeats) *
   100;
// Get access to the layout parameters
ViewGroup.LayoutParams params = barView.getLayoutParams();
// Set the Height depending of the maximal Height of the layout
   container;
params.height  = percentNormal * MAX_HEIGHT;
// set the new Height
barView.setLayoutParams(params);
```

DailyMonitorResultActivity

This activity shows the acquired information from the *DailyMonitor* mode. In contrast to the *CurrentHR* mode and the *AnalyzeECG* mode, the course of heart rate is passed to the *DailyMonitorResultActivity* instead of raw ECG values.

DailySimulateActivity and DailySimulateResultActivity

These two activities are generally the same as the activities for the *DailyMonitor* mode, but used for simulating previously recorded data. A *Switch* widget lets the user choose between ECG data recorded with the *DailyHeart* application or ECG data from the *MIT-BIH Arrhythmia Database*. The file to be simulated is chosen from the external storage with a *Dialog*.

HistoryActivity

The *HistoryActivity* lists all previous measurements that are stored in the internal database in a *RecyclerView*, which is a special *View* that can flexibly display a window of a large data set. Therefore, a *HistoryAdapter* (a subclass from `RecyclerView.Adapter`) has been implemented that maps the entries from the database to the *RecyclerView*.

HistoryDetailsActivity

By clicking on an item in the *HistoryActivity*, the ID of the selected measurement is passed to a new instance of *HistoryDetailsActivity*. It displays all information of the measurement that was obtained from the database. Because the kind of information depends on the record type (*CurrentHR*, *AnalyzeECG* or *DailyMonitor*), the layout of the activity consists of a *ViewPager* that can be dynamically adapted to the record type and filled with one or more tabs.

SettingsActivity

This is the activity for configuring the settings of *DailyHeart*. The user can set the duration of the *CurrentHR* and the *AnalyzeECG* measurements, enable or disable a tone being played for every heart beat, or configure his profile. These preferences are managed by the `PrefUtils` class. Listing 2.11 demonstrates how a `Preference`[23] is applied to the `PreferenceManager`[24] and then retrieved again.

Listing 2.11: Applying and retrieving a Preference

```
1  public static void markUserSoundEnabled(Context context,
       boolean enabled) {
2      // get instance from the PreferenceManager
3      SharedPreferences sp =
           PreferenceManager.getDefaultSharedPreferences(context);
4      // make the SharedPreferences instance editable and apply
           the preference as a key/value-pair
5      sp.edit().putBoolean(PREF_SOUND_ENABLED, enabled).apply();
6  }
7
8  // retrieve the preference
9  public static boolean hasUserSoundEnabled(Context context) {
10     SharedPreferences sp =
           PreferenceManager.getDefaultSharedPreferences(context);
11     return sp.getBoolean(PREF_SOUND_ENABLED, true);
12 }
```

[23] http://developer.android.com/reference/android/preference/Preference.html
[24] http://developer.android.com/reference/android/preference/PreferenceManager.html

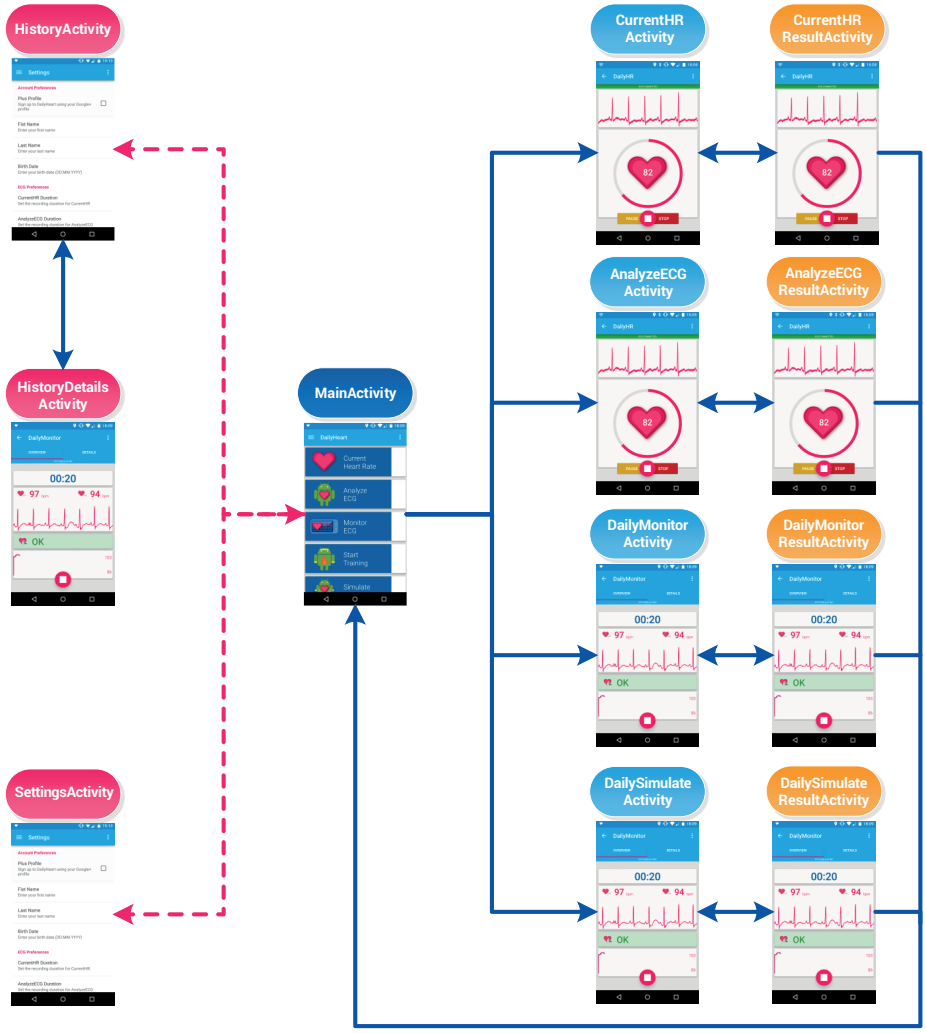


Figure 2.15: Navigation branch of *DailyHeart*. *red, dashed:* transition between activities via navigation drawer; *blue, solid:* transition between activities by tapping a button.

Navigation Drawer

A *navigation drawer* is a panel that transitions from the left edge of the screen and displays the main navigation options of the application. It can be brought to the front by swiping from the left edge of the screen or by touching the application icon on the action bar. In the *DailyHeart* application the navigation drawer is used as top level navigation for switching between the *MainActivity*, the *HistoryActivity*, and the *SettingsActivity*. The whole navigation branch of *DailyHeart* can be seen in Figure 2.15.

StatusBar and HeartStatusCardView

Besides the `CircularProgressBar`, other custom widgets have been implemented, such as a `StatusBar`. It is a subclass of `TextView` and is able to change its appearance depending on the following connection states:

- **Disconnected**. No connection between application and *Bluetooth LE* device because there was no connection attempt yet, the connection attempt has failed, the connection has been terminated, or there is no device to connect available.
- **Connecting**. There is an available device and the *BLEDeviceManager* tries to establish the connection.
- **Connected**. The *Bluetooth LE* device reports that the connection has been fully established and the device starts to stream.
- **Simulating**. The application is not in live mode, but simulating previously recorded ECG data.

`HeartStatusCardView` is a subclass of `CardView`. Similar to the `StatusBar`, it changes its appearance dynamically according to the current heart status.

2.4.4 DailyHeart – Database and Google Fit

An essential feature for an application that monitors the user's cardiac status is the option to save the recorded data. Therefore, a database was developed in the present work that allows to store all relevant data and to perform queries for retrieving the data and display it in the `HistoryActivity`. The Android operating system is shipped with a `SQLite database`[25] (version 3.4.0) and provides an API for database management.

Before implementing the database, one has to carefully think about the structure of the database, because extending an existing database is quite inconvenient and not recommended. The structure of the developed *DailyHeart Database* and all columns can be seen in Appendix C.3. For managing the creation and upgrade of a *SQLite database* within the application, Android provides the abstract class `SQLiteOpenHelper`[26]. The subclass `DailyHeartDBHelper` needs to implement the methods `onCreate()` and `onUpgrade()`.

[25] http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html
[26] http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

SQL databases are managed by *SQL statements* that are lines of SQL code acting as commands and executed against the database. Each statement begins with a *SQL clause* that tells the database which operation is to be performed. SQL clauses include, among others: **CREATE** (creation of a new database), **ADD** (adding new data), **SELECT** (retrieving data), **DELETE** (deleting data). Listing 2.12 shows a SQL statement for the creation of a database named *hearttable* with three columns: an *id* column that automatically increments the row id when inserting, a *name* column for storing strings and a *heartrate* column for storing integers.

Listing 2.12: SQLite statement for creating a database

```
1 CREATE table if not exists hearttable(id integer primary
    key autoincrement ,name text,heartrate integer);
```

By calling the method `SQLiteDatabase.execSQL(String statement)`, the SQL statement is executed.

Data can be added to the database by calling `SQLiteDatabase.insert(String, ContentValues)`. The latter parameter is a map that contains the data for the row to be inserted. The keys are the column names whereas the values are the column values of the current dataset. If database entries should be retrieved from the database again the table has to be queried by `SQLiteDatabase.query(...)`. This method has different signatures but always returns a `Cursor`[27] object over the result set. The most important parameters of the `query()` method are:

- String ***table*** The table name to compile the query against.
- String[] ***columns*** An array of the columns to be returned. Passing `null` will return all columns.
- String ***selection*** The filter criterion indicating which rows to return. Passing `null` will return all rows.
- String ***orderBy*** The way the results are ordered in the `Cursor`.

By converting the `Cursor` back to an object from type `DailyHeartData` the data can be extracted and displayed in the user interface (UI). It is also possible to compile the table against more complex queries for customizing the data that is displayed in the history. This means the user could filter the data by time interval (last week, last

[27] http://developer.android.com/reference/android/database/Cursor.html

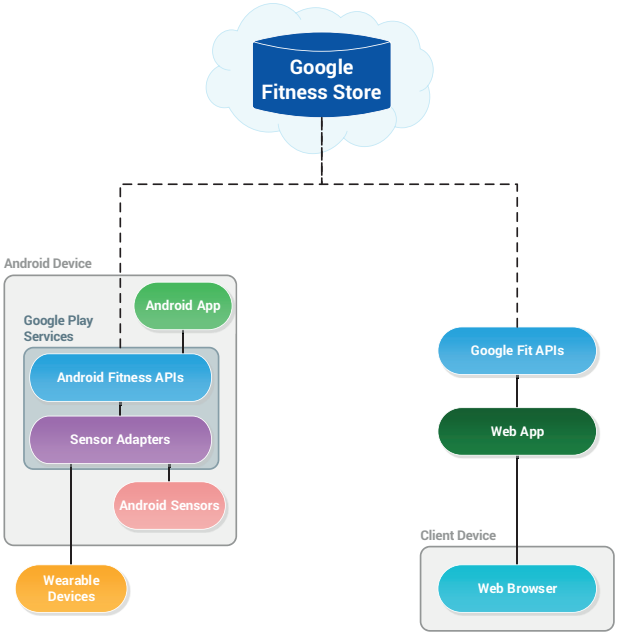


Figure 2.16: Google Fit platform overview. The platform provides common access for fitness date on mobile devices and web clients.

month, etc.), record type, or record duration. Furthermore, a *ContentProvider* (named `DailyHeartContentProvider`) was implemented in this work, but has not been used. A content provider encapsulates the data for providing content from the *DailyHeart* application to other applications. In combination with a `ContentResolver`[28] it acts as an interface for accessing the data. Besides the storage in the internal *DailyHeart database*, the user's heart rate can also be exported to *Google Fit*. It is a health-tracking platform by *Google* and was unveiled at the *Google I/O* developers conference in June 2014 and was launched to the public later in October. *Google Fit* acts as a central platform where users can access their fitness and health data from different devices and apps. It can store data collected by sensors from mobile devices, wearables or activity trackers, and can access data that was created by other apps. An overview of the *Google Fit* platform is shown in Figure 2.16. Data from an Android app can be uploaded to *Google Fit* by accessing the *Google Fit APIs* for Android. They are part of *Google Play services*[29], which means they are part of any versions of Android 2.3 (API level 9) or higher, and contain the following APIs:

[28] http://developer.android.com/reference/android/content/ContentResolver.html
[29] http://developer.android.com/google/play-services/index.html

- **Sensors API**[30]: It provides access to raw sensor data streams from internal sensors on the Android device or wearables.

- **Recording API**[31]: It provides automated background storage of fitness data in a battery-efficient manner

- **History API**[32]: This API provides access to the fitness history and can be used by applications to insert, delete, or read fitness data.

- **Sessions API**[33]: The Sessions API allows to store fitness data with sessions metadata, such as time intervals during which the user performs a fitness activity.

- **Bluetooth Low Energy API**[34]: It provides access to *Bluetooth LE* sensors in *Google Fit*. It can scan for available *BLE* devices and to store their date.

If *Google Play services* are enabled on the Android device and a *Google Account* is available, the *Fitness API* has to be activated in the *Google Developers Console*[35] in order to authenticate and communicate with *Google Fit*. After this, a connection to the fitness service has to be established in the application. Therefore, the application requests a connection to the fitness service with one or more *scopes* of access. After that, *Google Fit* prompts the user to grant the application the required permissions. If these are granted, the application can access fitness data of the types defined by the scopes. The available data types are defined in the `DataType`[36] class. They are grouped into:

- **Activity**: Data types that record the activity of the user. such as burned calories, pedaling rate, generated power or performed steps.
  Scope for granting permission: `FITNESS_ACTIVITY_READ_WRITE`

- **Body**: Data types that record parameters of the user's body, such as heart rate, height or weight.
  Scope for granting permission: `FITNESS_BODY_READ_WRITE`

[30] http://developer.android.com/reference/com/google/android/gms/fitness/SensorsApi.html
[31] http://developer.android.com/reference/com/google/android/gms/fitness/RecordingApi.html
[32] http://developer.android.com/reference/com/google/android/gms/fitness/HistoryApi.html
[33] http://developer.android.com/reference/com/google/android/gms/fitness/SessionsApi.html
[34] http://developer.android.com/reference/com/google/android/gms/fitness/BleApi.html
[35] http://www.console.developers.google.com
[36] http://developer.android.com/reference/com/google/android/gms/fitness/data/DataType.html

- **Location**: Data types that record location parameters, such as location, distance or speed.
  Scope for granting permission: `FITNESS_LOCATION_READ_WRITE`

The `GoogleApiClient`[37] for the *Fitness API* is then created as shown in Listing 2.13:

Listing 2.13: Building the Fit client

```
client = new GoogleApiClient.Builder(this)
        .addApi(Fitness.API)
        .addScope(new Scope(Scopes.BODY_LOCATION_READ_WRITE)
        .addConnectionCallbacks(...)
        .addOnConnectionFailedListener(...)
        .build();
```

As described in section 2.4.1, it is recommended to connect to clients in the `onStart()` method of the activity, and disconnect again in the `onStop()` method.

If the user presses the *Save* button in one of the result activities, the results of the measurement are inserted into the *Google Fitness Store*. Therefore, a `DataPoint`[38] has to be created and inserted into a `DataSet`[39] object by calling `Fitness.HistoryApi.insertData(...)`. If the data has been inserted into the *Google Fitness Store*, it can be seen in the *Google Fit* application for Android (like seen in Figure 2.17) or as an application in a web browser.

2.4.5 Android Wear

Besides the visualization of ECG data as an application for mobile phones, this work also focused on an extension of the *DailyHeart* application for bringing it to wearables like *Google Glass* and smartwatches based on *Android Wear*.

Android Wear is a modified version of the Android operating system, especially designed to extend Android on smartwatches and other wearables by integrating smartphone features with a watch form factor. It was unveiled on March 18, 2014, and the first smartwatches running on *Android Wear* were launched at *Google I/O* on June 25, 2014. Android Wear supports both rectangular and circular shaped watch faces. There are two ways to display information on Android Wear: Adding wearable features to

[37] http://developer.android.com/reference/com/google/android/gms/common/api/GoogleApiClient.html
[38] http://developer.android.com/reference/com/google/android/gms/fitness/data/DataPoint.html
[39] http://developer.android.com/reference/com/google/android/gms/fitness/data/DataSet.html

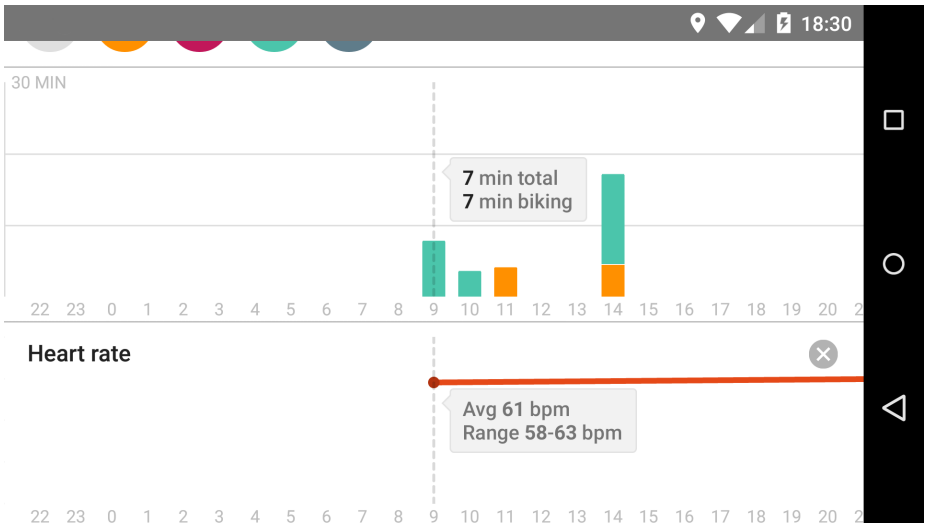


Figure 2.17: Google Fit Application. The *DailyHeart* data that was uploaded to Google Fit can be visualized in the Android app, together with additional information of physical activity.

regular Android notifications that are synchronized between mobile phone and wearable, or creating custom apps for both phone and wearable that are able to send and sync data mutually. The *DailyHeart* application uses a combination of both features.

Notification Sync

Notifications are an essential part of the Android operating system's user interface. They are messages that are displayed outside of the application's regular UI. A notification first appears as an icon in the *notification area* of the system's status bar. If the user swipes down the status bar, the *notification drawer* is expanded and reveals details of the notification. In the *DailyHeart* application, the `DailyHeartNotificationManager` class handles the creation and the update of notifications. They are built with a `Notification.Builder`[40] that returns a `Notification`[41] object when calling the `Builder.build()` method. The notification is displayed when passing the `Notification` object to the system by calling `NotificationManager.notify()`[42]. The code sample in Listing 2.14 shows how a notification with minimal required content is created:

[40] http://developer.android.com/reference/android/app/Notification.Builder.html
[41] http://developer.android.com/reference/android/app/Notification.html
[42] http://developer.android.com/reference/android/app/NotificationManager.html

Listing 2.14: Creating a notification

```
// minimal content required for a notification
Notification.Builder builder = new Notification.Builder(context)
    .setSmallIcon(R.drawable.ic_small)
    .setContentTitle("DailyHeart Notification")
    .setContentText("This is a notification!");

// Build notification and notify the system
NotificationManager notificationManager = (NotificationManager)
   getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.update(id, builder.build());
```

The notifications created for the phone are automatically shared with the wearable, but can be customized in various ways for both handheld and wearable notifications. By adding Intents to the `Notification.Builder` with a call to `Builder.setContentIntent(Intent)`, an activity can be launched when clicking on the notification. Furthermore, notifications can be extended with a *big view style* that shows more text when being expanded. In addition to this content and the action defined by `setContentIntent()`, other actions can be added to the notification by passing a `PendingIntent`[43] object to the `addAction()` method of the `Builder`. On the handheld (either a smartphone or tablet), the extra action appears as an additional button attached to the bottom of the notification, whereas on a wearable it appears as a large button when the user swipes the notification to the left. When the action button is tapped, the associated intent behind the action is executed on the handheld. If the actions should only be visible on wearable notifications but not on the handheld, a call to `WearableExtender.addAction()` has to be invoked. An instance of `WearableExtender`[44] is applied to the notification by calling `Builder.extend(WearableExtender)`.

The *DailyHeart* application uses different notification schemes for the different types of activities:

`MainActivity` On the handheld, the notification shown during the main menu does not contain any actions, whereas on the wearable, it displays action buttons to start a recording (*CurrentHR*, *AnalyzeECG* or *DailyMonitor*). Listing 2.15 describes how actions

[43] http://developer.android.com/reference/android/app/PendingIntent.html
[44] http://developer.android.com/reference/android/app/Notification.WearableExtender.html

are defined and added to the `Notification.Builder`:

Listing 2.15: Adding actions to a notification

```
Intent currentHrIntent = new Intent();
// set action constant to define intent
currentHrIntent.setAction(ACTION_CURRENTHR);
// add request code as extra to intent for the BroadcastReceiver
currentHrIntent.putExtra(BROADCAST_KEY_ACTION,
   RQS_START_CURRENT_HR);
// retrieve PendingIntent that will perform a broadcast
PendingIntent hrPendingIntent =
   PendingIntent.getBroadcast(context, RQS_START_CURRENTHR,
   currentHrIntent, 0);
// build actions for extending the notification
Action hrAction = new Action.Builder(R.drawable.ic_current_hr,
   "CurrentHr", hrPendingIntent).build();
builder.extend(new WearableExtender().addAction(hrAction));
```

When the user taps an action button, the PendingIntent behind the button is invoked and a broadcast is fired. In order to receive the broadcast in the `MainActivity`, an `IntentFilter` and a `BroadcastReceiver` have to be created in the activity's `onCreate()` method. If the broadcast's action is registered in the intent filter, a call to the broadcast receiver's `onReceive(Context, Intent)` method is invoked. As shown in Listing 2.16, an activity is started according to the action of the received intent. By adding an `boolean` as extra to the intent, the activity on the handheld will be started automatically.

Listing 2.16: Receiving and handling actions in the MainActivity

```
protected void onCreate(Bundle savedinstanceState) {
    [...]
    IntentFilter intentFilter = new IntentFilter();
    intentfilter.addAction(ACTION_CURRENTHR);
    BroadcastReceiver receiver = new ActivityReceiver();
    registerReceiver(receiver, intentFilter);
}


```

```
10 public class ActivityReceiver extends BroadcastReceiver {
11     public void onReceive(Context context, Intent intent) {
12         switch (intent.getIntExtra(BROADCAST_KEY_ACTION, -1)) {
13             case RQS_START_CURRENTHR:
14                 Intent startIntent = new
                       Intent(MainActivity.this,
                       CurrentHRActivity.class);
15                 startIntent.putExtra(EXTRA_ACTION_START, true);
16                 startActivity(startIntent);
17                 break;
18             [...]
19         }
20     }
21 }
```

CurrentHR-, AnalyzeECG-, DailyMonitorActivity The notifications for these activities display all relevant information that is also displayed in the activity's UI. The notification displayed during the *DailyMonitor* mode uses a `WearableExtender` for adding extra pages with additional information to the wearable notification. All notifications contain two buttons to pause/resume and stop the current recording from the wearable or from the handheld, even if the activity is not in the foreground.

CurrentHR-, AnalyzeECG-, DailyMonitorResultActivity After the recording has ended, the notifications are updated and show all acquired information at a glance. They were extended with action buttons to restart the measurement or to quit the activity and return to the main menu.

Custom Applications for Wear

In contrast to notifications extended with wearable features, wearable applications run directly on the device providing access to hardware components such as sensor and GPU. In general, they developed in the same way as Android applications for handhelds, but they differ greatly in terms of design and functionality. Therefore, it only makes sense to display the most necessary information, usually a small subset of the corresponding information on the handheld application. Furthermore, the Wear system enforces a

sleep period when the user does not interact with it. When the device wakes up again, the home screen is displayed instead of the previous activity being resumed. For this reason, continuous information should only be displayed as notifications and the usage of wearable applications should be reduced to the interaction between user and device.

Wearable apps can not be downloaded to the device directly and are not able to run without a companion app on the handheld device. Instead, the wearable app is bundled inside the handheld app and is automatically installed on the wearable once the application is installed on the handheld. For the *DailyHeart* application a wearable app was developed in order to enable voice capabilities and to fully control the application from the wearable.

The Wear application developed in this work consists of three activities, which were especially designed for a smartwatch form factor. Screenshots of *DailyHeart* for Wear are shown in Figure 2.18:

`MainActivity` This activity acts as the main menu on the watch, providing a `WearableListView` (a widget modified for displaying lists on a wearable) to select the activity to be started.

`CurrentHRActivity`, `AnalyzeECGActivity` These activities are the equivalents to the `CurrentHRActivity` and `AnalyzeECGActivity` on the handheld, with a simplified layout. By tapping the screen, the measurement is initialized by sending a message to the handheld device via the `MessageApi`[45] of Android Wear. This API is intended to perform such operations, also called *remote procedure calls (RPC)*. It can also be used for one-way requests or request/response communication models. The messages sent from the wearable are received on the handheld by a `WearableListenerService`[46]. After the handheld application established a connection to the *BLE-ECG Stamp*, the Wear application receives a message from the handheld to start the animation and notify the user that the measurement has begun. Furthermore, the handheld sends a message containing all results at the end of the measurement.

Voice capabilities are an important part of the interaction between wearable and user, since it allows the user to execute commands quickly and hands-free. The Android Wear platform supports two types of voice actions:

[45] http://developer.android.com/reference/com/google/android/gms/wearable/MessageApi.html

[46] http://developer.android.com/reference/com/google/android/gms/wearable/WearableListenerService.html

- **System-provided actions**: These voice actions are task-based and built into the system. They can be activated for an activity by declaring an `IntentFilter` in the manifest file of the Wear application. The system-provided voice intents can be found at the developer site of Android Wear[47]. The *DailyHeart* application uses a voice intent that triggers a heart rate measurement which has to be declared in the manifest file like this:

Listing 2.17: Declating a voice intent in the wearable app's manifest file

```
<activity
    android:name="de.lme.dailyheart.ui.CurrentHRActivity">
    <intent-filter>
        <action android:name="vnd.google.fitness.VIEW" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="vnd.google.fitness.data_type/
            com.google.heart_rate.bpm" />
    </intent-filter>
</activity>
```

  This voice intent allows the user to start the `CurrentHRActivity` on the wearable by saying either the phrase "Okay Google, what's my heart rate?" or "Okay Google, what's my bpm?".

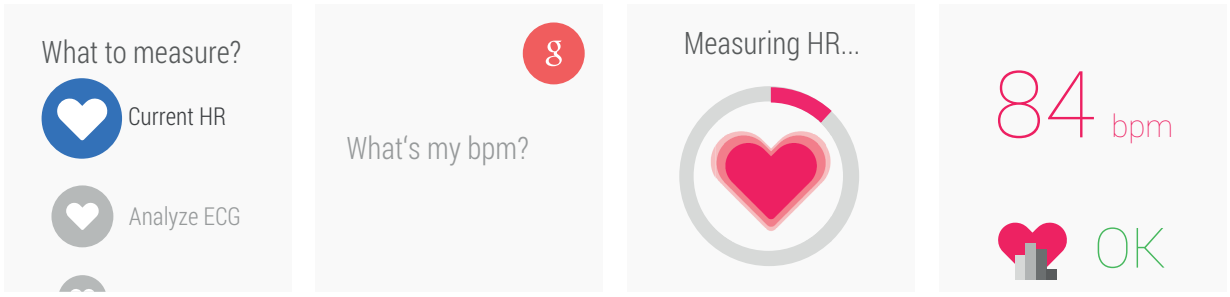
- **App-provided actions**: These actions are app-based and can be started by the user saying "Start" followed by the name of the application or activity, for example: "Start DailyHeart". This voice action results in starting the `MainActivity` of the Wear application.

With the combination of both types of voice actions, the *DailyHeart* application can be fully controlled from the wearable without having to use the handheld.

2.4.6 Google Glass

Google Glass (often abbreviated as *Glass*) is a wearable device with an ***optical head-mounted display (OHMD)***. It was developed by *Google X*, a semi-secret research facility by Google, and publicly announced in April 2012. In April 2013, the *Glass Explorer*

[47] http://developer.android.com/training/wearables/apps/voice.html#SystemProvided



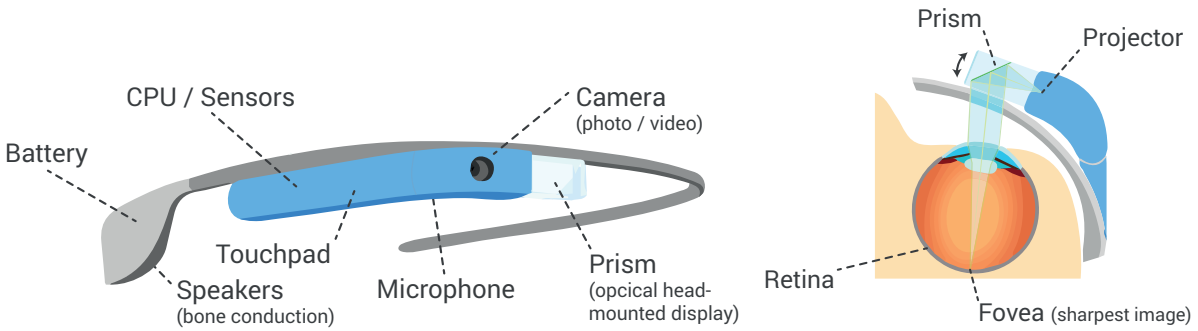
(a) Wear – Main Menu **(b)** Wear – Voice Capabilities **(c)** Wear – CurrentHR **(d)** Wear – CurrentHR Result

Figure 2.18: Android Wear application. User interface of *DailyHeart* designed for Android Wear: *(a)*: Main Menu to select a mode; *(b)*: Dialog to start the application by voice input; *(c)*: Heart rate measurement dialog on Wear; *(d)*: Heart rate result dialog on Wear.

Edition, a beta version, was released in the United States, making Google Glass prototypes available to developers for $1,500. On January 15, 2015 Google announced the end of the Explorer beta phase at *Google X*, but will it continue developing it further [Goo15].

Google Glass consists of a 640x360 *Liquid Crystal on Silicon (LCoS)* display (whose illumination is projected by a prism into the wearer's eye as seen in Figure 2.19b), a 5 megapixel camera, 16 GB internal storage, a 1.2 GHz SoC Dual-core processor by Texas Instruments (Dallas, TX, USA), as well as 2 GB RAM. It has several built-in sensors, including a 3-axis accelerometer, a 3-axis gyroscope, a 3-axis magnetometer, an ambient light sensor, a proximity sensor, and a GPS receiver. Furthermore it features Wi-Fi and Bluetooth chips for a wireless connection. Glass can receive command inputs by voice commands or by swiping and clicking on a touchpad located on the right side of the frame. A schematic representation of Glass is shown in Figure 2.19a. With the *MyGlass* companion app for Android and iOS the user can configure and manage the Glass as well as install applications called *Glassware* on the device.

There are three ways of developing applications for *Google Glass*: Using the **Glass Development Kit (GDK)**, the **Mirror API**, or the **Notification Sync** known from *Android Wear*. GDK is an add-on to the Android SDK which is used for developing *Glassware* applications that run directly on the wearable. Glassware applications have direct access to the hardware features of Glass and can run independently from a mobile phone. A disadvantage of using the GDK is that there is no possibility of sending messages between handheld and Glass via *Bluetooth* as in the case of the *Android Wear* platform. For that reason the GDK has not been used in this project.



(a) Google Glass – Main components

(b) Google Glass – Display Technique

Figure 2.19: Google Glass components. *(a)*: Schematic visualization of Google Glass; *(b)*: Technical principle of the optical head-mounted display.

The Mirror API is an API that can be used for building platform-independent and web-based Glassware applications. It is a cloud-based API and therefore does not require code running on Glass itself. The Mirror API creates cards that are displayed in the timeline using *REST API* commands. These cards can be extended with location or media information. This means that the *Mirror API* needs continuous internet connection for creating and updating the Glass timeline. The Glass software was updated to *XE22* in October 2014, introducing a Notification Sync feature that displays notifications from the Android phone on Glass in the same way like on *Android Wear*. Due to the lack of advantage the *Mirror API* provides for this application compared to the *Notification Sync* – and its simple implementation – the latter method was used to display information on the optical head-mounted display of Glass.

2.5 Evaluation

The evaluation of the present work is twofold: Firstly, the signal received from the *BLE-ECG Stamp* is evaluated quantitatively in order to assure that the signal quality is sufficient for a subsequent data processing and classification. Secondly, the developed application – especially the interaction with the wearables – is evaluated qualitatively in a suitability test for activities of daily life.

2.5.1 Sensor Evaluation

Because the *BLE-ECG Stamp* sensor developed at the *Max-Schaldach-endowed professorship for Biomedical Engineering* is a new type of hardware and at the time of this thesis has not been used to conduct a study yet, it is be evaluated here with a comparable *Bluetooth* ECG sensor. As gold standard hardware, a *Shimmer* (Shimmer Inc., Dublin, Ireland) ECG sensor was used. It has already been used in several publications (e.g. [Gra12], [Kug12], [Ric14]) and hence provides a validated ground truth.

Five male subjects participated in this study. The age of the participants was 22.2 ± 1.3 years (mean ± standard deviation). All subjects were healthy. Their suitability for participating in the study was assured by means of the Physical Activity Readiness Questionnaire (PAR-Q) [Tho92]. The *PAR-Q* allows the participants to check themselves before performing physical activity. It is used to sort out the individuals for whom physical activity might be inappropriate and who are suggested to consult a physician before. The questions of the *PAR-Q* are listed in Appendix B.1. All subjects participated in this study passed the *PAR-Q* and signed a written consent form about their participation.

The data was acquired at the lab facilities of the Digital Sports Group (Erlangen, Germany). All subjects were asked to perform an exercise on a stationary bicycle. Each sequence started with two minutes of resting, followed by five minutes of cycling. The first two minutes and the last two minutes of the cycling sequence were performed with a lower intensity, with one minute of heavier cycling in between. Another two minutes of resting concluded the protocol. The study protocol is also described in Appendix B.2.

The electrodes of the *Shimmer* sensor were placed in the same way as the electrodes of the *BLE-ECG Stamp* (as seen in Figure 2.20) and the same lead was used for data acquisition. The data acquired by the *BLE-ECG Stamp* is processed on-line using the *DailyMonitor* mode and the results of the data processing as well as the raw signal are stored on the external storage of the mobile phone. On the other hand, the data acquired by the *Shimmer* sensor was transmitted to a computer via *Bluetooth* where it was saved in separate CSV files for each recording. These CSV files are then transferred to the mobile phone and imported into the *DailyHeart* application by using the *simulation mode*. The course of heart rate during this measurement was recorded and exported to MATLAB® (The Mathworks, Inc., Natick, MA, USA) for further analysis.

This setup provides the possibility to compare the data processing results between the data acquired by the two different sensors. Because the intention of the application is to monitor the user's cardiac function during daily life, the parameter that is evaluated is

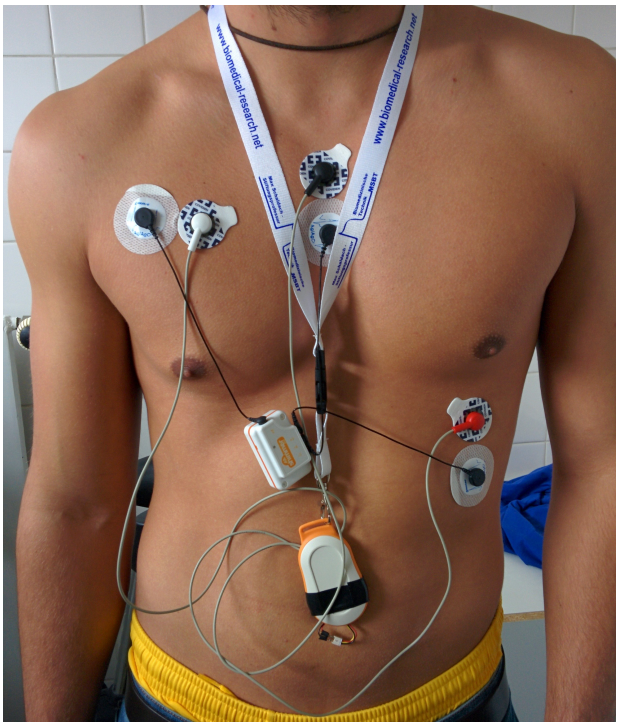


Figure 2.20: Setup for sensor evaluation. Both sensor systems were attached to the body. The leads were positioned according to Lead II of Einthoven's Triangle [Con03].

the course of heart rate over time.

2.5.2 App Evaluation

A major aspect in the development progress of *DailyHeart* was to provide an application that is made for daily use and does not disturb the user during everyday tasks. Therefore an evaluation of the application had to be performed in order to test the usability in daily life.

Ten subjects – five male and five female subjects – participated in this study. The age of the participants was 21.0 ± 1.7 years (mean ± standard deviation). All subjects were healthy and gave their written consent about participating in this study. The five male subject also participated in the quantitative sensor evaluation.

The study was performed at the Central Institute of Healthcare Engineering (ZiMT) (Erlangen, Germany). All subjects were asked to wear the *BLE-ECG Stamp* with the default electrode placement and a Nexus 5 (LG Electronics, Seoul, South Korea) mobile phone running on Android software version 5.0.1 "Lollipop". Furthermore they wore a LG G Watch (LG Electronics, Seoul, South Korea) and a Google Glass (Google Inc.,

Mountain View, CA, USA). The participants were instructed to test the usability of the wearables and all features of the *DailyHeart* application for a duration of one hour. During this session they were asked to perform daily life activities, such as sitting at the office table, answering the telephone, checking the mailbox, going for a five-minute-walk, and a one-minute-run. After the testing phase the subjects answered a questionnaire that can be found in Appendix B.4.

Chapter 3

Results

The last chapter described the methods used for the development of the *DailyHeart* application. Therefore, the concept of the *BLE-ECG Stamp* sensor device for measuring the user's ECG was introduced, followed by an algorithm that provides a real-time heart beat detection and classification on mobile phones. Finally, the features of *DailyHeart* were presented. The results of this work as well as the results of the evaluation are outlined in the following chapter.

3.1 DailyHeart Results: Main Contributions

The *DailyHeart* application developed within this work is able to perform real-time ECG analysis on mobile phones in different modes with a special focus on the usability in daily life without being disruptive or intrusive. The following are the main modes of the application:

- **CurrentHR mode**: Perform a snapshot measurement of the user's current heart rate and display the result.
- **AnalyzeECG mode**: Analyze the user's ECG within a short time interval and display the result.
- **DailyMonitor mode**: Continuously measure the user's ECG and frequently inform him about the heart status.
- **DailySimulate mode**: Simulate prerecorded ECG data either from this application or from the *MIT-BIH Arrhythmia Database* and treat the simulation as live data.

Additionally this application is compatible with state-of-the-art wearable systems like *Android Wear*-based smartwatches and *Google Glass* with the purpose of facilitating the

usability in daily life. Furthermore, it is possible to store the acquired data into a *SQLite database* as well as upload it to the *Google Fit* platform.

3.2 Evaluation Results

3.2.1 Sensor Evaluation

The results of this evaluation for each subjects are listed in Table 3.1. An example plot of the heart rate for both systems during the evaluation session is shown in Figure 3.1.

Table 3.1: Results of the Sensor Evaluation. The results are listed per subject as well as the mean results. *Avg. HR = Average Heart Rate during session*, *MAE = Mean Average Error*, *SD = Standard Deviation*, *CCC = Cross Correlation Coefficient*

	Avg. HR *BLE-ECG Stamp* (bpm)	**Avg. HR *Shimmer* (bpm)**	**MAE (bpm)**	**SD (bpm)**	**CCC**
Subject 1	101.4	101.5	2.43	2.91	0.957
Subject 2	97.0	98.2	6.54	6.58	0.920
Subject 3	101.6	102.4	3.85	3.45	0.912
Subject 4	98.7	99.3	5.64	4.29	0.913
Subject 5	119.9	118.7	5.68	5.05	0.906
Mean	103.7	104.0	4.83	4.46	0.922

The evaluation of the *BLE-ECG Stamp* reveals a cross correlation coefficient of 0.922 with a mean absolute error of 4.83 bpm and a standard deviation of 4.46 bpm. Additionally, the results show that the average heart rate over the complete session is almost identical.

3.2.2 App Evaluation

All subjects that participated in the study to evaluate the *DailyHeart* application answered the provided questionnaire which can be seen in Appendix B.4. The questions (4) – (11) have been answered on a scale from 1 to 5, where 1 represents the most negative answer and 5 the most positive answer. The mean of the answers and their standard deviation

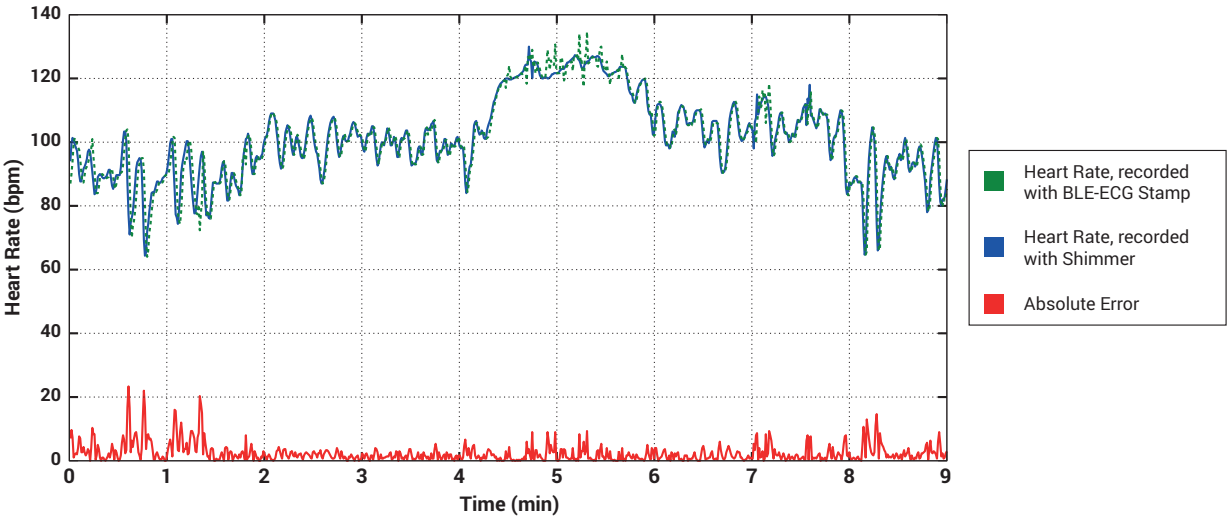
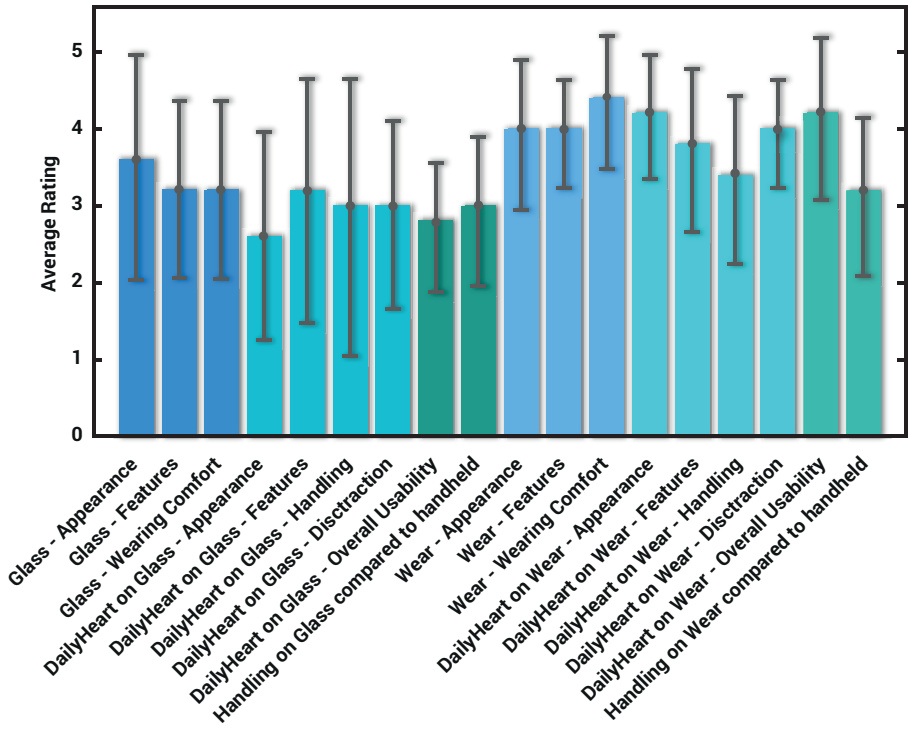


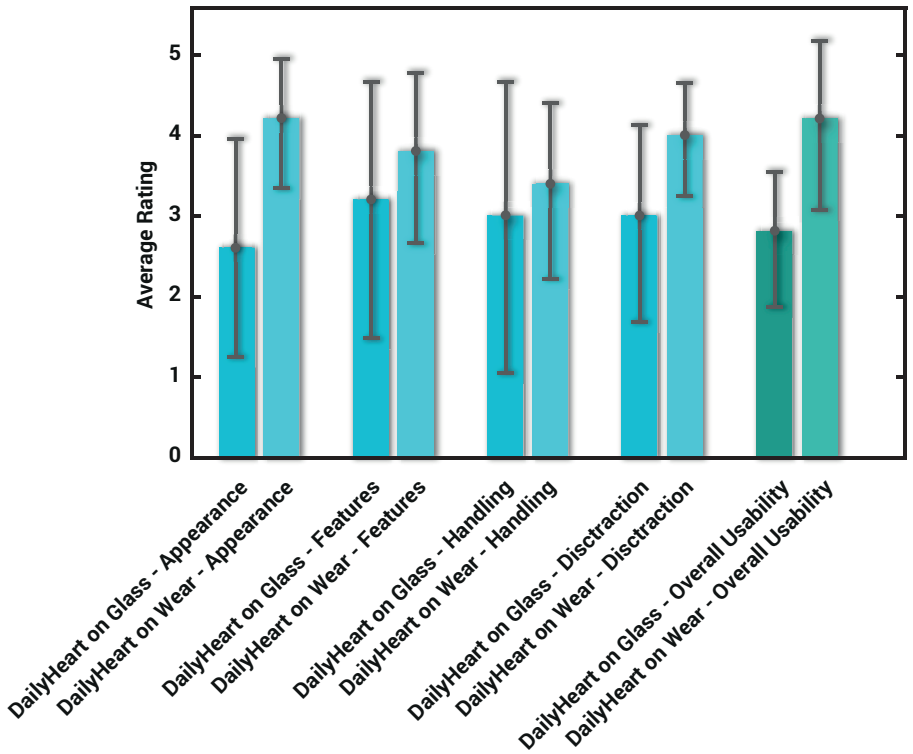
Figure 3.1: Course of Heart Rate. The course of heart rate for Subject 1, recorded with both sensors during the evaluation session;***green line:*** course of heart rate, recorded with the *BLE-ECG Stamp*; ***blue, dashed line:*** course of heart rate, recorded with the *Shimmer* sensor; ***red line:*** absolute error between *BLE-ECG Stamp* and *Shimmer*

are shown in Figure 3.2a. The figure clearly shows the ratings are higher for *Android Wear* than *Google Glass* throughout the entire evaluation. Furthermore, the standard deviation is lower as well. This becomes even more visible in a direct comparison of the ratings for both wearables in Figure 3.2b. The following results have also been found:

- **80%** of the participants have already heard of Google Glass and Android Wear, but just **20%** have heard of *Augmented Reality*.
- **30%** of the participants are afraid that Google might misuse the health data stored in *Google Fit*.
- **Every** participant would use the *DailyHeart* application on a handheld in combination with a smartwatch to monitor its cardiac function. However, no one would prefer the handheld application in combination with Glass.



(a) Results of questions (4) - (11)



(b) Results of questions in direct comparison between Glass and Wear

Figure 3.2: Results of DailyHeart Evaluation. The bar charts present the answers to the questionnaire that was filled out by the participants of the study, with 1 being the most negative answer and 5 the most positive answer.

Chapter 4

Discussion

The methods proposed in this work were used for the development of an application that incorporates novel human computer interaction principles and wearable technology for cardiac feedback. The primary goal was to design this application to be unobtrusive, easy to use and suitable for daily life.

The following chapter discusses the feasibility of the presented solution regarding each component given that each component cooperates together and shapes the complete system.

4.1 Data Acquisition

The data acquisition was performed with a newly developed sensor system from the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). It uses components like the MSP430 microcontroller and the CC2541 system-on-chip solution that are both designed with respect to energy constraints and thus enable a long-term monitoring without having to charge the device multiple times a day. This energy saving aspect is enhanced by the *Bluetooth Low Energy* communication standard which provides a considerably reduced power consumption compared to standard *Bluetooth*. With the possibility of performing a heart beat classification on the chip, the power consumption could be reduced dramatically.

Five healthy male subjects have been asked to participate in a study that involves performing a physical exercise on a stationary bicycle in order to validate the quality of the newly developed sensor device. They have been asked to wear both sensor systems simultaneously and cycle in a predefined sequence of different intensities as

well as resting before and after the exercise. This study design was chosen to validate the correctness of the sensor in a setup that is known to be challenging due to motion artifacts and friction. The results show a mean absolute error of 4.83 bpm at a standard deviation of 4.46 bpm of the *BLE-ECG Stamp* compared to the *Shimmer* setup. The high heart rate cross correlation coefficient of 0.922 proves that the *BLE-ECG Stamp* provides a solid solution for a wearable sensor system that is able to acquire data for continuous cardiac monitoring.

However, the standard deviation of 4.46 bpm reveals the key deficit of the *BLE-ECG Stamp*. The artifacts caused by body motion, friction of clothes, and lead movements manifest in a distortion of the ECG signal, which is a common problem for wearable body sensors. In comparison to the *BLE-ECG Stamp*, the *Shimmer* sensor uses shorter leads, which can be a possible reason for the lower noise level of the ECG signal acquired by the *Shimmer*. This can be especially observed during the cycling periods that were performed at a higher intensity. An approach to tackle this problem might be the usage of shorter leads that are also more tightened to the body, as well as algorithms for reducing motion artifacts. Furthermore, a new wearable sensor is in working progress, which uses a digital ECG amplifier and may result in an improved signal quality.

Nonetheless, the sensor evaluation proved the quality of the sampled ECG signal to be sufficient. The 12-bit resolution of the ADC provides a solid basis for the following steps of the processing pipeline, the heart beat detection and classification.

4.2 Data Processing

The data processing was performed on a mobile phone and uses an algorithm for a real-time heart beat classification and arrhythmia detection in an ECG signal. This algorithm provides high detection rates for QRS complexes which is an important basis for the correct calculation of the user's heart rate. The implemented algorithm offers a high sensitivity and a low number of false negatives when tested with records from the *MIT-BIH Arrhythmia Database* and the *MIT-BIH Supraventricular Database* [Gra12]. Live operation during the sensor evaluation was tested as well and was working as expected, providing a continuously monitored ECG signal with no abnormally detected beats. The algorithm ran in real-time on the testing mobile device but created high computational load which could be observed by a slight warming of the mobile phone.

However, the template selection process of the algorithm relies on the first ten beats

having high quality. Hence, poorly conditioned templates due to noisy data at the beginning of the record heavily decrease the quality of the algorithm and lead to false negative or false positive decisions. This could be improved in future work so that a false classification of the user's heart status is reduced. The application was designed in a modular way and due to the portability of the code, as it is mostly written in Java, the class containing the data processing can easily be replaced as long as the results of the algorithm are formatted in a way the application can handle.

Nevertheless, this algorithm is an essential part of the present work and guarantees a solid cardiac monitoring with a minor constraint regarding clinical aspects.

4.3 Data Visualization

This section discusses the features of the *DailyHeart* application, with respect to the *Bluetooth* communication, the user interface, and the wearable integration.

4.3.1 Sensor communication

The communication with the *Bluetooth LE* sensor is based on `Handler` and is entirely performed in the background. The device manager starts scanning for available devices and discovers the available services on the devices that were found. If an *EcgService* was found, this service will be enabled and the sensor is able to start streaming. The user is involved in this process only for starting and stopping the connection (by pressing the *Floating action button*) and is informed about connection updates by the `StatusBar` and the `ProgressDialog`.

Currently, the application is designed only to communicate with that particular kind of sensor device since it uses the *EcgService* for streaming data, which was defined for proprietary use. Services for streaming raw data have not been proposed by the Bluetooth SIG yet and are not in the proper sense of the *Bluetooth LE* guidelines. However the Bluetooth SIG already provides services for transmitting physiological data, such as a *Heart Rate service*. This service only transmits the heart rate and therefore requires data processing on the sensor device, but is already used by several sensors. The compatibility of these systems could be integrated in future work.

The handler model was chosen due to the easy implementation and comprehensibility, but requires a `Handler` instance for every class that should receive messages. This should be replaced in future work by using *local broadcasts* or by implementing an API that

handles the communication between device and sensor and delivers results in callback methods.

4.3.2 User Interface

DailyHeart is an application that was designed and implemented following the latest guidelines for the Android operating system proposed by *Google*. The colors used in this application were chosen according to a color palette proposed by Android. Combining a discrete primary color with a bright accent color and keeping this consistency throughout the application ensures a consistent and distinctive interface. Reducing the activity to the bare essentials provides clarity and prevents unnecessary loss of time getting used to the handling of the application. Using the navigation drawer for top-level navigation allows the user to quickly jump between activities which don't have direct relationships between each other. Furthermore, it is already known to the user as it is a common pattern and can be found in other applications, especially since the release of the new *material design* principles. By incorporating meaningful transitions between parent activities and their child, the structure of the application is emphasized and facilitates the navigation below top-level. *DailyHeart* requests access to the user's *Google+* profile to display its name in the header of the navigation drawer. Future work could use more profile information for creating more user-specific profiles and store this information into the database.

The *SQLite database* developed for *DailyHeart* stores the acquired data with additional information. The database is accessed by the `HistoryActvity`, that displays all saved entries in a list. The current implementation retrieves all data from the database without any filtering options. By using more detailed queries in future work that are compiled against the database, results can be narrowed to specific record types or time intervals. The `HistoryActivity` is based on a `RecyclerView`[1], which is a more advanced and flexible version of a `ListView`[2]. It uses a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views to be displayed simultaneously, which makes is suitable for displaying the entries of the *DailyHeart database*. It furthermore provides more flexibility for data collections that change at runtime and therefore would allow to dynamically delete database entries by tapping on the corresponding view in the activity.

[1] http://developer.android.com/reference/android/support/v7/widget/RecyclerView.html
[2] http://developer.android.com/reference/android/widget/ListView.html

4.3.3 Wearable Integration

The intention of this thesis was to incorporate novel human computer interaction concepts like Google Glass and smartwatches with an application for cardiac feedback. Android Wear provides several methods for displaying information on the display of the watch. When a smartwatch is connected to an Android handheld (phone or tablet), the handheld automatically shares the notifications with the wearable. By adding wearable-specific functionality to the notifications it provides a better user experience and facilitates controlling the application from the wearable. Adding voice capabilities is an important part of the wearable experience, since it allows the user to interact with the application hands-free and quickly. This functionality is enhanced by implementing a wearable application. The *Messaging API* and the *Data API* use Bluetooth communication layers for sending messages and syncing data items between handheld and wearable and thus provide a possibility for cardiac monitoring in daily life. The release of *Android Wear* software version 5.0 "Lollipop" (API level 21) in December 2014 introduced a new API that supports creating custom watch faces. It can be used to show contextual information to the user, but requires a careful blending of data and visual elements that is able to inform the user without requiring too close attention. Future work could implement a custom watch face for displaying cardiac and time information simultaneously.

The original intention of Google Glass was to present a standalone wearable system since it has enough computational resources and sensor technology to work independently from a handheld device. However, excessive usage of Google Glass rapidly leads to overheating and significant battery loss. It even causes the Glass to inform the user is has to cool down in order to work properly. Therefore it is not recommended to perform complex calculations on Glass, which makes a Glassware version of *DailyHeart* not feasible. Before the introduction of *Notification Sync* on Glass with software version *XE22* it was not possible to synchronize data between Glass and the handheld device without a using the web-based *Mirror API* as a detour. Unlike Android Wear, the Google Glass operating system did not provide an API for accessing Bluetooth data layers in order to establish a communication, although the *MyGlass* companion app for handhelds uses that kind of communication. However, *Notification Sync* for Google Glass approaches the functionality of Android Wear and opens new possibilities for creating a better user experience.

The results of the app evaluation prove that the mean rating of the usability of smartwatches (4.2) is higher than the Google Glass rating (2.8). The optical head-

mounted display is placed in the top right corner of the human field of vision, which is known as the least comfortable area. The reason behind this is making the display as unobtrusive as possible as the most comfortable areas are straight right in front and below the eye. Due to that, staring at the display for a longer period of time is counterintuitive and causes discomfort in the eye muscles [Smi14].

In comparison, wristwatches have been worn for a long time by a major part of the population and are therefore an integral part in daily life. Since Glass is an entirely new technology, smartwatches show higher public acceptance and lower habituation periods. Concerns against Google Glass have been raised by various sources regarding the intrusion of privacy due to the possibility of recording people in public without their awareness and their permission [Mar13] [Art13]. The device has already been banned in several facilities like casinos, movie theaters, bars, and several companies because of the violation of privacy and law. This public controversy about Google Glass can even be observed in this study, since the standard deviation of the questions about Google Glass is clearly higher than the standard deviation of the same questions about smartwatches.

Privacy concerns do not only exist because of the camera integrated in Google Glass, but moreover about the privacy policy of Google in general. 30% of all participants are afraid that their health data that will be stored in the Google Fitness Store could be misused for several purposes. Google listed these key principles for a responsible use of Google Fit by developers[Goo14c]:

- Always clearly explain to the user **what** data will be collected and **why**.
- Honor user requests to **delete** their data.
- Do not use Google Fit APIs for **non-fitness purposes**, such as storing medical or biometric data, or using data for advertising.

Especially the last principle is important for the present work. Uploading data to Google Fit is an optional feature that does not have to be made use of by every user. The intention is to upload only their heart rate if they want – it is not intended for storing all collected cardiac features on Google servers. This data can be stored in the internal database of *DailyHeart*. However, future work clearly has to improve the overall security of this application.

Chapter 5

Conclusion and Outlook

This work presents a novel application for cardiac monitoring. Although similar applications have already been proposed by several research groups, none of them has made use of the benefits the growing market of wearables offers.

DailyHeart allows the user to unobtrusively check his cardiac functions throughout the day. Several modes of recording the ECG signal are included within the application that range from measuring the current heart rate to monitoring the ECG for an unspecified time. Storing the acquired data in the internal database and the *Google Fit* platform allows a subsequent supervision for the user as well as for physicians. The combination of these core features from the smartphone application with the new possibilities of the highly emerging market of wearables presents a solution that is suitable for everyday usage. The benefits of wearables are incorporated in this work for providing the user a hands-free interaction over voice commands or an interaction between the user and the wearable without having to pull the smartphone out of the pocket.

The evaluation of the *BLE-ECG Stamp* shows an mean absolute error of 4.83 bpm with a standard deviation of 4.46 bpm and therefore proves that this sensor is a specially designed alternative compared to other sensors, such as *Shimmer*, but furthermore features the support of *Bluetooth Low Energy* as well as performing a preprocessing on the sensor itself. However, it also points out a possible approach for future work. The *BLE-ECG Stamp* can be improved by designing it more robust with respect to motion artifacts. Furthermore, the battery runtime can be significantly increased if a heart beat classification would be performed on the *BLE-ECG Stamp* itself instead of streaming the raw signal. It is also feasible for *DailyHeart* to provide support for other heart rate and ECG sensors in order to make the application usable for wider public.

The results from the app evaluation show that the goal of this work was fulfilled and the developed system for cardiac monitoring is usable for daily life. The average usability rating for *DailyHeart* on *Android Wear* was 4.2/5 compared to 2.8/5 for *DailyHeart* on *Google Glass*. This difference could be observed throughout all questions. However, the difference in the rating between *Glass* and *Android Wear* show that the public acceptance of smartwatches is inevitably higher than the acceptance of *Google Glass*. The higher standard deviation of the *Glass*-related questions emphasizes the high controversy that the device was faced with. An important issue that has to be addressed in future work and may not be neglected are privacy aspects of the acquired health data. It must be ensured that it is impossible to access this data for third parties.

Another possible approach for future work would be the extension of *DailyHeart* in order to enhance the range of features. The information obtained from the ECG sensor could be combined with additional data from GPS and accelerometers for a *DailyTrain* mode specialized for various types of physical activity such as running or biking. By including GPS data, the user gets feedback about location, speed, and altitude [Ric14]. The signal obtained from an accelerometer can be used for an energy expenditure estimation during physical activity [Bou94]. Furthermore, the clinical aspect of *DailyHeart* could be enhanced by performing further analysis regarding the heart rate variability. It has been proved that a decreased heart rate variability is associated with congestive heart failure, diabetic neuropathy, depression, or pathologies in the autonomous nervous system [Por09]. Therefore, incorporating the analysis of heart rate variability can be an important aspect for the improvement of cardiac monitoring and the prevention of diseases [Tha10].

Appendix A

Patents

A.1 Heart Monitoring System Usable with a Smartphone or Computer

Publication Number	US8509882 (B2)
Date of Publication	Aug. 13, 2013
Inventor(s)	**David Albert**, Oklahoma City, OK (US); **Bruce Richard Satchwell**, Carrara (AU); **Kim Norman Barnett**, Mt. Tamborine (AU)
Assignee	**AliveCor, Inc.**, San Francisco, CA (US)
Abstract	A personal monitoring device has a sensor assembly configured to sense physiological signals upon contact with a user's skin. The sensor assembly produces electrical signals representing the sensed physiological signals. A converter assembly, integrated with, and electrically connected to the sensor assembly, converts the electrical signals generated by the sensor assembly to a frequency modulated physiological audio signal having a carrier frequency in the range of from about 6 kHz to about 20 kHz.

A.2 Wearable Device for Continuous Cardiac Monitoring

Publication Number US20130338460 (A1)

Date of Publication Dec. 19, 2013

Inventor(s) **David Da He**, Cambridge, MA (US);
Charles G. Sodini, Belmont, MA (US);
Eric Steven Winokur, Danvers, MA (US)

Assignee **David Da He**, Cambridge, MA (US);
Charles G. Sodini, Belmont, MA (US);
Eric Steven Winokur, Danvers, MA (US)

Abstract A physiological monitor for measuring a pulsatile motion signal (MoCG) that is delayed from, but at the same rate as, the heartbeat of a user. In one embodiment, the system includes a housing configured to be worn on the body of a user; at least one MoCG sensor, within the housing, that measures a pulsatile motion signal (MoCG) that is delayed from, but at the same rate, the heartbeat of the user; and at least one data processor that calculates, solely based on an output of at least one MoCG sensor, at least one of (i) heart rate (HR) and activity level for the user, and (ii) respiratory rate (RR), stroke volume (SV), and cardiac output (CO) for the user. In another embodiment, at least one data processor is within the housing.

A.3 Wearable Cardiac Monitor

Publication Number US20140100432 (A1)

Date of Publication Apr. 10, 2014

Inventor(s) **George Stefan Golda**, El Granada, CA (US)
Daniel Van Zandt Moyer, Menlo Park, CA (US);
Mark P. Marriott, Palo Alto, CA (US);
Sam Eletr, Paris (FR);
Bruce O'Neil, San Francisco, CA (US)

Assignee **George Stefan Golda**, El Granada, CA (US);
Daniel Van Zandt Moyer, Menlo Park, CA (US);
Mark P. Marriott, Palo Alto, CA (US);
Sam Eletr, Paris (FR);
Bruce O'Neil, San Francisco, CA (US)

Abstract Systems, methods and devices for reducing noise in cardiac monitoring including wearable monitoring devices having at least one electrode for cardiac monitoring; in some implementations, the wearable device using a composite adhesive having at least one conductive portion applied adjacent the electrode; and, in some implementations, including circuitry adaptations for the at least one electrode to act as a proxy driven right leg electrode.

A.4 System for Cardiac Arrhythmia Detection and Characterization

Publication Number US8233972 (B2)

Date of Publication Jul. 31, 2012

Inventor(s) **Hongxuan Zhang**, Palatine, Il (US)

Assignee **Siemens Medical Solutions USA, Inc.**, Malvern, PA (US)

Abstract A system for heart performance characterization and abnormality detection includes an interface that receives a signal representing electrical activity of a patient heart occurring during individual heart cycles of multiple sequential heart cycles. A signal processor decomposes the received signal into multiple signals comprising a heart cycle signal primary wave and one or more heart cycle signal wavelets occurring at corresponding successively higher frequencies. The signal processor determines multiple amplitude representative values of the heart cycle signal primary wave and the one or more heart cycle signal wavelets. A comparator compares the multiple amplitude representative values with corresponding multiple predetermined threshold values to provide comparison indicators. A patient monitor in response to the comparison indicators indicating at least one of the amplitude representative values exceeds a respective predetermined threshold value, generates an alert message associated with the respective predetermined threshold.

A.5 Method and System for Detection of Cardiac Arrhythmia

Publication Number US7120485 (B2)

Date of Publication Oct. 10, 2006

Inventor(s) **Leon Glass**, Montreal (CA);
Katsumi Tateno, Kitakyushu (JP)

Assignee **Medtronic, Inc.**, Minneapolis, MN (US)

Abstract There are many different serious cardiac arrhythmias. The present invention uses measurements of RR intervals (inter beat intervals) to detect, in particular but not exclusively, atrial fibrillation of a patient. Atrial fibrillation is a serious ailment in which the heart beat is generally rapid and irregular. Probability density histograms of ΔRRs (difference between two successive RR intervals) collected during atrial fibrillation of a plurality of subjects are used as a template for the detection of atrial fibrillation. In one implementation, there are 16 standard probability density ΔRRs histograms every 50 ms of mean RR interval of a certain number of beats, Where the mean RR interval ranges from 350 ms to 1149 ms. Similarity between the standard probability density histograms and a test density probability histogram of ΔRRs of a patient is estimated by the Kolmogorov-Smimov test. If the test density histogram is not significantly different from the standard density histogram, atrial fibrillation is detected.

A.6 Real time QRS duration measurement in electrocardiogram

Publication Number	EP2676604 (A1)
Date of Publication	Dec. 25, 2013
Inventor(s)	**Vasile Zoicas**, La Gaude (FR)
Assignee	**Texas Instruments France**, Villeneuve Loubet Cedex (FR)
Abstract	ECG data may be processed in a mobile device by receiving a stream of filtered ECG data samples comprising PQRST pattern cycles. An R point of a PQRST pattern in the filtered ECG data is determined. A portion of samples is selected from the filtered ECG data surrounding the R point. QRS duration of the PQRST pattern is then determined by processing the selected portion of samples using an application pro- gram executed by the mobile device.

A.7 Wearable Vital Sign Monitoring System

Publication Number US6102856

Date of Publication Aug. 15, 2000

Inventor(s) **Clarence P. Groff**, Columbia, SC (US);
Paul L. Mulvaney, Columbia, SC (US)

Assignee **Clarence P. Groff**, Columbia, SC (US);
Paul L. Mulvaney, Columbia, SC (US)

Abstract The present invention is a wearable vital sign monitor. The device is worn preferably about the chest just below the pectoral muscles and monitors at least the following: ECG data, respiration rate, oxygen uptake, pulse rate, and body temperature. These data are collected and analyzed to determine if there is a deviation from the wearer's normal condition, which the device learns. If there is, the device sends a signal to a remote central facility to be received by an attendant who is capable of ascertaining whether the abnormal condition is in fact a warning sign of an adverse health condition. If necessary, the attendant can communicate by voice with the wearer. Optionally, the attendant can locate the wearer, assuming that the wearer is unable to speak, using a ground positioning satellite (GPS) locating system. Finally, the device is capable of producing periodic reports of the recorded data.

A.8 Adaptive data transfer using bluetooth

Publication Number	US20140273858 (A1)
Date of Publication	Sept. 18, 2014
Inventor(s)	**Heiko Gernot Albert Panther**, San Francisco, CA (US) **James Park**, Berkeley, CA (US)
Assignee	**Fitbit, Inc.**, San Francisco
Abstract	Biometric monitoring devices, including various technologies that may be implemented in such devices, are discussed herein. Additionally, techniques, systems, and apparatuses are discussed herein for utilizing two different Bluetooth communications interfaces, one that provides Bluetooth classic (base rate/enhanced data rate) communications functionality and one that provides Bluetooth low-energy communications functionality, in a common device. The techniques, systems, and apparatuses may elect to use a particular Bluetooth interface based on various criteria.

Appendix B

Evaluation

B.1 Physical Activity Readiness Questionnaire (PAR-Q)

For most people, physical activity should not pose any problem or hazard. The PAR-Q is designed to identify the small number of adults for whom physical activity might be inappropriate or those who should seek medical advice concerning the type of activity most suitable for them.

1. Do you have a bone or joint problem such as arthritis, which has been aggravated exercise or might be made worse with exercise? **(yes/no)**
2. Do you have or ever suffered a heart condition? **(yes/no)**
3. Have you ever felt pain in your chest when you do physical exercise? **(yes/no)**
4. Is your doctor currently prescribing you drugs or medication? **(yes/no)**
5. Have you ever suffered from shortness of breath at rest or with mild exercise? **(yes/no)**
6. Do you ever feel faint, have spells of dizziness or have ever lost consciousness? **(yes/no)**
7. Do you know of any other reason why you should not participate in a program of physical activity? **(yes/no)**

If you answered NO to all questions:
If you answered the PAR-Q honestly and accurately, you have reasonable assurance of your present suitability for participating in the study about activity recognition.

B.2 Sensor Evaluation – Study Protocol

Location: Pattern Recognition Lab, Digital Sports Group
Haberstr. 2, 91058 Erlangen

Activity: Cycling

Duration: 9 min

Sequence: Predefined

1. *Resting* (2 min)
2. *Cycling* – low intensity (2 min)
3. *Cycling* – high intensity (1 min)
4. *Cycling* – low intensity (2 min)
5. *Resting* (2 min)

B.3 App Evaluation – Study Protocol

Location: Central Institute of Healthcare Engineering (ZiMT)
Henkestr. 91, 91052 Erlangen

Activity: Simulation of everyday activities

Duration: 1 hr

Sequence: Undefined, but contains the following activities:

- Sitting at the office table
- Answering the telephone
- Kitchen work (cooking coffee, loading the dishwasher, etc.)
- Checking the mailbox
- Walking (5 min)
- Running (1 min)

B.4 App Evaluation – Questionnaire

Questions (1) - (3) should be answered with "yes" or"no"

1. Have you heard of **Google Glass** before?
2. Have you heard of **smartwatches** before?
3. Have you heard of **Augmented Reality** before?

*Questions (4) - (11) should be answered on a scale between **1** and **5**, where 1 is the most negative answer and 5 the most positive answer*

4. How **satisfied** are you with the following *Google Glass* features?
   (a) **Appearance** of *Google Glass*
   (b) **Features** of *Google Glass*
   (c) **Handling** of *Google Glass*
5. How **satisfied** are you with the following features of *DailyHeart for Google Glass*?
   (a) **Appearance** of *DailyHeart for Google Glass*
   (b) **Features** of *DailyHeart for Google Glass*
   (c) **Handling** of *DailyHeart for Google Glass*
   (d) Does *DailyHeart for Google Glass* **distract** you from your normal activities?
6. How is the **overall usability** of *DailyHeart for Google Glass* in daily life?
7. How is the **handling** of *DailyHeart for Google Glass* compared to the handling of *DailyHeart for mobile phones*?
   (1 = much more difficult, 2 = slightly more difficult, 3 = no difference, 4 = slightly easier, 5 = much easier)
8. How **satisfied** are you with the following *Android Wear* features?
   (a) **Appearance** of *Android Wear*
   (b) **Features** of *Android Wear*
   (c) **Handling** of *Android Wear*
9. How **satisfied** are you with the following features of *DailyHeart for Android Wear*?
   (a) **Appearance** of *DailyHeart for Android Wear*

(b) **Features** of *DailyHeart for Android Wear*
(c) **Handling** of *DailyHeart for Android Wear*
(d) Does *DailyHeart for Android Wear* **distract** you from your normal activities?

10. How is the **overall usability** of *DailyHeart for Android Wear* in daily life?

11. How is the **handling** of *DailyHeart for Android Wear* compared to the handling of *DailyHeart for mobile phones*?
(1 = much more difficult, 2 = slightly more difficult, 3 = no difference, 4 = slightly easier, 5 = much easier)

Questions (12) - (14) should be answered with „yes“ or „no“

12. Are you afraid that Google might **misuse your health data**?

13. Would you use *DailyHeart* for **monitoring your cardiac functions**?

14. If yes, which setup would you prefer?
    - DailyHeart on smartphone
    - DailyHeart on smartphone + Google Glass
    - DailyHeart on smartphone + Android Wear

Appendix C

Source Code

C.1 Handler Messages

This section lists the most important messages and their meaning that were used in the implementation of the *DailyHeart* application. All constants of the application were stored in its separate interface called `Constants`.

Listing C.1: Handler messages

```
1  public interface Constants {
2      // Message indicating the service is about to be killed
3      int MSG_SERVICE_DEAD = 201;
4
5      // Message indicating a new BLE-ECG device has been found
6      int MSG_NEW_ECG_DEVICE = 202;
7
8      // Message indicating a successful connection to the sensor
9      int MSG_DEVICE_CONNECTED = 204;
10
11     // Message indicating a connection loss to the sensor
12     int MSG_CONNECTION_LOSS = 205;
13
14     // Message indicating the connection to the sensor has ended
15     int MSG_CONNECTION_ENDED = 211;
16
17     // Message indicating that the BLE scan has returned no
```

```
        result
18      int MSG_NO_DEVICE_AVAILABLE = 212;
19
20      // Message to start scanning for available BLE-ECG devices
21      int MSG_START_SCAN = 408;
22
23      // Message indicating that the BLE-ECG device or the
            simulator is ready and begins with data streaming
24      int MSG_START_STREAMING = 410;
25
26      // Message indicating that the simulation file is done
            loading and ready for streaming
27      int MSG_SIM_READY = 501;
28
29      // Message indicating a empty simulation file
30      int MSG_SIM_EMPTY_FILE = 502;
31
32      // Message indicating the end of the simulation file
33      int MSG_SIM_FILE_ENDED = 503;
34
35      // Message delivering new processed data to the UI
36      int MSG_PROCESSED_DATA_AVAILABLE = 605;
37
38      // Message indicating the current heart beat has been
            segmented
39      int MSG_SEGMENTATION_FINISHED = 606;
40
41      // Messages delivering new data to the BLEService
42      int MSG_DAILYHEART_LIVE_DATA = 997;
43      int MSG_DAILYHEART_SIM_DATA = 998;
44      int MSG_MITBIH_SIM_DATA = 999;
45  }
```

C.2 BluetoothGattCallback

This section outlines the implementation of the abstract `BluetoothGattCallback` class in the `BLEDeviceManager`, as it represents one of the most important components for the *Bluetooth LE* connection.

Listing C.2: BluetoothGattCallback

```
1  private BluetoothGattCallback gattCallback = new
       BluetoothGattCallback() {
2      public void onConnectionStateChange(BluetoothGatt gatt, int
           status, int newState) {
3          if (status == BluetoothGatt.GATT_SUCCESS && newState =
               BluetoothProfile.STATE_CONNECTED) {
4              // Once successfully connected, services on the
                   device have to be discovered before their
                   characteristics can be read and written.
5              gatt.discoverServices();
6              // Notify other components that connection to
                   sensor is established
7              serviceHandler.obtainMessage(
8                  Constants.MSG_DEVICE_CONNECTED,
                       gatt.getDevice().getName()).sendToTarget();
9          } else if (status == BluetoothGatt.GATT_SUCCESS &&
               newState == BluetoothProfile.STATE_DISCONNECTED) {
10             // At any point of disconnection notify other
                   components
11             serviceHandler.sendEmptyMessage(
12                     Constants.MSG_CONNECTION_ENDED);
13             gatt.disconnect();
14         } else if (status != BluetoothGatt.GATT_SUCCESS) {
15             // If there is a failure at any stage, simply
                   disconnect
16             serviceHandler.sendEmptyMessage(
17                     Constants.MSG_CONNECTION_ENDED);
18         }
```

```
19      }
20
21      public void onServicesDiscovered(BluetoothGatt gatt, int
           status) {
22          // With services discovered, the right service is about
               to get enabled
23          if (gatt.getDevice().getName()
24                  .equals(Constants.DEVICE_NAME) {
25              ecgDevice.enableECGSensor(gatt);
26          }
27      }
28
29      public void onCharacteristicChanged(BluetoothGatt gatt,
           BluetoothGattCharacteristic characteristic) {
30          if (gatt.getDevice().getName()
31                  .equals(Constants.DEVICE_NAME) {
32              RawECGData[] data =
                   dataprocessor.handleNewData(characteristic);
33              ecgDevice.scheduleWriting(data);
34          }
35      }
36  }
```

C.3 DailyHeart Database

The following section lists all columns used in the *DailyHeart* application in order of implementation into the *SQLite database*.

Column Name in SQL	Data Type	*Description*
_id	Integer	*Id of the current row, Automatically assigned by the database*
firstname	Text	*First name of the user*
lastname	Text	*Last name of the user*
age	Integer	*Age of the user*
profileid	Integer	*Profile id of the user, either from local or from Google+*
starttime	Long	*Start time of measurement*
endtime	Long	*End time of measurement*
duration	Integer	*Duration of measurement (endtime-starttime)*
durationhours	Integer	*Duration in hours*
durationminutes	Integer	*Duration in minutes*
durationseconds	Integer	*Duration in seconds*
year	Integer	*Year the measurement was started*
month	Integer	*Month the measurement was started*
day	Integer	*Day the measurement was started*
hour	Integer	*Hour the measurement was started*
minute	Integer	*Minute the measurement was started*
averagehr	Integer	*Average heart rate during measurement*
minimumhr	Integer	*Minimum heart rate during measurement*
maximumhr	Integer	*Maximum heart rate during measurement*
averagerr	Integer	*Average RR interval during measurement*
averageqrsta	Integer	*Average QRST area during measurement*
numnormalbeats	Integer	*Number of heart beats classified as normal*
numarrbeats	Integer	*Number of heart beats classified as abnormal*
heartstatus	Integer	*Overall heart status of the measurement*
recordtype	Integer	*The record mode of the measurement*
heartmap	String	*JSON-formatted String containing the course of heart rate over time*

Glossary

ADC Analog-to-digital-converter. A device that converts a continuous-time and continuous-amplitude analog signal (a physical quantity, usually voltage) into a discrete-time and discrete-amplitude digital signal (digital numbers representing the amplitude of the quantity). 9, 62

API Application Programming Interface. A part of a program that can be accessed by other programs. 12, 19, 20, 24, 31, 41, 43–45, 50, 53, 63, 65

BSN Body Sensor Network. A wireless network of wearable computing devices, such as sensors, smart phones, smart watches, etc. 1

CPU Central processing unit. The electronic circuity within a computer that carries out the instructions of a computer program by performing speficied arithmetic, logical, control and input/output operations. 88

EMG Electromyogram. Measurement of the electrical activity produced by skeletal muscles. 3

FRAM Ferroelectric RAM. A RAM that uses a ferroelectric layer to achieve non-volatility. They are known for a low power usage, fast write performance, and a great maximum number of write-erase cycles. 9

Holter monitor A portable device that provides telemetric cardiac monitoring via ECG [Tho09]. 8, 9, 89

HRV Heart Rate Variability. The variation of heart rate between two consecutive heart beats. It is a feedback from the autonomic nervous system and is influenced by sympathetic and parasympathetic activity, as well as respiration, blood pressure, thermoregulation, etc. 3

IDE Integrated development environment.. 21

JSON JavaScript Object Notation. A data format to transmit data objects consisting of attribute-value pairs. 36

Kernel A computer program that manages the input/output requests from software and translates them into instructions for the central processing unit (CPU) and other computer components. It is a fundamental part of a computer's operating system. 19

MEMS Micro-Electro-Mechanical Systems. Miniaturized mechanical and electro-mechanical elements that are made using the technique of microfabrication. They consist of microsensors, microactutators and microelectronics. 1

OHMD Optical head-mounted display. A wearable display that can show projected images to the user. 51

RAM Random-access memory. A form of computer data storage that allows data to be read and writter in roughly the same time regardless of the order in which data items are accessed. 87

serial port A communication interface, through which information is transfered one bit at a time. 11

SoC System-on-a-chip. An integrated circuit that integrates all necessary components of a computer into a single chip. 8, 9, 11

UUID Universally unique identifier. A 128 bit value that should identify information in distributed systems uniquely. 12

wearable Miniature electronic devices for measuring, computing, and visualization purposes that are worn under, over or in clothing [Man13]. 4

List of Figures

List of Tables

Bibliography

[Abt14] F. Abtahi, A. Berndtsson, S. Abtahi, F. Seoane, K. Lindecrantz: *Development and Preliminary Evaluation of an Android-based Heart Rate Variability Biofeedback System*, in *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*, IEEE, 2014, S. 3382–3385.

[All07] J. Allen: *Photoplethysmography and its Application in Clinical Physiological Measurement*, *Physiological Measurement*, Bd. 28, Nr. 3, 2007, S. R1.

[Art13] C. Arthur: *Google Glass: Is it a threat to our privacy?*, http://www.theguardian.com/technology/2013/mar/06/google-glass-threat-to-our-privacy, 2013, accessed 2015-02-02.

[Bou94] C. V. Bouten, K. R. Westerterp, M. Verduin, J. JANSSEN: *Assessment of energy expenditure for physical activity using a triaX— aI, Age (yr)*, Bd. 23, Nr. 1.8, 1994, S. 21–27.

[Bou08] C. Boulton: *Google Open-Sources Android on Eve of G1 Launch*, http://www.eweek.com/c/a/Mobile-and-Wireless/Google-Open-Sources-Android-on-Eve-of-G1-Launch, 2008, accessed 2015-02-01.

[Car06] G. Carone, D. Costello: *Can Europe Afford to Grow Old?*, http://www.imf.org/external/pubs/ft/fandd/2006/09/carone.htm, 2006, Accessed: 2015-01-15.

[Con03] M. B. Conover: *Understanding Electrocardiography*, Elsevier Health Sciences, 2003.

[Dem13] A. Dementyev, S. Hodges, S. Taylor, J. Smith: *Power Consumption Analysis of Bluetooth Low Energy, ZigBee and ANT Sensor Nodes in a Cyclic Sleep Scenario*, in *Wireless Symposium (IWS), 2013 IEEE International*, IEEE, 2013, S. 1–4.

[DH03] E. Den Hond, H. Celis, G. Vandenhoven, E. O'Brien, J. A. Staessen, others: *Determinants of White-Coat Syndrome Assessed by Ambulatory Blood Pressure or Self-Measured Home Blood Pressure*, *Blood pressure monitoring*, Bd. 8, Nr. 1, 2003, S. 37–40.

[Eas14] J. Eason: *Android Developers Blog – Android Studio 1.0*, http://android-developers.blogspot.de/2014/12/android-studio-10.html, 2014, accessed 2015-02-01.

[Gom12] C. Gomez, J. Oller, J. Paradells: *Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-power Wireless Technology*, *Sensors*, Bd. 12, Nr. 9, 2012, S. 11734–11753.

[Goo14a] Google: *Android 5.0 Lollipop*, http://www.android.com/intl/en_us/versions/lollipop-5-0/, 2014, accessed 2015-01-27.

[Goo14b] Google: *Android: Be together. Not the same*, http://googleblog.blogspot.de/2014/10/android-be-together-not-same.html, 2014, accessed 2015-01-26.

[Goo14c] Google: *Google Fit – Platform Overview*, https://developers.google.com/fit/overview, 2014, accessed 2014-02-10.

[Goo14d] Google: *Material design - Google design guidelines*, http://www.google.com/design/spec/material-design/introduction.html, 2014, accessed 2015-01-24.

[Goo15] Google: *We are graduating from Google[x] labs*, https://plus.google.com/+GoogleGlass/posts/9uiwXY42tvc, 2015, Accessed: 2015-01-29.

[Gra12] S. Gradl, P. Kugler, C. Lohmüller, B. Eskofier: *Real-time ECG Monitoring and Arrhythmia Detection using Android-based Mobile Devices*, in *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, IEEE, 2012, S. 2452–2455.

[Her14a] J. Hernandez, Y. Li, J. M. Rehg, R. W. Picard: *BioGlass: Physiological Parameter Estimation Using a Head-mounted Wearable Device*, in *International Conference on Wireless Mobile Communication and Healthcare (MobiHealth)*, 2014.

[Her14b] J. Hernandez, R. W. Picard: *SenseGlass: Using Google Glass to Sense Daily Emotions*, in *Proceedings of the adjunct publication of the 27th annual ACM symposium on User interface software and technology*, ACM, 2014, S. 77–78.

[Kra07] V. Krasteva, I. Jekova: *QRS Template Matching for Recognition of Ventricular Ectopic Beats*, *Annals of biomedical engineering*, Bd. 35, Nr. 12, 2007, S. 2065–2076.

[Kug12] P. Kugler, U. Jensen, B. Eskofier: *Recording and Analysis of Biosignals on Mobile Devices*, *Sportinformatik 2012*, 2012, S. 182.

[Mai14] T. Maiwald: *Development of a Platform for Wearable Biosensor Networks with Bluetooth Low Energy*, *Friedrich-Alexander-Universität Erlangen-Nürnberg*, 2014.

[Mak14] M. M. Makki, G. A. Saade, A. G. Altouma, S. Al-Terkawi, A. Baobeid, R. Tafreshi: *Acquiring and Analyzing Electrocardiograms via Smartphone to Detect Cardiovascular Abnormalities*, in *Biomedical and Health Informatics (BHI), 2014 IEEE-EMBS International Conference on*, IEEE, 2014, S. 277–280.

[Man13] S. Mann: *Wearable Computing*, *The Encyclopedia of Human-Computer Interaction, 2nd Ed.*, 2013.

[Mar97] R. Mark, G. Moody: *MIT-BIH Arrhythmia Database*, *See http://ecg. mit. edu/dbinfo. html*, 1997.

[Mar13] G. Marshall: *Google Glass: Say goodbye to your privacy*, http://www.techradar.com/news/mobile-computing/google-glass-say-goodbye-to-your-privacy-1134796, 2013, accessed 2015-02-02.

[Mis] Misscurry: *Setup of a Holter Monitor*, http://upload.wikimedia.org/wikipedia/commons/9/97/Alex_CM4000.jpg, accesses 2015-01-16.

[oha07] *Open Handset Alliance – Overview*, http://www.openhandsetalliance.com/oha_overview.html, 2007, accessed 2015-02-01.

[Ore10] J. J. Oresko, Z. Jin, J. Cheng, S. Huang, Y. Sun, H. Duschl, A. C. Cheng: *A Wearable Smartphone-based Platform for Real-time Cardiovascular Disease Detection via Electrocardiogram Processing*, *Information Technology in Biomedicine, IEEE Transactions on*, Bd. 14, Nr. 3, 2010, S. 734–740.

[Pan85] J. Pan, W. J. Tompkins: *A Real-time QRS Detection Algorithm*, *Biomedical Engineering, IEEE Transactions on*, , Nr. 3, 1985, S. 230–236.

[Piz85] G. Pizzuti, S. Cifaldi, G. Nolfe: *Digital Sampling Rate and ECG Analysis*, *Journal of biomedical engineering*, Bd. 7, Nr. 3, 1985, S. 247–250.

[Por09] S. W. Porges: *The Polyvagal Theory: New Insights into Adaptive Reactions of the Autonomic Nervous System*, *Cleveland Clinic journal of medicine*, Bd. 76, Nr. Suppl 2, 2009, S. S86–S90.

[Ric14] R. Richer, P. Blank, D. Schuldhaus, B. M. Eskofier: *Real-Time ECG and EMG Analysis for Biking Using Android-Based Mobile Devices*, in *Wearable and Implantable Body Sensor Networks (BSN), 2014 11th International Conference on*, IEEE, 2014, S. 104–108.

[Riv14] J. Rivera, R. van der Meulen: *Worldwide Smartphone Sales to End Users by Operating System in 3Q14*, http://www.gartner.com/newsroom/id/2944819, 2014, Accessed: 2015-01-23.

[Sha49] C. E. Shannon: *Communication in the Presence of Noise*, *Proceedings of the IRE*, Bd. 37, Nr. 1, 1949, S. 10–21.

[Smi14] J. I. Smith: *Google's Eye Doctor Admits Glass Can Cause Pain*, http://observer.com/2014/05/googles-eye-doctor-admits-glass-can-cause-pain/, 2014, accessed 2015-02-02.

[Suh10] M.-k. Suh, L. S. Evangelista, C.-A. Chen, K. Han, J. Kang, M. K. Tu, V. Chen, A. Nahapetian, M. Sarrafzadeh: *An Automated Vital Sign Monitoring System for Congestive Heart Failure Patients*, in *Proceedings of the 1st ACM International Health Informatics Symposium*, ACM, 2010, S. 108–117.

[Sur08] B. Surawicz, T. Knilans: *Chou's electrocardiography in clinical practice: adult and pediatric*, Elsevier Health Sciences, 2008.

[Sut14] A. L. Sutton: *Cardiovascular Disorders Sourcebook*, Omnigraphics, Inc., 5th. Ausg., 2014.

[Tha10] J. F. Thayer, S. S. Yamamoto, J. F. Brosschot: *The Relationship of Autonomic Imbalance, Heart Rate Variability and Cardiovascular Disease Risk Factors*, *International journal of cardiology*, Bd. 141, Nr. 2, 2010, S. 122–131.

[Tho92] S. Thomas, J. Reading, R. J. Shephard: *Revision of the Physical Activity Readiness Questionnaire (PAR-Q).*, *Canadian journal of sport sciences*, 1992.

[Tho09] H. Thomas, T. Helms, G. Mikus, H. Katus, C. Zugck: *Telemetry in the Clinical Setting*, *Herzschrittmachertherapie and Elektrophysiologie*, Bd. 19, Nr. 3, 2009.

[Win83] B. B. Winter, J. G. Webster: *Driven-Right-Leg Circuit Design*, *Biomedical Engineering, IEEE Transactions on*, , Nr. 1, 1983, S. 62–66.

[Yan14] G.-Z. Yang: *Body Sensor Networks*, Springer, 2014.

[Yil10] T. Yilmaz, R. Foster, Y. Hao: *Detecting Vital Signs with Wearable Wireless Sensors*, *Sensors*, Bd. 10, Nr. 12, 2010, S. 10837–10862.