



Development of a mobile service for dispatch life support: With emphasis on sensor data protocol design and real time CPR signal processing

Medical technologies
Master Program Digital Healthcare
Master thesis

Name of the author:

Florian Grassinger BSc

Name of the 1st advisor:

FH-Prof. Jakob Doppler MSc

Name of the 2nd advisor:

Markus Wagner MSc

Date of delivery: 16.08.2016

Preface

This thesis is dedicated to all prospects in dispatch life support and medical emergency situations. Especially for laypersons or people who do not have much experience with medical emergency situations. Though the thesis should also help already well educated paramedics to understand more about the reanimation process and the immense possibilities to enhance it. The result of this thesis was already tested during two user studies which have been performed by Stefan Loitzl and Peter Pavlecka, who both are also partners in this project. Therefore, let's point to their respective thesis and great results, which showed, that a guided reanimation can be enhanced with a simple smartphone sensor and a website to monitor sensor readings. I would also like to thank my advisors Markus Wagner and Jakob Doppler and especially our head of studies Helmut Ritschl for supporting me during the whole process of application and general software development. Only by their support it was possible to find project partners and grow a basic idea to a full project. Especially Markus Wagner and Jakob Doppler, who helped me a lot with calculation theories and gave me a new prospect on difficult problems.

Of course also Raphael van Tulder and Heinz Novosad have to be named here and thanked for their generous support and their ideas in detecting the chest compression rate and depth in order to monitor it. They have told us, what it lacks in current emergency dispatch life support. Raphael van Tulder's ideas gave us the right direction for further development and showed us, what is urgently needed in dispatch life support situations. It is an application (small, easy to understand and lightweight), which uses a low cost sensor, to monitor the reanimation a layperson is performing, while guided by an emergency dispatcher. The emergency dispatcher is then able to give verbal instructions to the reanimating person while getting visual information from the ongoing reanimation.

Also to be mentioned are my dear friends Zhongnan and Rainer who helped me with difficult issues and gave me new directions and ideas. Generally, they always supported me and helped me to continue my work, which gave me a huge motivation.

Finally, I would like to thank my family for supporting me during the whole study and backing me up during hard times. They had patience with me and always helped in hard times as well as gave me the possibility to study the subject I like. Especially my mother Elisabeth, my grandmother Anna and my sister Teresa, who always motivated me and kept me on going and generally support me during my life. I would like to dedicate this thesis especially to my family as they are the main reason for my success. Of course I have to mention my other friends and family members too, who support me, but those described above are the most important.



Declaration

I declare that I have developed and written the enclosed Master Thesis completely by myself, and have not used sources or means without declaration in the text. Any thoughts from others or literal quotations are clearly marked. The Master Thesis was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Location

Date

Signature

Executive Summary

Keywords: cardiopulmonary resuscitation, chest compression depth & rate, acceleration-based depth estimation, acceleration-based frequency estimation

Essential problem in this regard is, that the current dispatch life support (DLS) by laypersons often lacks the appropriate treatment and technique in a guided cardiopulmonary resuscitation (CPR). For example, most laypersons who reanimate for the first time in their life, either do it too fast or too slow or generally not in the right rhythm. They are normally guided by the dispatcher or call taker on the phone, but he has only limited resources in order to review the reanimation process of the layperson. Until now the only way to review the ongoing reanimation was verbal feedback from the layperson who is guided by the dispatcher (using a metronome for counting). It is difficult for both sides to count only on the verbal feedback, when saving a life is involved.

This leads to a general wish for improvement and great potential. A possible solution is to use current smartphone technologies in order to get variables or parameters of the current reanimation. The most influential parameter is the chest compression rate (CCR), which is often inappropriate and out of rhythm. Also the chest compression depth (CCD) is very important for a reanimation. During this thesis the CCR and CCD acquisition with mobile technologies is further investigated along the following major question:

Can low-cost and mobile acceleration sensors in mobile devices (e.g. smartphones) provide high-quality, robust and sensitive data for real-time CPR signal processing and data transmission?

Mobile technologies are used nowadays that often, that nearly everyone carries some sort of smartphone with them. Particularly in this thesis a device (smartphone) is used in order to collect data and transfer it to a server, where it gets visualized and evaluated. The server is not the main part of this thesis, though it is a part of the overall project, wherefore it will be described briefly. Especially for the visualization of the collected information and feedback the server and the associated website are important.

Another important part of this thesis is the data transmission and the stability of the application, as the application should run at least ten minutes (the time the ambulance statistically needs to arrive at the crash scene) [1]. There are numerous challenges, which will be listed below and described shortly:

- Despite the availability of various high-quality, high-cost specialized devices for chest compression depth and frequency determination, low-cost solutions based on widespread mobile devices are not considered yet.
- The stability of the transmission is directly connected to an available internet connection.
- It is difficult to compute the actual compression depth out of simple accelerometer data as well as transmit only the relevant information.
- The quality, sensitivity, sample frequency and resolution of built-in accelerators vary a lot in modern mobile devices.

The aim of this work is to evaluate various algorithms for chest compression rate and chest compression depth detection as well as prototype and test them in a smartphone service/application. Another aim is to optimize the running application and guarantee a flawless transmission to the server (the main distribution place for data points and communication between clients). Therefore, a lightweight transmission system was needed, which operates fast and allows easy data transmission.

The theoretical background of the thesis is the algorithm implementation as well as the general CPR process and the associated time critical events. The two major issues in a time critical situation are the laypersons, who are overwhelmed by the situation and the difficulty for the dispatchers to instruct them over the phone [2]. Especially if a cardiopulmonary resuscitation is necessary, most people are not able to perform telephone-assisted CPR and chest compression at the right frequency, depth and with the required duration of up to 30 minutes until emergency aid arrives. Therefore, it is an enhancement for both sides, if the process is further guided with a live visualized reanimation process.

For the thesis an experimental research method is used, which basically involves a straightforward experiment. The implemented algorithms have been compared to various studies, which have already dealt with the optimized compression rate and depth. Also quantitative research approaches were made in order to measure numerical data. The main aim is to develop a running application with implemented algorithms or self-developed ones with an optimized data transmission for mobile devices.

The results have shown, that it is possible to develop an algorithm for peak detection or frequency detection, though it is physically and technically nearly impossible to measure a distance difference of cm's with only an accelerometer sensor (commonly used in every smartphone).

It is proofed, that it is possible to develop an application, which runs for at least ten minutes stable and transfers the data with minimal loss to the server and visualize it on a website with the required information.

Finally, it can be said, that the thesis or project can lead to further interesting investigations on the topic of quality enhancement in dispatch life support. Especially in situations, where a cardiopulmonary resuscitation is necessary and performed by a layperson.

List of Contents

1	Introduction.....	7
1.1	Problem	7
1.2	Research Question.....	8
1.3	Method.....	9
1.4	Goals	9
1.5	Structure of the Thesis	10
2	Theoretical Background and State of the Art.....	11
2.1	Design Considerations on a Mobile Resuscitation Platform	11
2.1.1	Basic Principal of the Mobile Application Client Usage.....	11
2.1.2	Basic Principal of the Visualization Server.....	12
2.2	Theories and Models	12
2.2.1	Physical Models of spatio-temporal parameters	12
2.2.2	Technical Models for spatio-temporal computation.....	17
2.3	Server, Streaming & Website	20
2.4	State of the Art in CCD and CCR detection	22
3	LifeStream Design & Implementation	24
3.1	Usage scenario for LifeStream	24
3.2	Architecture of the Mobile First Responder Client	26
3.2.1	Hardware Considerations	26
3.2.2	Usability Considerations	27
3.3	Implementation of the Mobile First Responder Client	28
3.3.1	User Interface Concept and Design	28
3.3.2	Low-level Prototype	29
3.3.3	High-level Prototype.....	31
3.3.4	Final Prototype.....	33
3.4	Implementation of the EMD Visualization Server.....	36
3.4.1	Server development and setup	36
3.4.2	Website development and setup.....	37
3.4.3	Data transmission	39
3.4.4	Visualization.....	40



4 Design and Development of Chest compression depth algorithm Approaches for the Mobile First Responder Client	41
4.1 Accelerometer research.....	41
4.1.1 General Calculation Idea & Sensor.....	42
4.1.2 Issues, Limitations & Hints.....	45
4.1.3 Possible Solutions & Filters	46
4.1.4 Research Insights	48
4.2 Implemented Algorithms (self-developed and existing)	49
4.2.1 Experimental App: Filtering & Integration.....	50
4.2.2 Experimental App: Re-calibration, Speed & Magnitude	55
4.2.3 Final App: Peak detection & frequency	61
4.2.4 Further Possible Methods.....	68
5 Evaluation of Chest compression depth algorithm Approaches for the Mobile First Responder Client.....	70
5.1 Aim of this Investigation.....	70
5.2 Description of Method of this Examination.....	70
5.2.1 Field of Research.....	70
5.2.2 Course of time	71
5.3 Final Application Results	71
6 Discussion and Conclusion.....	74
7 Summary	78
8 Reference List.....	80
9 List of Tables	83
10 List of Figures.....	83
11 List of Formulas.....	84
12 List of Listings	84
13 List of Abbreviations	85
14 Appendix	86

1 Introduction

The fast population ageing of society leads to an increasing number of out-of-hospital cardiac arrest situations. Often dispatch life support and CPR interventions have to be performed by physically weak or untrained laypersons and bystanders rather than medical professionals. This includes mostly family members and friends [3]. The fear of making bad decisions often restrains people from helping and saving life's or bridge the critical minutes until the ambulance arrives [4]. In Austria usually every car licence holder has to complete a first aid course during his training, but this course only covers the basic aspects of saving a life, like securing the crash scene, calling for help or providing minimal dispatch life support [5].

Especially if a cardiopulmonary resuscitation (CPR)¹ is necessary, most people hesitate to either push on a non-breathing person's chest as hard as they can or ventilate them. A cardiopulmonary resuscitation often requires immediate reaction and even if the chest compressions are not totally appropriate, the attempt is crucial to save a person's life. Over the past year's cardiopulmonary resuscitation has continuously improved and was further investigated during various researches (e.g. "Cardiopulmonary resuscitation and the 2500 universal algorithm: Has the quality of CPR improved?" [7]). Though the process is well organized, there is room for further improvement especially during the reanimation process performed by a layperson, who is guided by a dispatcher.

Today's mobile information- and communication technologies, such as smartphones, are equipped with multimodal sensors to measure important context and even vitalparameters, that can be used to assess the situation during the reanimation process. For example, nearly every smartphone includes numerous low cost sensors, like the accelerometer sensor, the main sensor type, that was used during this thesis for estimating chest compression parameters. Additionally, every smartphone is able to connect to some sort of network, as well as maintain an ongoing phone call and perform background tasks, such as transmitting real-time data to a remote host. These features of smartphones have been investigated and used during this thesis in order to develop a functional prototype, as well as algorithms for chest compression rate and depth detection. The described time critical situations require urgent acting as well as optimized processes and can be further enhanced by using the already existing technology, which everyone carries around – the smartphone.

1.1 Problem

Time critical medical emergency situations, as the name suggests, are situations, where a proper execution of all steps in the chain of survival [8] is crucial and therefore every second counts. If the procedure is not optimized or not well performed, the overall outcome has a negative effect on the health of the emergency patient. The major issue is, that the layperson is often overwhelmed by the situation and the responsibility for another person's life. Especially if a cardiopulmonary resuscitation is necessary, most people are not able to perform it appropriately. For example, they often hesitate, to push as hard as they can on someone's chest, as they fear to break their ribs. On the other side of the communication channel, while the emergency call occurs, the EMD (emergency medical dispatcher) coordinates the ambulance while continuously talking and instructing the calling person. In the worst case scenario, the layperson is alone and has to perform the CPR on a non-breathing person as well as to communicate with the EMD. Most persons will be overwhelmed with this situation and the communication during the process.

¹ Cardiopulmonary resuscitation is an emergency procedure performed in an effort to manually preserve the brain function and basic body functions until further measures are taken to restore the normal circulation and breathing in a person who is in cardiac arrest [6].

This leads to errors during the reanimation process like incorrect chest-compression frequency or unnecessary arrhythmical pauses. Though the EMD tries to keep the contact with the reanimating person as well as to motivate them, verbal instruction and communication lacks situational information, like the actual chest compression rate (frequency) and chest compression depth of the reanimated person. The dispatcher gives supporting instructions over the phone during the reanimation and instructs the layperson. The reanimating person is asked to count the pushes aloud, while the dispatcher orientates on a visual metronome. If the frequency is incorrect the dispatcher can lead the reanimating person in the right direction by instructing them to push faster or slower. This process is well established and investigated, though needs to be improved. Based on the current demand at improving the CPR, the project "LifeStream" from the University of Applied science in St. Pölten was established with various project members. One of these partners is Notruf NÖ GmbH the main emergency dispatcher in Lower Austria. The second partner is the Medical University of Vienna. The main question of research, as well as the subquestions, have been generated in this consortium.

The problem to be solved is, that the dispatcher is unable to review parameters of the reanimation process visually, which would be a great enhancement to emergency medical dispatchers. With a visualization the dispatcher would be able to further guide the reanimation process with usage of the current reanimation data (this means getting information about the current chest compression rate² and chest compression depth³).

With this task and cooperation between the project members it was possible to frame the central research question for the project as well for this thesis.

1.2 Research Question

The main question for this thesis is the result of the current demand in dispatch life support and time critical medical emergency situations. It is based around the reanimation process and the need to collect and transmit data. A requirement was, that the data acquisition is performed by the same smartphone, which is used for the emergency call. All data is streamed (within this thesis defined as in real-time data transmission) to a server, which processes the data and visualizes it on a prototypical client website, that acts as the emergency medical dispatcher's visualization software.

The main question (Q1) and sub-questions (Q1.1 – Q1.3) of this thesis are:

- Q1: Can low-cost and mobile sensors like the accelerometer in mobile devices (e.g. smartphones) provide high-density data for real-time CPR signal processing and data transmission?
- Q1.1: Which real-time algorithms can be used for frequency detection (chest compression rate = CCR) and distance detection (chest compression depth = CCD)?
- Q1.2: Is it possible to make a testimonial evidence about distance detection with the accelerometer (low cost sensor)?
- Q1.3: How can the application be optimized, as well as the data transmission, to ensure a fluent transmission and appropriate visualization on the client side (website)?

The most specific problems which will be dealt with during this thesis along with the other questions, are the algorithms for the chest compression depth and rate detection.

² The actual rate used during each continuous period of chest compressions within a one minute interval independent of pauses [9].

³ The actual depth of the compressed chest during a cardiopulmonary resuscitation.

1.3 Method

For this thesis an experimental research method with changing variables is used, which involves a straightforward experiment. The variables during this experiment have been the changing algorithms or some of the parameters during the several tests on a professional reanimation phantom. The control of the checked variables was performed with a megacode reanimation phantom, which allows to record the reanimation frequency and chest compression depth. Several algorithm approaches were implemented and were examined in a personal performance test on a reanimation phantom. Additionally, the implementations are compared to various studies, which have already dealt with the optimal chest compression rate and chest compression depth, as there are already existing standards for the optimal frequency and depth.

According to the European Resuscitation Council – ERC guidelines [10] the current (updated 2015) optimal chest compression depth is 54mm and the chest compression frequency is 100 pushes per minute. The calculation of every tested algorithm is compared to the European standards and validated respectively. Each algorithm or application has the same test setting, which makes it possible to compare the results, if they are worth to compare.

Also quantitative research approaches are made in order to measure the numerical data, which is generated while the application is running. The general aim is to develop or compare algorithms and an optimized application as well as a stable transmission via 4G LTE internet and the usage of a TCP protocol on mobile Android devices, which collect data during a medical emergency situation. The collected data gets analysed and appropriately visualized on a client website. The visualization of the data and the calculated chest compression depth and rate can also be compared to medical manikins (see [11], [12]).

1.4 Goals

The main goal of the thesis is to implement or develop algorithms for CCR and CCD detection, compare them to existing standards as well as to optimize the application for a stable and continuous data transmission.

Not only the application itself is important, there are many more things to watch out for. For example, the hardware of the accelerometer sensor varies from smartphone to smartphone. Therefore, the algorithms need to be adjusted individually for each new smartphone tested. This is achieved by a configuration menu or a dynamic correction of the algorithms during the running application.

The goals of this thesis can be achieved by further investigation into the right topics (for example the research of distance measurement and frequency detection with Android). Also the implementation of the algorithms in the pre-created application is necessary, as the basic structure of the application is similar each time.

The most important literature for this thesis are certainly various papers, which deal with the optimal compression depth and the right frequency, as well as some books about the algorithmic implementation (e.g. Peak detection and filtering of values). Various android- and programming forums are also used as book in order to enhance the application.

Furthermore, various projects are interesting, which deal with fast and effective data transmission over the network, as well as stabilizing the general application. Another goal is to store the data on the phone or export it in order to monitor the reanimation process. The data needs to be accessed easily and should be usable for further visualizations or calculations.

1.5 Structure of the Thesis

This chapter should give a basic overview of the thesis structure and can be used as guideline. For the avoidance of doubt, the chapters are visualized and described briefly (the descriptions do not correspond to the actual subchapters).

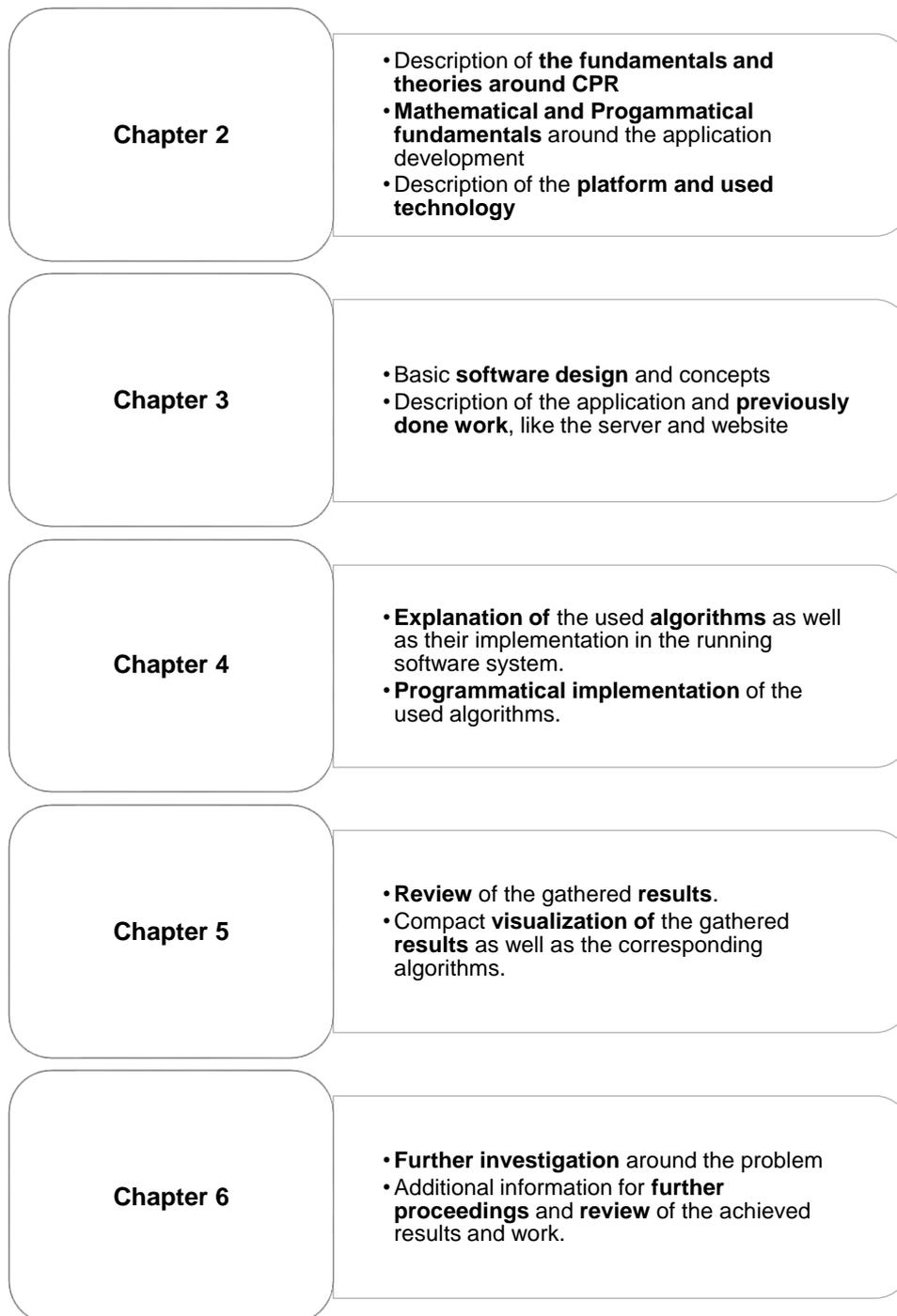


Figure 1: Structure of the thesis.

2 Theoretical Background and State of the Art

This chapter covers most of the theoretical background, used for the application and server development as well as the mathematical background for the tested and developed algorithms. It is important to give this introduction to the used technology as well as the mathematical considerations to be able to further investigate the thesis results.

More accurate descriptions about the application itself and the website can be found in Chapter 3, which covers most of the application and server (website) development. Also described here are some of the fundamental thoughts about the general idea as well as the implementation possibilities and restrictions.

2.1 Design Considerations on a Mobile Resuscitation Platform

During the project “LifeStream”, which is a result of the current demand for an improved reanimation during a cardiopulmonary resuscitation, an application was developed. The aim of the application is to support people in a time critical dispatch life support situation, more precisely during a cardiopulmonary resuscitation, which is guided by a dispatcher.

As nowadays nearly everyone is using a smartphone or carries a smart device around with them, the motivation was to develop a mobile application. The idea was that the application should be able to support a first responder during a reanimation in order to enhance the outcome of the cardiopulmonary resuscitation. The already telephone-instructed reanimation can drastically be enhanced for the dispatcher by a possible use of the website platform and the corresponding visualization tool. Also the layperson has a great enhancement by using the application on the mobile phone.

As a result, it should produce a better outcome for the patient as well as support the layperson or even professional first responders during the reanimation. The reanimating person is able to conduct the reanimation optimally guided by the dispatcher. The dispatcher is able to direct the layperson in the right direction not only by verbal feedback but also by visual feedback from the application, which is a huge enhancement. The audio visual feedback offers a new layer of insight for the dispatcher, as he is able to directly control the action of the reanimating person and is not reliant just on the verbal feedback from the layperson.

2.1.1 Basic Principal of the Mobile Application Client Usage

This point shall only give a short introduction or concept idea about the application and the idea behind it. A specific use case scenario can be found in Chapter 3.

At best the LifeStream-App is used the following way on condition of a previous installation:

- The layperson or first responder makes an emergency call and the competent dispatcher asks him to start the application (the phone call shifts to the background and the application overlays the screen).
- The dispatcher instructs the reanimating person to position the phone face up on the patient’s chest and place their hands on the phone.
- Afterwards they are instructed to press the start button which starts the data collection, calculation and transmission to the server:
 - Data is distributed to the client websites and visualized in a moving line graph.
 - The data or graph shows the current frequency of the reanimation as well as other parameters and allows interpretations about the compression depths.

2.1.2 Basic Principal of the Visualization Server

The website will also be further explained in the next chapter, though not that detailed as it is not a major part of this thesis. However, it is important for the whole project, as the dispatcher can thereby visually validate the reanimation process.

It is possible to visually gather the following information from the website (it continuously updates during the reanimation process):

- During the reanimation the application streams⁴ the calculated parameters and the raw data to a sever, where it gets further redirected and processed. The website shows who is currently logged in or active (accomplished by giving a unique identification number for each client, separated by phone client and pc client).
- The website shows the current reanimation curve or more precisely the reanimation depth approximation.
- Also the current frequency of the calculation is updated in a fixed time interval. The website shows always the mean frequency during the time interval. For example, it shows the mean frequency of the last 15 seconds. After additional 15 seconds the next average frequency is shown.

2.2 Theories and Models

This section covers the physical and technical theories and model concepts which were implemented during the project. The algorithms in this thesis are based around the physical concept of acceleration and its first and second order integral velocity and distance.

Although the server and the corresponding website are not a major part of the thesis, their basic concept and system has to be described, as they are directly related to the developed mobile application and not less important for the overall project. Therefore, a brief introduction to the concepts is necessary. The application will be described in detail in Chapter 3.

2.2.1 Physical Models of spatio-temporal parameters

As this thesis is based on utilizing already built in low cost sensors (in smartphones), it is important to understand and describe the relationship of acceleration, velocity and distance and how each of this magnitudes can be calculated. The used low cost sensor in the LifeStream App is the accelerometer sensor, which measures acceleration in m/s^2 . With this sensor the important CPR parameters are calculated or at least estimated.

Of course the smartphone has a bunch of other sensors, which could be used to further refine the calculations, but they are deliberately not chosen, as the application should run on older smartphones too. Nearly every smartphone contains at least a physical sensor like the accelerometer, which is normally used for gesture detection or something else.

The other sensors could be combined to get a more accurate calculation, but this would require a higher operating system level (e.g. a higher version of the Android operating system) as well as a newer smartphone. As nearly everyone should be able to use the LifeStream-App, it was given, that the accelerometer sensor is used, which should be able, to give information about the distance travelled with the mobile phone in hand.

⁴ Streaming is the transfer of data at a steady high-speed rate. Data streaming requires for real-time human perception of the data, the ability to make sure that enough data is being continuously received without any noticeable time lag [13].

For future calculations and models during this thesis it is important to know the following words as well as their definition and relation to each other:

- **Time**

Definition: This is simply used in order to describe a time duration. More precisely the difference between two time stamps represents a time duration. Typically measured in seconds.

Relation: Time is directly related to all other quantities, as they are dependent to time and use the time duration during the equations [14, p. 24], [15].

- **Distance**

Definition: Distance is the total length which is covered by a moving object independent of the direction of motion. Distance itself is a scalar quantity and refers to the total length of travel irrespective of the direction of motion. This basically means no matter which direction the object is moving, the distance is the path between two points.

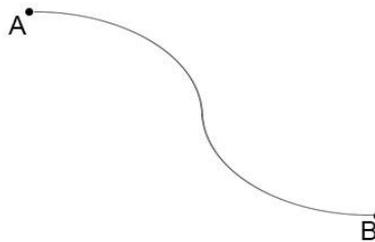


Figure 2: Simple visualization of a travelled distance. Note: not the direction is important, only the way.

Mathematically it can be computed with the following formulas, where the first is for continuous acceleration and the other not (in Chapter 4 there is even a distinction between vector and scalar calculations):

$$d = v_i t + \frac{1}{2} a t^2 \quad \text{Eq. (1)}$$

This formula above relates original velocity v_i , time t and constant acceleration a .

$$d(\text{or } s) = \int v \, dt = \iint a \, dt \, dt \quad \text{Eq. (2)}$$

In this equation distance is the integral of velocity or the second integration of acceleration.

Relation: Distance is related to all of the other listed physical quantities. However, the relation to displacement is most important, as it can easily be mistaken (see displacement for further visualization). Sometimes the distance is the same as the magnitude of the displacement, but it is not always the case for groups of segments. The distance normally increases continuously, even if a person is walking around a desk for example. On the other side the displacement fluctuates a bit and then returns to zero, after the person walked around the desk (compared to the starting point of the person) [14, p. 24], [15].

- **Displacement**

Definition: Displacement is the distance measured in a straight line or linear distance and always in a specified direction. This means, that displacement is a vector quantity and can be defined as the change in position of an object.

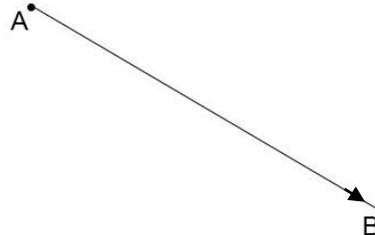


Figure 3: Simple visualization of displacement. This time the direction is important, as it is a vector quantity.

Mathematically it can be calculated by the following equation, where the sign of the resulting value designates a direction (positive or negative d). The formula is used to define the displacement in the normal three dimensional space:

$$\Delta d = d_2 - d_1 \quad \text{Eq. (3)}$$

d_2	...	refers to the value of the final position
d_1	...	refers to the value of the initial position
Δd	...	symbol used to represent displacement

Relation: Since the displacement is measured along the shortest path between two points, its magnitude is always less than or equal the distance. The shortest path between the two points A and B is a straight line, as seen in the following Figure 4.

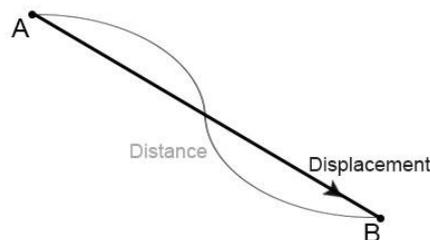


Figure 4: The relation or difference between distance and displacement.

The displacement of a moving object is also directly proportional to both the velocity and time ("Move faster – Go faster & Move longer – Go farther"). Sometimes the two quantities distance and displacement seem the same or nearly the same with their equations. For this thesis though, these above mentioned definitions are used (sometimes slightly adapted), as the compression depth can be approximated by the change of position or the displacement between two points [14, p. 24], [15].

- **Speed**

Definition: Speed can be defined as the rate of change of distance or in other words, speed is the distance moved per unit of time. With speed it is possible to make a point about how fast and slow an object is moving. Speed involves both, distance and time, where “faster” means either “farther” (or a greater distance covered) or “sooner” (less time needed). Speed itself is a scalar quantity:

$$\bar{v} = \frac{\Delta d}{\Delta t} \quad \text{Eq. (4)}$$

Where the line over the \bar{v} stands for mean or average and the Δ -Symbols indicates a change. In another form, speed can also be written as the first derivative of distance with respect to time:

$$\bar{v} = \lim_{t \rightarrow 0} \frac{\Delta s}{\Delta t} = \frac{ds}{dt} \quad \text{Eq. (5)}$$

Here the distance is represented by the variable s as this could be complicated with the derivation later.

Relation: In relation to the other quantities, speed corresponds to a slope on a distance-time graph. Thus the instantaneous speed of an object with non-constant speed can be found from the slope of a line tangent to its curve. Speed is more directly related to velocity as distance is to displacement [14, p. 24], [15].

- **Velocity**

Definition: Velocity is nearly the same as speed, though it is a vector quantity and defines the rate of change of displacement with time. Velocity therefore contains a directional information too. The formula for the velocity is technically the same as the one for speed, though it works with vector quantities and therefore has a bold \vec{v} in order to distinct it from speed:

$$\vec{v} = \frac{\Delta d}{\Delta t} \quad \text{Eq. (6)}$$

Relation:

Velocity is also related to the acceleration, though a final velocity can only be calculated, if the initial velocity is known and a constant acceleration is given. As the accelerometer of the smartphone is not accelerating constantly the equation is not useful here. The following Figure 5 shows the relation of velocity and speed [14, p. 24], [15].

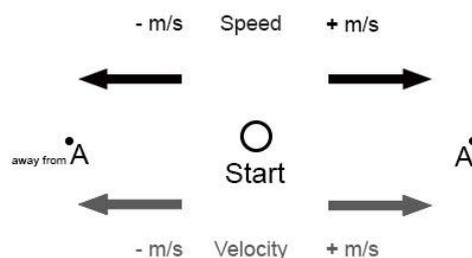


Figure 5: Velocity and speed relation.

- **Acceleration**

Definition: Acceleration describes the rate of change of velocity with time. Generally, acceleration happens, when an object speeds up, slows down or changes direction (all three happen during the usage of the accelerometer sensor in a smartphone). Acceleration itself is a vector quantity, which means it has a direction. Mathematically the acceleration can be calculated by the following equation:

$$a = \frac{dv}{dt} = v' = \frac{d}{dt} \left(\frac{ds}{dt} \right) = s'' \quad \text{Eq. (7)}$$

This means a is the first derivate of the velocity and the second derivate of the displacement after time. This formula is nevertheless only useful if the velocity is constant and instantaneous. The other formula here is used for calculating the average acceleration:

$$\vec{a} = \frac{\Delta v}{\Delta t} = \frac{v_f - v_i}{\Delta t} \quad \text{Eq. (8)}$$

v_f ... refers to the final velocity
 v_i ... refers to the initial velocity

Relation: Any change in the velocity of an object results in acceleration, either if the speed is increasing, decreasing or the direction is changed. This means acceleration is directly related to velocity, or depends on the change of it. Again as acceleration is a vector quantity, there are two kinds of acceleration, the average and instantaneous.

The average acceleration is determined over a long time interval. The velocity at the start of the interval is called initial velocity or starting velocity and the velocity at the end is called final velocity. The average acceleration is then a quantity calculated by the difference of both as seen above [14, p. 24], [15].

Physical Insight

After dealing with various physical models about basic motion detection and recognition based on one of the above described quantities, it is possible to further investigate the algorithms, which are implemented in the application. The relation of acceleration, displacement and velocity is important, as all of these three quantities are vector quantities. A vector quantity is a directed magnitude, which gives information about the direction. Using the example of displacement, it is theoretical possible to determine the final position of the mobile phone, if it is used for CCD detection. It would be useless to determine the distance, as this is only a scalar and counts up the travelled way.

Another point of interest is the way or direct way, the reanimating person has made with the phone from the normal chest position of the victim to the compressed chest position of the victim. This path can be obtained theoretically by using the displacement, as it covers or calculates the direct way from point A to point B. Based on these considerations the further investigations of this thesis are constructed or orientated. Even the frequency can be detected with this approach, because after a certain push threshold is exceeded, the push is correct and this counts to the total frequency.

2.2.2 Technical Models for spatio-temporal computation

This subsection deals with the basic technical considerations and theories which are used in the whole project and during this thesis. The chapter is segmented into several sub items which deal with the following topics:

- Accelerometer basics
- Server and Streaming
- Website and Visualization

It has to be mentioned, that during this chapter only the basic knowledge is presented and the basic concepts, which can easily be adapted to other projects. These concepts are relevant for the whole thesis and will be further examined in Chapters 3 and 4.

2.2.2.1 Accelerometer Basics

The accelerometer is a simple yet powerful low cost sensor and mechanical tool, which is implemented into nearly every smartphone, either modern or old. The accelerometer is combined with other sensors inside the smartphone or even in other devices and can give a lot of information, especially in conjunction with other sensors like the Gyroscope or the Magnetometer.

Though the raw sensor or basic accelerometer is a mechanical device for acceleration measurements (often used to detect shakes of the phone or tilts for example). It offers the possibility to measure the acceleration in a specified direction. The values measured by the smartphone are in m/s^2 and always include the acceleration and deceleration. In fact, this behaviour is a bit complicated or not easy to understand, as normally it would be logical, that if a car is accelerating, for example, it will not decelerate the same way or even further.

In order to understand the concept of the accelerometer the following Figure 6 can be used as help (imagine a ball inside a box).

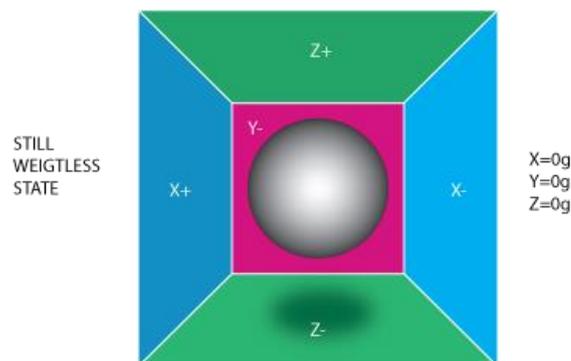


Figure 6: Accelerometer explained, with a ball inside a box [16].

If the box is seen as a vacuum space, where no gravitation affects the ball, the ball would rest in the middle like in Figure 6. Each axis has some kind of wall, as the point of view is from above, the Y+ Wall is removed and needs to be imagined. Each wall represents a movement and is pressure sensitive (imagine a phone flat on the table and a person sitting in front of it, doing the following movements):

- X+ and X- represent a horizontal movement (e.g. right and left with the phone).
- Y+ and Y- represent a vertical movement (e.g. down and up with the phone).
- Z+ and Z- represent a depth movement (e.g. toward the persons' viewpoint and fromward).

Now if the box suddenly gets moved to the left (see Figure 7 below), the ball inside the box will hit the X- Wall and it is possible to measure the pressure force, which the ball applies to the wall and therefore outputs a value for the X axis.

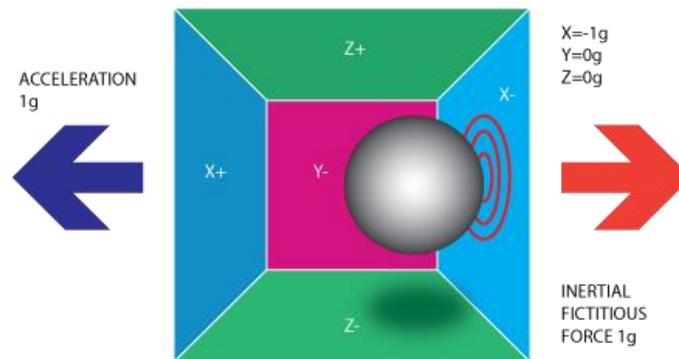


Figure 7: Moving the ball inside the box [16].

An accelerometer measures indirectly through a force that is applied to one of its walls. This force can be caused by the acceleration, but has not to be always.

Because if the box is placed on the earth, which it actually is inside a smartphone, the earth gravitation ($g = 9.81 \text{ m/s}^2$) influences the whole system. The ball will fall down to the Z- Wall and will apply a force of $1g$ on the bottom of the wall. This force is also included in every accelerometer data reading from a smartphone and has to be filtered out, by either various filters or by using another sensor type – the linear accelerometer (see Chapter 3).

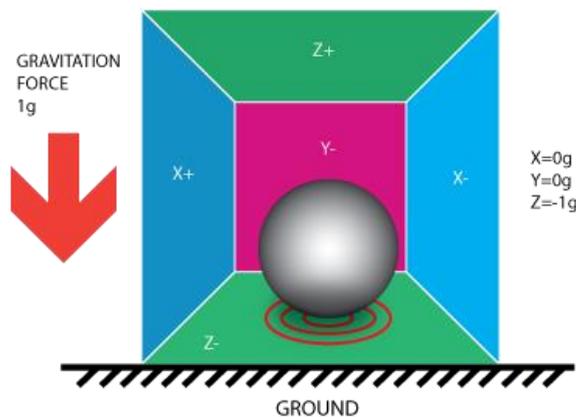


Figure 8: Ball in resting position with gravity [16].

In essence accelerometer measures force that is applied not acceleration. Acceleration just causes an inertial force that is captured by the force detection mechanism of the accelerometer. So far this model analyses the output for a single axis, but the accelerometers in smartphones are triaxial accelerometers and can detect forces applied to all three axes of the accelerometer. If the box would be rotated to the right for example, the ball inside the box would now touch the X- and Z- Wall and produce a value for these both axes.

The previous model has a fixed gravitation force and rotates the imaginary box, while the force vector remains constant. Though this is useful to understand how the accelerometer works with the outside forces, it is more practical to perform the calculations in a fixed coordinate system, where the force vector rotates around (shown in Figure 8).

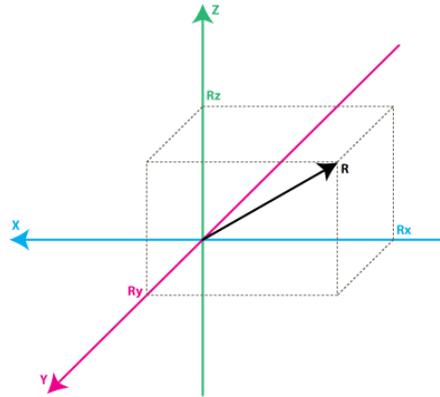


Figure 9: Coordinate system and force vector [16].

The vector \vec{R} is the force the accelerometer measures and R_x, R_y, R_z are projections of the vector on the X, Y and Z axes. The magnitude M of the \vec{R} vector can be calculated with the following relation (Pythagorean theorem in 3D [17]):

$$M = \sqrt{R_x^2 + R_y^2 + R_z^2} \quad \text{Eq. (9)}$$

Now if the acceleration needs to be found moment by moment (as it is done in smartphones), another equation takes into account, developed by Isaac Newton (law of motion). He defined acceleration in another way, by relating it to mass and force. If a certain force is applied to a mass, e.g. a kick on a football, the mass will accelerate.

Newton's second law of motion relates the three quantities force, mass and acceleration in the following equation:

$$F = m \cdot a \quad \text{OR} \quad a = \frac{F}{m} \quad \text{Eq. (10)}$$

In other words, this means acceleration is the amount of force needed to move each unit of mass. This equation is the theory behind accelerometers, as they measure acceleration not by calculating how the speed changes over time, but by measuring the force.

Of course there are various other types of accelerometers and other implementations, which also take into account for the developed application. The type of smartphone even makes a huge difference in accelerometer readings and calculations. Accelerometers are still physical sensors or mechanical sensors and can produce slightly different results which vary from type to type and smartphone to smartphone.

To put it another way, it is important to get more information about the phone and concentrate on the accelerometer sensor and the sampling rate as well as the usage of the either mechanical sensor or the technical one (which is a combination of various physical sensors) [16], [18].

2.3 Server, Streaming & Website

In this thesis only a small introduction to the basic server and website is given, as they are not directly essential for this thesis, though relevant for the whole system LifeStream and the LifeStream-App. Of course there are numerous possible implementations for servers and different ways to program them. For the project the NodeJs architecture has been chosen, which offers a slim and portable system with an easy installation. The research which lead to the decision for NodeJs are explained in the following Table 1.

Table 1: Challenges and solutions for server.

Challenges	Solution
 Some servers have latency – this means they are not really fast at processing huge amounts of data simultaneously and in real-time. This is important as the reanimation curve or data needs to be shown at the right moment. As real-time data transmission over mobile networks demands low-latency transmission.	 The only real solution to this problem, beside caching, scripting and configuration optimization, is to ensure that the server has sufficient CPU and memory. The server is provided by the university of applied sciences in St. Pölten (Austria) and offers a fair amount of space and RAM already, which provided a good base for the project setup and implementation.
 Servers normally need HTTP requests in order to send data – this means that they are in an idle state and waiting for some client to ask them for information or sending them a website. For the project the clients (websites) would have to fetch continuously for new data points in order to visualize them in real time. This would be very time-consuming and also inefficient for this specific issue.	 The solution is fairly simple though complex to understand. In order to ensure a bidirectional communication (server → client & client → server), Websockets are used. For this project the NodeJS architecture is used as already mentioned.
 Two way streams are not a general implementation of servers. They need to be implemented additionally or afterwards by using various technology.	 Websockets are able to stream and can be easily implemented.
 The connected clients need to be separated, as commonly more websites will connect to the server (this would be a minor problem) and more LifeStream smartphone applications. If more smartphone applications are connected, there must be a distinction between them in order to separate the various reanimation curves (produced by each application).	 As this thesis and project is mainly a proof of concept or enhanced prototype, only the basic structure for a multiclient handling was implemented, by giving unique identification numbers IDs to each client. Distinction between: <ul style="list-style-type: none"> • Android client • Website client

As stated above in the table, there are several issues in association with developing servers and streaming data, especially if it happens all in real-time. It has to be mentioned that there are various other problems which deserve a solution, but the mentioned ones are the most influent for this thesis and project.

According to the researches, sundry technologies and solution approaches have been tried out and reviewed. In the following a few of the most interesting ones and the conclusion to each approach are listed and explained. The last one is explained a bit more in detail as it is used for the LifeStream server.

2.3.1.1 Web service approach

A possible solution for the server would have been, to cede the streaming and data processing to an external website, as the processing, streaming, saving and redirecting of data is a complicated procedure. There are numerous streaming services available, which offer a great layout and platform, but often need registration or are even paid services.

For example:

- *plot.ly* (<https://plot.ly/>)
Allows to transmit the data from a smartphone directly to the plot.ly server where it gets plotted in real-time.
- *PubNuB* (<https://www.pubnub.com/>)
This service is specialized in simplifying the general process of data streaming and packaging. Nevertheless, the whole API has to be involved and additional functionality needs payment.

Conclusion:

Existing web services are conceivably improper for this project and thesis, as they are not only in charge but also not easy to implement in the existing application. They do not allow personal preferences and changes in the existing API.

2.3.1.2 Sockets

For another approach sockets have been further investigated and worked with, which allow a bidirectional connection. This means the client is able to send messages or notifications to the server and vice versa. Part of the first trials has been the development of a simple Java Server, which displays the received data (from the smartphone application) on a console.

This worked great as long as the device and client have been in the same network or connection. In order to grant full accessibility Websockets are important, as they allow the bidirectional communication over the web.

Conclusion:

As sockets are easy to implement and allow fast data transmission, they are qualified for simple bidirectional communication between a client and server. Nevertheless, as the application and website should be reachable from all around the world, Websockets are the technology of choice.

2.3.1.3 Websockets

After a few inappropriate attempts or technologies, the final decision was to use Websockets. Former real-time communication was achieved by using technologies like Ajax or Comet (both web application systems) and the Http-Request-Response system.

Websockets are technologies, which allow a full duplex communication over a single TCP connection. This means, it is possible to implement real-time systems much easier, as client and server are able to communicate with each other (over the web). Furthermore, the latency can be reduced by using Websockets and the transmitted data in the HTML header is also reduced.

During project development it turned out, that NodeJS is suitable for a fast and easy solution, as it even supports multiclient handling, which is important for the future usage and dealing with multiple smartphones, that transmit data to the server.

NodeJS is an event based JavaScript runtime environment, based on google chrome. Normal program code is processed sequentially. Node is single threaded and uses a concurrency model based on normal event loop. Node is a non-blocking architecture, which does not make the program wait, but instead registers a callback and lets the program continue. As a result, it is possible to handle concurrent operations without multiple threads of execution (scaling the program is much easier). A callback function is called or executed at the completion of a given task. This allows to jump basically from function to function without waiting for blocking functions like for example the file input and output. This results in the high scalability of NodeJs, as it can process high numbers of requests without waiting for any function to return its results.

The main advantages are, that Node allows to perform other tasks, while waiting to be notified when the response is available and that the Node application is not buffering data into memory (instead it is outputting it chunk-by-chunk) [19]. That is the reason, why this architecture is used for the thesis as well as the project, as it allows a simple but powerful and fast implementation and is intended to work in real-time.

2.4 State of the Art in CCD and CCR detection

The following examples should only give a brief introduction into current CCR and CCD detection devices. There are numerous other approaches of detecting the distance moved with the smartphone, but none of them so far is directly related to this thesis concept (using the smartphone in dispatch life support and visualize the results at the emergency communication and coordination centre for the dispatcher).

The examples listed in the following Table 2 are only a small part of the available technologies and researches. They are the most interesting ones, as they are somehow related near to the project behind this thesis. The thesis itself though deals with comparison of algorithms and the trial and error of various implementations.

Table 2: Related work and state of the art.

Name & short description	Image
<p>PocketCPR: This device is produced by an international company which also is situated on the German market, called "Zoll". Under www.zoll.com it is possible to see their products as well as their fields of research and interest. They have created a mechanical device, which enhances the quality of CPR. It is called "PocketCPR" and gives simple audio-visual feedback in real time. The device shall also help to reduce the stress of the situation, as it gives simple commands and tries to calm down the user [20].</p>	 <p data-bbox="1050 1738 1241 1783">Figure 10: PocketCPR by Zoll [17].</p>

<p>PocketCPR for Droid: The next product is also from the same company and is directly an application which gives real-time feedback of an ongoing CPR through the smartphone. It utilizes the accelerometer sensor of the smartphone to measure the compression depth and compression rate and gives the user audio-visual feedback. The application also introduces the user to the whole process of CPR if he is not familiar with it through the first aid course. The application is currently only available for testing and learning purposes [20].</p>	 <p>Figure 11: PocketCPR for Droid by Zoll [17].</p>
<p>Development of Android Based Chest Compression: One of the most interesting studies, which was performed in 2014, involves also the usage of inbuilt accelerometer sensors in smartphones. The aim is to improve the quality of the CPR by directly measuring the CCR and CCD. According to the article they have been able to gain meaningful information with the accelerometer sensor, though not providing insight in the full algorithm and used sensor technology. The main difference is, that the feedback is directly on the smartphone and not on a website visualized or directly at the dispatcher [21].</p>	 <p>Figure 12: Monitoring screen App [21].</p>
<p>CPREzy CPR Assistance Device: This device is mainly designed for CPR assistance and offers a simple interaction. The device does not give voice prompts for the reanimating person and gives live feedback during an ongoing reanimation. It has an audible chirp and visual light pacing system with a metronome component. The metronome gives a chirp tone at 100 compressions per minute in order to match the CPR guidelines. According to a study from 2005 (“CPREzy: an evaluation during simulated cardiac arrest on a hospital bed”) the device was compared with a normal reanimation and the results have showed, that there was no significant difference in compression rate or duty cycles between the techniques. Though the CPREzy device was associated with significant changes or enhancements in chest compressions [22].</p>	 <p>Figure 13: CPREzy device and included stuff [23].</p>

Conclusion:

According to the primary research it is clear, that there are various other projects which generally deal with the same subject or field of research. Most of them not directly, as they are either more concentrated on the direct user feedback or on the dispatch life parameter detection. The main idea though of every related other project and this thesis is to enhance the overall quality of the CPR and give the layperson feedback.

The mentioned applications provide the feedback directly to the laypersons, whereas the project behind this thesis intends to stream the information to the emergency call centre. There the dispatcher can judge the current reanimation on a corresponding website and give feedback to the reanimating person.

3 LifeStream Design & Implementation

In this chapter the main design of the whole LifeStream project as well as the basic usage scenario is defined in order to give a brief overview of the whole project structure. Especially the mobile client application is explained in more detail, as it is the base for the whole algorithm implementation in Chapter 4 and evaluation in Chapter 5.

To realize a project, not only the project members and the main idea are important, but also the schedule of the used technology as well as the definition of a use case scenario. The following sub items show the important parts of the project and thesis as well as their design process and basic idea behind each part. Once again, the project consists of the following main parts:

- The LifeStream Mobile Client
- a LifeStream Emergency Medical Dispatch Visualization Server to handle clients
- and a LifeStream visualization website for the reanimation data

Mentioned must be, that the server and website are not fully described here but nevertheless important. Although the implementation of the server and the website programming is somehow related to the application and was also a huge part during the algorithmic testing.

3.1 Usage scenario for LifeStream

A usage scenario refers to examples of the actors in contrast to use cases, which refer to generic actors (e.g. a customer). Scenarios are easier to understand as they are more personally, which increases their understand ability. They also describe a single path of logic, whereas use cases typically describe several paths (basic and alternate paths) [24].

The following two short scenarios are fictional and do not refer to a real person or real event happened.

The scenarios are only inspired by the book “60 Fälle Rettungsdienst” [25]. It is intended to show both sides (the dispatcher and the reanimating person), during a sudden cardiac arrest. Both scenarios are related to each other and are described from a different point of view or angle in order to give an impression about the application usage and possible outcome.

Scenario Cardiac Arrest with the perspective of EMD:

The emergency call arrives at 8:43 AM at Michael Maier’s workstation. The emergency cue is “unconscious person” and the place of accident a property market. An employee (Thomas) of the market set the emergency call, as he found an elder man lying on the customer toilet not addressable. The man was asking for a toilet and after 20 minutes of disappearing the staff sent an employee to look after him.

According to the previous information, the time without any treatment is estimated to 4-6 minutes. The dispatcher Michael instructs the employee with the basic treatment (positioning, check for breath, etc.). The employee interrupts Michael and mentions that he has a medical application called “LifeStream” installed on his smartphone. Michael immediately instructs him to position the smartphone appropriate and start the application. With the usage of the application Michael is able to further instruct the employee and correct him during the reanimation process. Due to the application usage and immediate reaction the old man was stabilized until the arrival of the emergency medical service (EMS) who saved his life.

Scenario Cardiac Arrest with the perspective of layperson: property market

Thomas Bauer started his work at 8:00 in the property market and is currently working at a shelf with screwdrivers. He is 27 years old and a big fan of new smartphone apps and always trying out new ones. Thomas recently installed a new application called "LifeStream", which is intended to be used during a sudden cardiac arrest situation. He thought it would not be harmful to have such an application, though it is very unlikely to get in a situation where it is needed. Suddenly the area director is calling him via the speakers and tells him to return to the information point in order to get a new task. When Thomas arrived at the information point and was starting to talk to his area director an elder man interrupted them, asking for the toilet. He seemed a bit disoriented and was sweating, but they pointed him the way to the toilet. After a while, during their conversation, the area director advised Thomas to look for the elder man, as he was not returning from the toilet.

Thomas was looking in the customer toilet and found the elder man lying motionless on the floor. He immediately called the emergency number and they instructed him further. During the conversation he remembered that he had installed the new application and directly told it to the dispatcher. The EMD instructed him to lay his smartphone on the middle of the chest and start the application. Thomas was than further guided by the dispatcher and pushed with the smartphone between his hands and the chest of the non-breathing man. After 8 minutes the emergency medical service arrived and was able to take care of the elder man.

Both scenarios above can only give a small introduction to the possible usage of the application and the system. The dispatcher for example is able to view the reanimation curve as well as the frequency directly on the website and can then correct the reanimating person directly instead of just relying on audio feedback or a questioning scheme.

In the following Figure 14 a basic workflow or usage is visualized which covers the whole process and possibilities of the project.

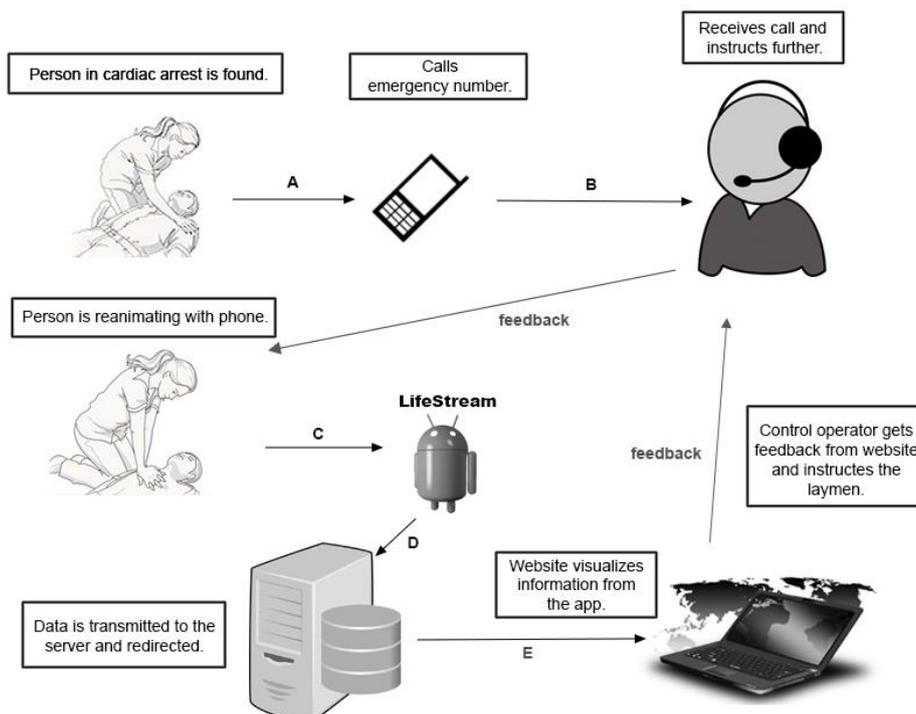


Figure 14: Reanimation process visualization.

As seen in the Figure 14 above, the reanimation is intended to be done with the smartphone laying on the chest of the victim. The reanimating person must perform the reanimation with the smartphone between the hands and the chest, as shown in Figure 15.



Figure 15: Reanimation with phone and LifeStream-App.

During tests and the planning phase of the project as well as this thesis, most people were concerned about the phone. The biggest hesitation for them was to push as hard as they can on their phone or generally a smartphone, as it could easily crack.

Against most people's concerns the smartphones do not crack that easily, especially if they are placed horizontally as shown in the above Figure 15 and the pressure on it is not too punctual. They can even support the reanimation as they offer a wider contact face, which also helps increasing the overall force applied to the victim's chest.

Based on these user scenarios and some interviews the further application design is influenced, as it should be able to use the application during a medical emergency situation, where a complicated system would be obstructive.

3.2 Architecture of the Mobile First Responder Client

During the application development a lot of researches are necessary which lead to various results and approaches. In this subitem the application development is described further as well as the application itself and its main functionality and parts. The following described application and design considerations are equally with only small variations for the algorithm implementation. With the scaffold it is possible to further expand or vary the application as well as its functionality.

3.2.1 Hardware Considerations

The whole project is programmed and realized with different technologies and platforms. They all work together in order to produce the prototype. For the application Android Studio was used as development environment. Android Studio is the official IDE for Android app development. The applications are programmed in Java, with some additional functionality of the development environment.

As external library only the following Socket.io client implementation for Android was used (<https://github.com/Gottox/socket.io-java-client>), which is easy to use and incorporate into an existing android application. A simple and small part of code allows to transmit and receive messages with the Socket.io technology. This technology enables a real-time bidirectional event-based communication.

All applications have been tested with the following real devices (smartphones), which all include an accelerometer sensor and were using at least Android 4.4 (KitKat).

- Samsung Galaxy S4 i9505
- Nexus 5
- Nexus 5x

The android application itself requires at least API⁵ level 19 which is KitKat, but is targeted in the final version for API level 23 or Marshmallow. The reason for such a high target API is, that some of the great new functionalities of this API are used. For example, the powerful Toolbar or conversion of the ActionBar into a toolbar. With a toolbar it is possible to create a simple but effective overlay menu.

3.2.2 Usability Considerations

In order to design the application interface appropriate, it was important to define the usage scenario and the basic requirements to the application. Based on the usage scenario, various interviews with experts and continuous testing during the development, the following requirements to the application are defined:

1. Simple usability

Essentially this means, that the application is easy to understand and does not overcharge the user with a huge amount of functionality and menus. The purpose is simply to start the data gathering, transmit it and stop the data gathering. Maybe save the data locally on the phone, for eventual analytics. No fancy colours or designs are used which could distract the user (whereas they would anyway be unnecessary as the user is covering the phone with his hands).

2. Restricted functionality

As the layperson is often frantic during a time critical situation, especially if he or she has to reanimate another person, the application must be protected against unwanted termination. This means the buttons (as far as possible without overriding the android operating system), which are normally used to terminate the application or go back, are overridden with other functions. These functions prevent the application from accidental close, as the user or reanimating person is touching nearly the whole screen.

The application is also running in full screen mode, again as far as possible without changing the android operating system, which disables the notification bar. In order to fully control the application or phone and even the home button, it would be necessary to directly release a specific android operating system, which allows the change of the home button (this button cannot be accessed by normal application development environments). As this would be another huge task, it is not part of the overall project, nor this thesis. Nevertheless, it is worth to mention and possible.

Another important feature of the application is to stay in wakeup state the whole time. Normally an application would close or change to battery safe mode after a pre-defined amount of time (e.g. 1 minute), if no direct interaction is with the devices screen. This had to be disabled as the phone should run at least ten minutes⁶.

⁵ API = application program interface; This is a set of routines, protocols or tools in order to build software applications. It specifies how software components should interact [26].

⁶ Ten minutes = average time until the ambulance arrives at the crash scene [1].

3. Easy configuration

Not only the simple and slim design is important for the application, but also the configuration possibilities, especially for the algorithms. In order to adjust various parameters for the calculation a simple and non-intrusive menu is used. The menu allows to change different parameters of the calculation as well as the general application. Each menu will be described separately in combination with the used algorithm during Chapter 4.

For the final application, the menu is not directly necessary, although it allows to adjust the calculation to special circumstances (e.g. older smartphones with slower runtimes, etc.).

4. Fast transmission and small data format

This is another important part of the whole app, as the functionality of the application relies on a connection to the internet. Without this connection no data transmission is possible, thus no calculation or visualization. As the connection is often not stable or perfect, the transmission and sent data must be optimized (see point 3.4).

With these important requirements to the application a first mock-up was designed and later implemented. During the design process it was often necessary to adjust various parts of the application in order to further improve the final result.

3.3 Implementation of the Mobile First Responder Client

In order to develop an application a design process is necessary. During this design process the application is always adjusted. Each part of this adjustment is described briefly in the following sub items as well as the concept or functionality. The following sub items are intended to be a roadmap to the final result. They give an impression about the first ideas and the considerations, which lead to the final prototype result.

3.3.1 User Interface Concept and Design

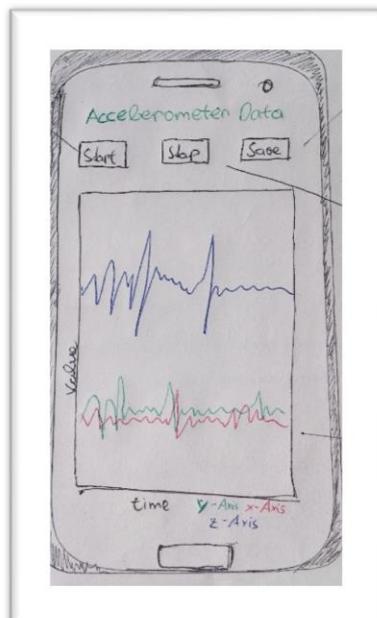


Figure 16: App Mock-up.

At the beginning there was just a simple hand drawn mock-up, which was used as guideline for the application development.

In this sketch, the application consists of the following parts:

- **Start Button:** This button allows to start the data gathering and begins the data transmission.
- **Stop Button:** With this button the streaming and data gathering can be stopped.
- **Save Button:** After the application is stopped, it is possible to save a .csv file and an image with the collected data on the phone.
- **Graph view:** Here the accelerometer readings are visualized after the reanimation finishes.

It is intended to visualize the data only after the collection. The screen should be free for the reanimating person in order to place the hands there and to push.

The above shown rough concept of Figure 16 was further refined and improved during the development process. Nevertheless, the main purpose of the application is always to support the users as good as possible and not interfere them.

With the chosen design it is possible to understand the application easily and fast, as there are not more than 3 buttons. Essentially the design is concentrated on easy handling and rapid reaction, since a time critical situation needs no fancy or special design. In this case functionality comes for design, as the reanimating person needs nearly the whole surface of the phone for the reanimation process.

Another important feature which is included in the application design is the locking of the screen, once the Start button was pressed, in order to prevent the user from accidental restarting the application. The Start button starts the data gathering with the accelerometer and the transmission of the data packages (also the calculation in later implementations). With the Stop button, the transmission and the data acquisition can be stopped. In order to prevent an accidental hit of the button, it is locked too and can only be unlocked under certain circumstances.

3.3.2 Low-level Prototype

The first implementation of the application included only one activity, the main activity, which had 3 Buttons. Before the further functionality is explained as well as the 3 buttons and the application interface, a short introduction to the Android activities, views and background services is given, because activities are the chalkboard of application development, where functionality is placed. Views are used to present information or further stuff and background services are important to transmit information, wait for it or do other stuff, which the user does not has to see.

Activities in Android:

Activities are application components that provide views with which a user can interact with, such as pressing a button or entering a name for example. Each activity has a window, that typically fills the whole screen of the smartphone or smart device, where the interface is drawn. The window can also be smaller and flow on top of other windows as well as link between them. This means an application usually has a main activity and some other activities, where data can be passed around. The main activity is present, when the user is launching the app and can lead to further activities [27].

Views in Android:

Generally, the android user Interface consists of views. A view is a representation of a widget that appears on the visible screen, in order for the user to interact with it. There are various types of views for example basic views (TextView, EditText, e.g.), list views (long lists of items are displayed inside most of the time) or menus (display additional items) [27].

Services in Android:

A service is a component that runs in the background of the application and has no direct interaction with the user (this means it is also not bound to the life cycle of an activity). Services are used for repetitive and long running operations, such as downloading a file or streaming. By default, the services run in the same process as the main thread of the application. Therefore, it is common to use asynchronous processing to perform resource intensive tasks in the background (e.g. saving data to a .csv file). Most of the time the services run in their own threads, which are terminated if the service is no longer needed or stopped [28].

Now carrying on with the explanation of the first programmed app. The first application does not include all the above mentioned technologies, as it only has one main thread, that is running as well as one activity. In this activity all three buttons are placed as well as an initially invisible view, where the accelerometer graph is plotted. These are the placed buttons as well as their functionality and relation:

- **Start Button:** Upon launching the application this button is visible and the others are greyed out and inactive. With a press on this button, the accelerometer sensor starts to read the data and transmits it to the server (also the eventual algorithm is executed and calculations are performed).
- **Stop Button:** Once the application is launched with the Start button, this button is enabled and can be pressed in order to stop the data acquisition as well as the transmission. The other two buttons are greyed out in the meantime. After the application was stopped, the data is plotted on the device screen within a scalable view.
- **Save Button:** This special button is greyed out from the beginning until the Stop button is pressed, as data can only be saved, once it is acquired. After pressing this button, the collected raw data is saved into a file and an image of the plotted reanimation curve is stored on the local phone storage as well.

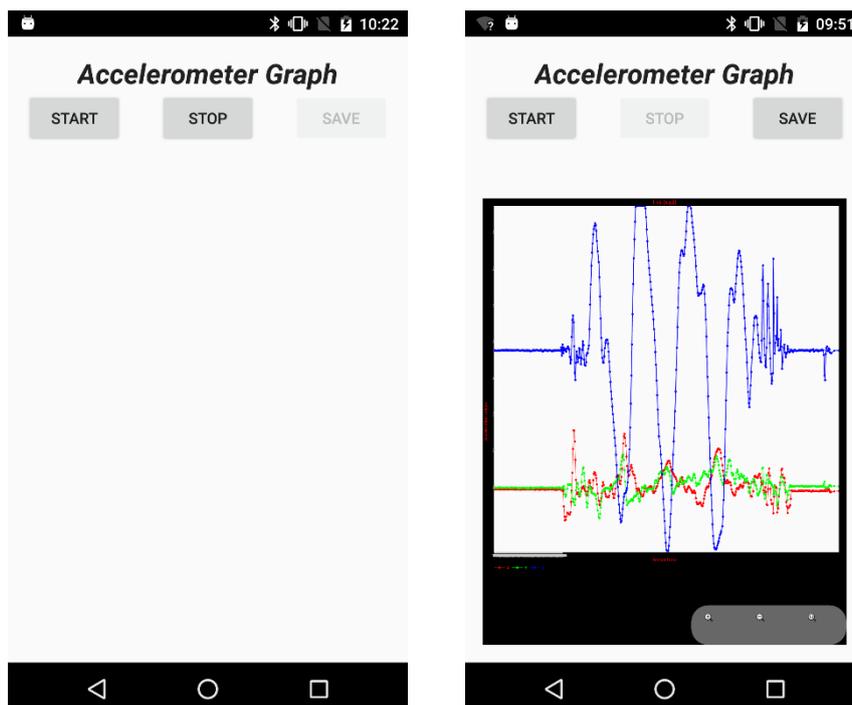


Figure 17: First implementation of the draft.

Annotations:

As the above described design implementation is one of the first ones, it was not really optimized yet. There is no full screen enabled for the application, nor are the buttons overridden in order to block their accidental usage. The plotted data consist only of raw accelerometer readings and is processed with a low pass filter (further insight at Chapter 4). By clicking the Save button, in this version, a .txt file is created and a .jpg file, which are stored locally on the phones storage.

3.3.3 High-level Prototype

Of course there have been several other versions of the design during the application development process, but they will not be mentioned here in full detail. Although there is one interesting design idea, which was also implemented and involves gesture control. The idea was basically, that the user has to make a certain gesture, in this particular case, drawing a circle, in order to unlock the device screen. The intention of this gesture controlled application is, that the user cannot accidentally hit the Stop button during an ongoing reanimation, as he or she directly has to touch the device screen with one finger and perform a special predefined unique gesture.

The gesture was defined by using Google's "Gesture Builder". This is a simple but powerful application (supported since Android 1.6 and included in the API), which allows developers to create, load, store and recognize unique gestures. Once the gesture is created in the builder application, it can be saved as library and used afterwards in any other application.

In order to make use of the created gesture, it has to be implemented in the target application via a `GestureOverlayView`. A gesture overlay is a simple, invisible drawing board, on which the user can draw his gestures. The overlay can be on top of other views and recognizes gestures in either the whole application or only a specific part of the application [29].

A circular gesture is used, as it is still complicated enough to not be done accidentally but easy enough to remember or perform after the application can be finished and the person is still under stress. The circular gesture has to be performed even in a specific direction and with a minimal radian. In order to check for an exact gesture, a much more complicated implementation would be necessary. For example, if the user is only allowed to draw the circle in a specific amount of time and with a defined radian, it would require a lot more mathematical operations. As this is mainly a proof of concept and works already pretty well with the directional circle gesture, further improvements do not seem necessary.

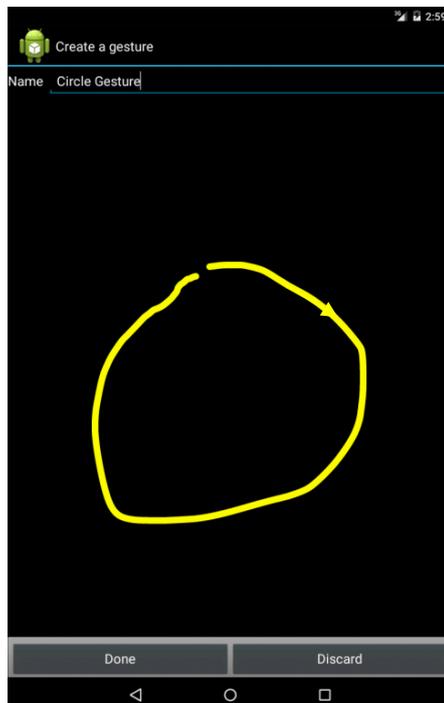


Figure 18: Circle gesture with "Gesture Builder".

The left Figure 18 shows a similar circle gesture like the one used in the following designed application. The circle is not fully filled as shown in the figure and directed clockwise.

With the direction of motion, it is possible to further secure the application, as only one specific direction allows to unlock the screen. Though the shape of the circle does not have exactly to match the predefined one, the direction allows to enhance the difficulty a bit further. With the generated gesture, it was possible to change the design of the application and enhance the security (in terms of accidental closing the application or stopping the data transmission).

Only after performing the defined circle gesture, it is possible to unlock the device and stop the data transmission.

After implementing the gesture overlay as well as changing the functionality slightly the final application looks as follows. In order to start the accelerometer reading as well as data processing, the user has to press the button in the middle of the application, which changes the view to the right application interface, shown in Figure 19 below.

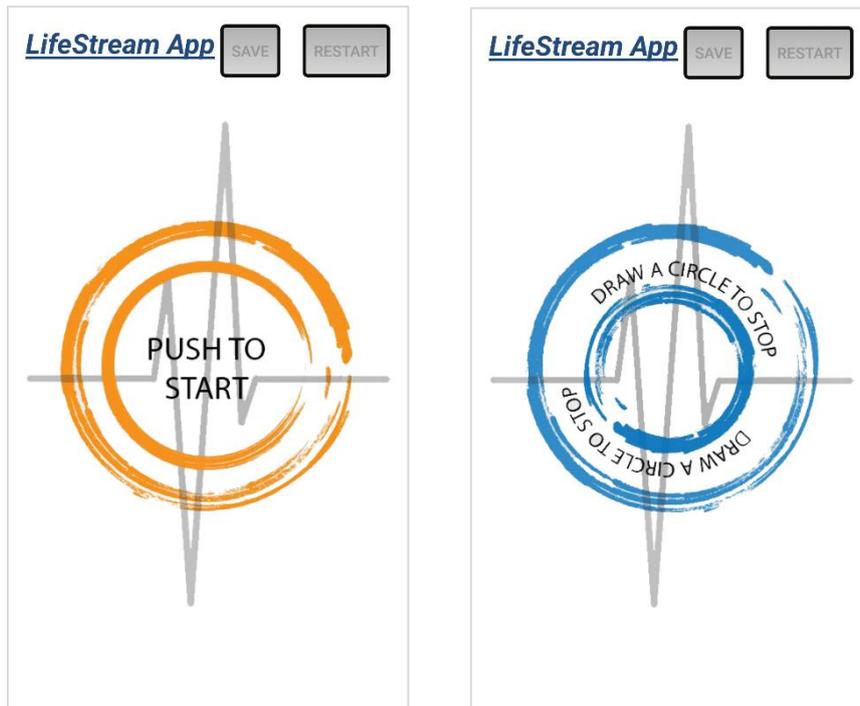


Figure 19: Gesture design implementation of the app. Design made by Melanie Kain & Nina Hladil.

The left application interface shows the start button, which basically is a big button with an overlaid image. Once the user presses the button, the view switches to the other screen and the gesture overlay is enabled. The user has to perform the circular gesture and can unlock the screen. The circle has to be performed from left to right (a graphical hint would be implemented in the final result).

Annotations:

In this app version two additional small buttons are included, which are necessary for the application. The Save button has the same functionality as in the other applications and stores a .txt file with the raw data as well as a .jpg file with the plotted curve. The Restart button is special and was introduced only in this app version, because the control of the application is only in one activity, the main activity. In order to switch between the both screens, a view switcher is used, which shows one component and hides the other ones. It is possible to cycle between the saved views in this view switcher. In order to show the starting view, it was necessary to implement the Restart button, as the user needs to switch back to the main screen.

Although the design is unique and great, it was not used for the final application. One of the reasons, why it has not been chosen is, that a gesture is often hard to perform in a stressful situation, especially in a time critical event. Also the validation of a specific radian for the circle as well as size would require much more calculations and resources. The available resources are used for the data transmission as well as the calculation.

3.3.4 Final Prototype

This version of the application is the foundation for the algorithm development in Chapter 4. Only the menu was changed for each algorithm, but the base application is equal to the following explained design.

Again this design's intention is to support the user and not to overload him or her with information or functionality. Therefore, only two Buttons are present in the main activity:

- **Start button:** Again this button starts the accelerometer and the calculation as well as the transmission to the server. It is slightly bigger than the Stop button, as the user should directly see where to activate the app. Upon pressing the button, the data acquisition starts. One special feature here is, that the Stop button will not get enabled, if the Start button was pressed (as in previous versions).
- **Stop button:** This button can only be pressed, if the user unlocks it with the small menu, located in the upper corner of the interface. Once the button is pressed, the accelerometer stops the data acquisition and the calculation and the application switches to another activity which holds the results.

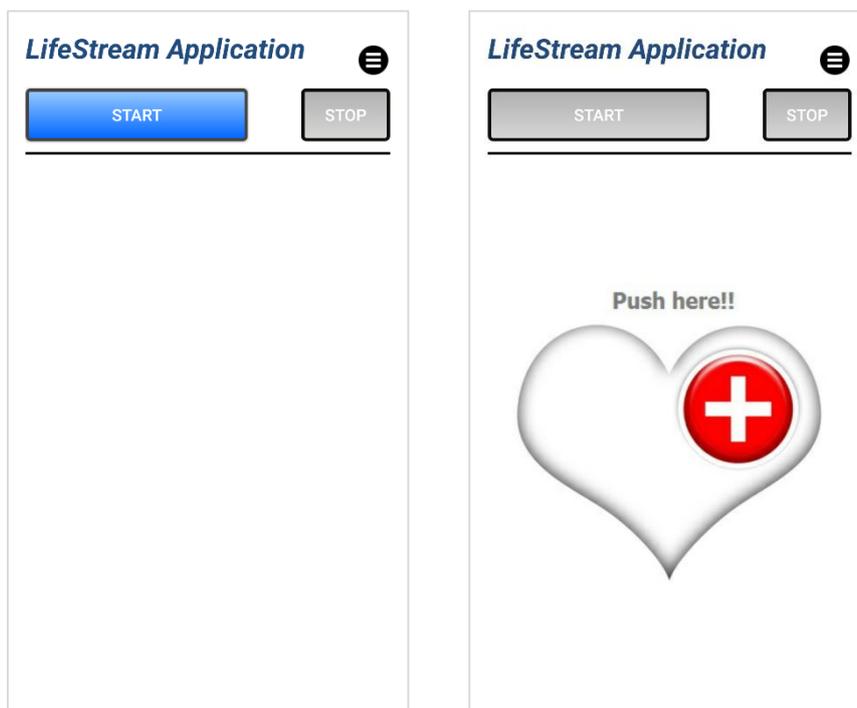


Figure 20: Final used application design.

The left part of this Figure 20 again shows the application after launching it and also after restarting it. In the right part of the figure the running application is shown, which has a blocked Stop button. Once the stop button is pressed the results are visualized on a second activity. This basically means, that the application has further pages (like a website and the user is changing from the homepage to another).

With the additional activity the resource management is drastically improved as well as the readability. The results do not have to be overlaid over the reanimation screen and can be visualized in the background. The visualization is performed by another thread, while the saving of the data is also performed in this thread. The data can be saved in a .csv file in this application in order to make further analyses with other tools much easier.

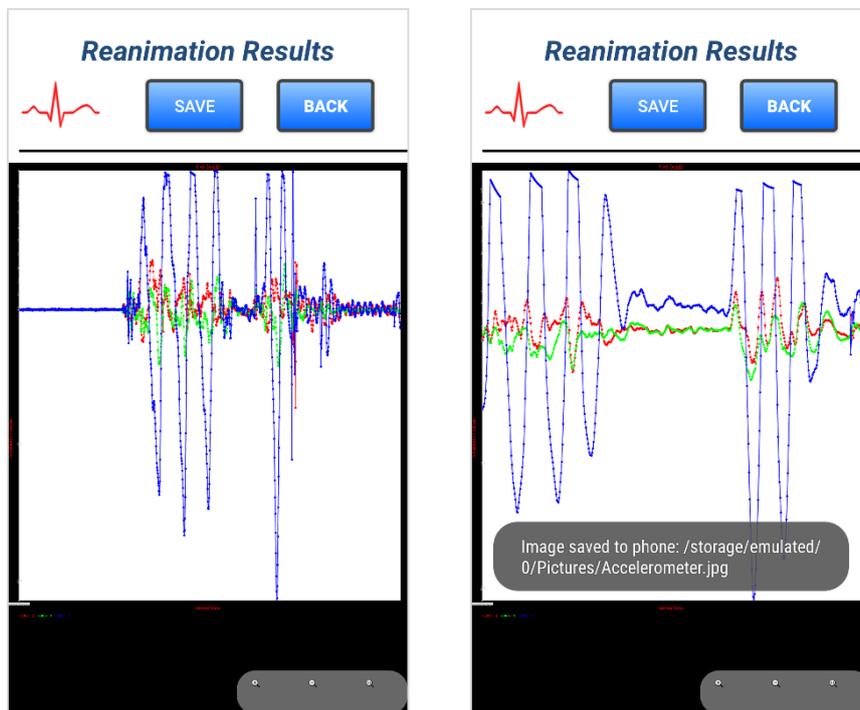


Figure 21: Final application result screen and saving.

The Figure 21 shows the resulting screen on the left, which is opened, after the Stop button was pressed. It allows the user to zoom in again in the curve as well as store a snapshot and the current data on the phone. On the right the notification is shown, after the Save button is pressed.

Special feature – menu:

As this version of the app is also used for the algorithm implementation, it includes a menu which offers great versatility for the user and especially for configuring the algorithm. Inside the menu there is a menu item to adjust various parameters of the algorithm. The menu is also used to unlock the Stop button, as it is only a small button in the right corner of the device screen and is hard to be accidentally clicked.

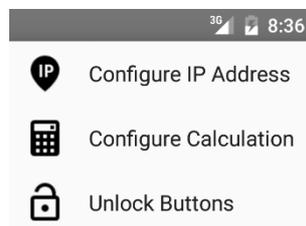


Figure 22: Subarea of the application - menu.

In Figure 22 above a subarea of the application, which includes the configuration menu, can be viewed. It has three menu items, where one is for configuring the IP address of the sever (if it changes), the other for the algorithm configuration and the last menu item unlocks all locked buttons (re-enables them).

Special feature – full screen:

Another important part of this implementation is the full screen mode, which is used to hide the virtual system buttons. On older smartphones there are often hardware buttons, which have to be overridden in order to block their usage. On newer smartphones it is at least possible to hide them partially with the full screen mode. In order to switch back from full screen to normal, the user has to perform a special gesture. This blends the system buttons in as well as the notification bar (see Figure 23).

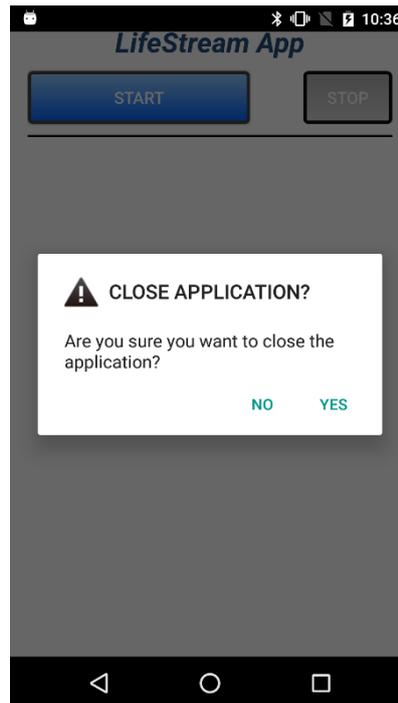


Figure 23: Fullscreen and back button overridden.

Special feature – blocked buttons:

By using the standard Android operating system, it is not possible to override all system buttons, especially the home button is reserved and blocked. In order to override this button, the operating system needs to be changed. Still it was possible to override at least the back button of the application, which asks the user if he or she really wants to close the application (same for menu button if present). The message box, which pops up, can also be seen in the above Figure 23. It should prevent the user from accidental closing the application or at least ask one more time if he or she really wants to terminate it.

Annotations:

Of course this final application and its design is not perfect, though it was already optimized for different types of smartphones as well as operating systems. For example, the menu of the application uses the Toolbar functionality of Android 6.0, which requires a very up to date operating system. Though it is not necessary, because inside the code, the requirements are checked dynamically with annotations⁷. This allows even older smartphones and operating systems to run the application with a slightly changed appearance (base app stays the same).

⁷ Annotations are metadata, which basically means they contain information about a code block for example. They give the compiler additional information about a piece of code or method [30].

3.4 Implementation of the EMD Visualization Server

In the following a short introduction to the server and the website is given. The server acts as connection between the android application and the website clients. Currently the structure of the server, website and android clients is as the following Figure 24 shows.

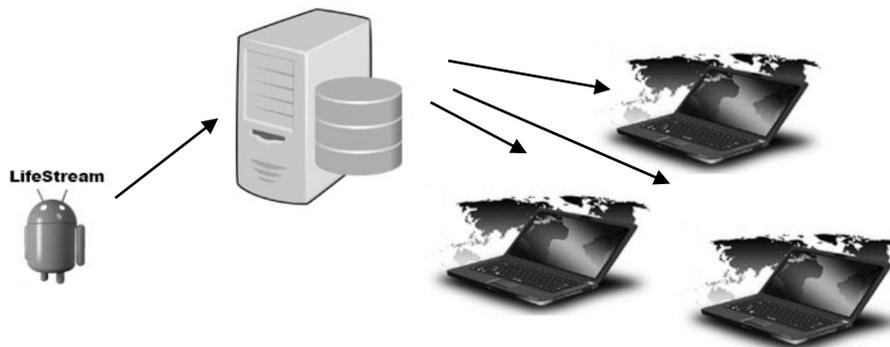


Figure 24: Data transmission schematic.

3.4.1 Server development and setup

The server is able to handle multiple connections, but currently it only processes one android client to many website clients, as shown in Figure 24 above. In a further expanded project the server is able to distinguish between various android clients and PC clients. He is even capable of handling multiple connections now, but the function is not really used. In order to handle the multiple connections, each android client and PC client gets a unique identification number on connecting to the server and can be placed in separate rooms. Possibly the easiest way is to imagine multiple chat rooms, where the PC clients can connect and the chat room theme (the acceleration data) is provided by the android client. So each android client has its own room and only those, who need the information from that specific device, are in this room.

For the server the technology of NodeJs is used (as described in Chapter 2), which allows a bidirectional communication over Websockets. The server is running on a computer of the university of applied sciences St. Pölten in Austria.

The server gets the information from the android phones and processes it further to the website. The data is currently only stored for a short amount of time on the server and always dumped after some time period. Although it would be possible to store the data in a database like MongoDB⁸ (a database system). As for the proof of concept was no need for such a feature, it is currently not implemented, though it is possible to add later on.

In general, the server is programmed for further enhancements. It is easily scalable and can be further extended by a database, which stores the accumulated data. Even multiclient handling is possible with the current setup. Basically this means, that every android client gets its own website and the control operator only sees the information for the actual call and not the call, another control operator is dealing with. The server is also a lightweight system, which runs within the node runtime environment and allows easy usage on other computers, as the whole node framework and server is portable.

⁸ MongoDB is a document-orientated database, which uses no SQL. MongoDB performs extremely well and is easily scalable. It is based on a collection of documents with key/value attributes [31].

3.4.2 Website development and setup

For the website various technologies have been combined as well. The website itself is programmed in HTML and makes heavy use of JavaScript. For the styling plain CSS is used with external resources (mainly for the icons). As support the following two libraries are used:

- **jQuery**
This powerful tool makes it easy to manipulate a page of HTML after it is displayed by the browser. User interaction can be caught, tools can be used to create animations and communicate with a server [32].
- **D3.js**
D3.js is a JavaScript based library, which allows manipulation of documents, based on data. With D3.js it is easy to bring data to life by using HTML, SVG and CSS. In general, it allows binding arbitrary data to a Document Object Model (DOM) and then apply data driven transformations to it [33].

Combining all these resources together resulted in the website, which is available all over the world under the following Domain: <http://lifestream.fhstp.ac.at>

The website is shown to every client, who is connected to the server as usually and not over the Websockets connection and the specific messages, that are shared between the android client and the server. The server distinguishes between normal clients and android clients, who transmit data to it. If a normal client (e.g. computer, smartphone, etc.) accesses the server over the browser, it will receive the website as result. Basically they send a http request and the server responds with the website. After they received the website while an android client is connected and streaming data, every normal client can view the reanimation data.

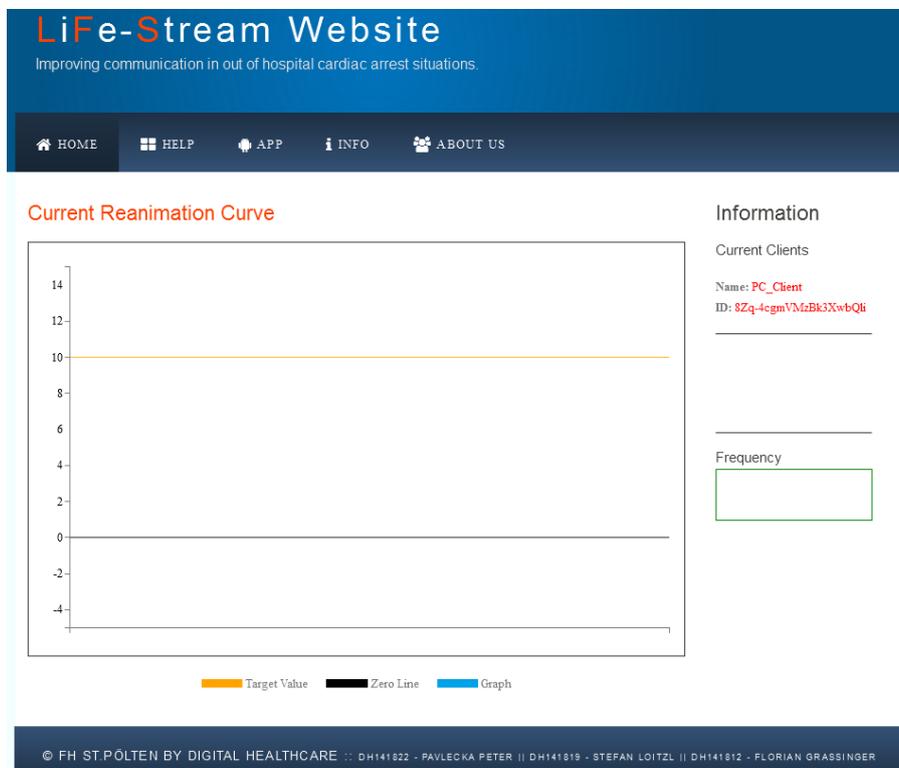


Figure 25: Website design and home page view.

Figure 25 shows the current implementation of the website and the design. It is orientated on the application design and tries to be simple as well as easy to understand. On the main page or home page the user can directly view the updating chart, once an android client is connected as well as the current frequency on the right side of the chart. The other parts of the page are currently not fully finished but have the following purpose and name:

- **Help:**
This subpages purpose is for user questions or information about the reanimation curve or how to read the frequency and data information. It is also intended to be used for asking questions like in a forum.
- **App:**
On this page the user should be able to download the android application, if he has not already. There is also a link to the project on GitHub in order to download it there as well as the source files.
- **Info:**
Here the user can get further information about the general project and the idea behind the reanimation process in time critical situations. The main purpose is to give further information and enlighten laypersons.
- **About Us:**
This page is just reserved for personal information about the LifeStream project members and their contribution to the project as well as the partners of the project.

On the main page the focus point is the updating line chart (using D3.js) and the right information column, which shows the important parameters of the reanimation (further information in Chapter 3.6).

It is important to mention, that only the current data is shown to each client and not the past data, as this would not be necessary. For example, if client A connects at 10:00 am and client B connects at 10:05 am, assuming that an android client is streaming data, then client B will see the current curve of the reanimation from time 10:05 and not from beginning when the android client connected. It would make no sense to show client B the data from the begging of the stream, as it is already in the past and does not correspond the current reanimation curve.

All in all, the website is a simple but effective tool for the visualization. Although the website is not optimized for all devices currently, it is very useful for the dispatcher already. Of course the final project would not include a website for the visualization. The server would stream the data to the headquarters and there to each dispatcher's computer. As they are using a software for the standardized question scheme as well as for the whole process, the visualization of the reanimation data and the calculation have to be done inside the running system.

Therefore, the websites purpose inside this project is mainly to show a possible implementation of the reanimation curve, as well as the calculated parameters and the gained information out of them. For the final project it is necessary to adept the data and the presentation of it to the current system and integrate it. It would be inefficient, if the dispatcher had to open a separate website during the time critical situation.

Once again it has to be mentioned here, that the current implementation of the website is only finished for the main page, as it is the most important part for the prototypic realisation of the project. The above described features of the subpages are currently only partially implemented and can change in the final version of the website. Also the design is currently only a proof of concept and can be further adapted in later versions.

3.4.3 Data transmission

The data transmission is one of the main parts during this project and the thesis, as with the delivery of the data the whole idea stands and falls. If there is no connection to the internet available, the data would be gathered on the smartphone, but could not be transmitted to the server or the website in order to visualize it there. Nowadays nearly everywhere is an internet connection available, but often enough the connection is not appropriate or unstable. Given the situation of unstable and unsecure connections a fast data transmission and small data package size is important. Therefore, the data is packaged into the JSON format, which is a lightweight data interchange format. The format is based on JavaScript and uses two structures:

- A collection of name and value pairs, often referred as objects.
- An ordered list of values, often referred as array.

The charm of JSON is the simplicity and the cross platform usage it guarantees. A message formatted according to the JSON standard is composed of a single top level object or array, where this can contain further arrays or objects [34].

Inside the android application two small JSON object are created before sending them to the server. One includes the information for the chart, the other the calculated parameters out of the raw data. The data is split into two small packages, as the second one does not need to be sent every time, the sensor measures a new value. The server receives the data packages, which include the information every time the sensor updates (fixed point of time, more in Chapter §). The packages contain various information as seen in Listing 1 below.

Listing 1: Default data package.

```
01  {
02      "time" : value,
03      "magnitude" : value
04  }
05
06  {
07      "freq" : value,
08      "freqR" : value,
09      "timeMeasure" : value
10  }
```

The above listed packages contain no real values. The `value` is different in every single package, though the key or identifier is the same. Each package then is read on the server and redirected to the website, where the information is retrieved and visualized.

In order to send the data or stream it from the android client, the Socket.io technology is used (see 3.2.1 Hardware considerations). By implementing the Socket.io technology the android client and the server can communicate in both ways. If the android client has information (the gathered and calculated data), he sends or emits a message to the server with a tag. The server listens for messages with that special tag and further deals with them.

The data is sent over Transmission Control Protocol (TCP), which reliably sends a package to a device on the same or different network. If a package does not arrive at the receiving point, the protocol resends it, until it arrives. This is necessary for the calculated information, e.g. the frequency of the reanimation, as this package only gets sent at a fixed time stamp. The data of course could be transmitted using User Datagram Protocol (UDP), because if a data point is lost, it would not be that harmful. Nevertheless, the data is also sent with TCP, as only a small amount is visualized and nearly each point is necessary for the visualization.

For the server and website communication the same system is used. Once the server receives a special message from the browser, he responds with the webpage. The address of the website contains the special request for the server. Even the website can send information to the server (e.g. the unique identification).

Another important part of the data transmission is the optimized android application. In order to further reduce the resource usage, the accelerometer sensor usage and the background process are optimized. For example, during the application runs a background thread⁹, that performs the calculation on the device and transmits it over to the server. The streaming happens also in another thread.

3.4.4 Visualization

For the visualization on the android side an open library is used, which allows to plot data points in a line graph with some features. For example, it is possible to zoom into the chart and move around there. The used library is called “aChartEngine” and is available under <http://www.achartengine.org/>.

It is a simple charting software for android applications, like mobile phones, tablets, etc. and has various interesting features that allow simple but powerful graphing. Using the charting software, the reanimation data is plotted after the application finishes.

On the website the visualization is more advanced as the dispatcher must be able to get the relevant information out of the graph and the website easily. Therefore, the visualization is separated into two major parts. On the one side is the dynamically updating line chart, which is plotted with D3.js (see 3.4.2 Website development and setup) and constantly plots the acquired reanimation data from the smartphone. On the other side is the information section, which includes information about the registered clients and the reanimation parameters (e.g. the frequency). The following Figure 26 shows an example partial visualization of the website using the algorithm for the study with the application.

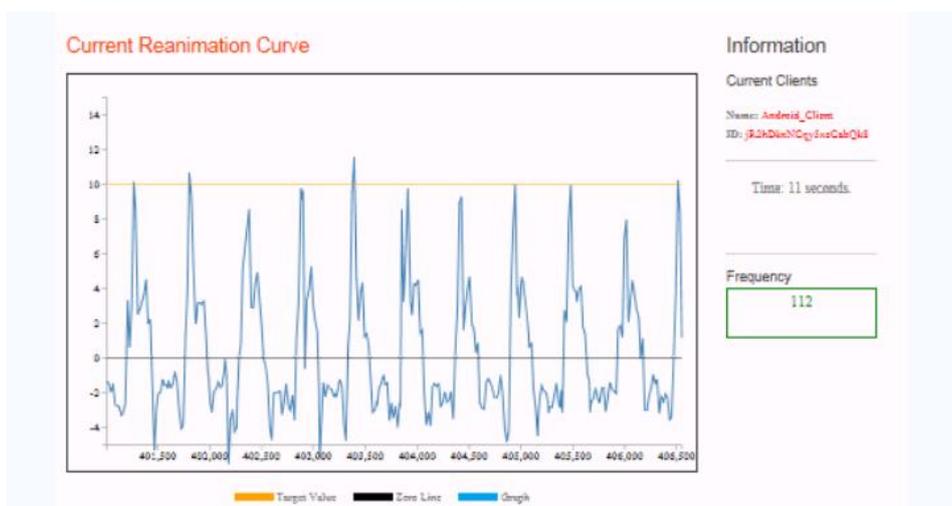


Figure 26: Partial visualization example on the website.

Further explanation for the visualization is given in the following Chapter 4, along with the corresponding algorithm used for it.

⁹ A thread is a portion of code, that may be executed independently of the main program and workflow. This allows the main program to perform other tasks in the meantime.

4 Design and Development of Chest compression depth algorithm Approaches for the Mobile First Responder Client

In this chapter, which is one of the major parts of the thesis, some of the tested algorithms are described. Only the most interesting ones, as well as their integration in the above described application are discussed. Of course there are further algorithms and approaches for the chest compression rate (CCR) and chest compression depth (CCD) detection, but the aim was to compare some of the most promising ones.

Each algorithm is described with the basic ideas and limitations using it and the effect on the CCR and CCD detection. For each algorithm later a comparison is made in Chapter 5, which shows the problems, solutions and limitations of each implementation. Primary to the implementation and testing of algorithms it is important to further investigate the accelerometer, the smartphone and the general physical interaction under the condition of using the low cost accelerometer sensor in smartphones. It is difficult to make a general point for the chest compression rate and chest compression depth detection with smartphones, as every smartphone is different. Not only the operating system is different (e.g., Android, IOS), but also the hardware and the used components. With the aim of creating a widely adaptable application, the further investigation in sensor usage and physics is necessary.

4.1 Accelerometer research

The following subitem covers most of the theoretical concepts for acceleration and the accelerometer in smartphones. Prior a short review of the three major quantities is given, as the whole algorithmic calculation is based upon the physical relation behind these three:

- **Acceleration**
Acceleration, which is measured by the common low cost accelerometer sensor in the smartphone, describes the rate of change of velocity over time. Acceleration happens when an object speeds up, slows down or changes its direction. The acceleration is a vector quantity, which means that it is directional. Directional movement along the x-axis means for example, that the phone is moved to the right or to the left. The direction is indicated with +/- signs and can also be understood as accelerating and deaccelerating of the phone.
- **Velocity**
With velocity the rate of change in displacement is described. Acceleration is also a vector quantity and tells the magnitude and direction the object is moving.
- **Displacement**
This quantity is the major part of the thesis beside the frequency. Displacement is always equal or less than the distance, which describes the total way an object was moving. For the chest compression depth only the displacement is important, because it is not the point of interest how many cm's the reanimating person has moved with the phone. Only the final position is important, related to the starting position, because this can make a statement if the chest compression depth was achieved or not. For example, if the phone is lying flat on chest, which is position A and the layperson pushes the phone, then position B is the point, where he starts to release again the push. The displacement between A and B can be calculated, which results in the chest compression depth (ideally).

Most of the following research results are gathered from various books, like the “Professional Android Sensor Programming” [35] or community sites like stackoverflow or from Google Tech Talk videos. There are numerous theories around distance or displacement detection with smartphones and of course with sensors. Often those theories use a collaboration of multiple sensors, which is condensed under the term “Sensor Fusion”. With Sensor Fusion it is possible to eradicate the weaknesses of each sensor and combine them to an enhanced digital sensor. Digital, as most of the sensors or nearly all are physical like the accelerometer or the magnetometer and combined they result in a kind of imaginary sensor.

During the application development and especially during the algorithm implementation it became more and more clear, that it is difficult to calculate the depth or travelled distance out of the accelerometer (usually it is used only for shake or tilt detection on the smartphone). Of course there are projects that calculate the distance out of accelerometer readings, like fitness trackers, but those use special sensors, which are developed for this purpose. During this thesis only low cost accelerometer sensors are used and no other sensors, as also older phones include the accelerometer sensor. Of course nowadays nearly every smartphone includes a fairly large number of different sensors, like the magnetometer, gyroscope or even location based sensors for position detection. By only using one of the oldest sensors, the accelerometer, it is difficult to make a point about the travelled distance of the smartphone.

Based on the following research results the algorithms are chosen and adapted. Each algorithm has its advantages and disadvantages. There is no general implementation available for this specific type of problem, as there are numerous things to consider, like the sensor, the smartphone or even physical limitations.

4.1.1 General Calculation Idea & Sensor

Physically and theoretically it should be possible to calculate the travelled distance of the phone by using the accelerometer as the following two equations result in the travelled distance. If the acceleration is integrated once, the result is the velocity of the object (in this case the smartphone with the accelerometer sensor). After a second integration the result is the travelled distance [15], [36].

$$v(t) = v_0 + \int_{t_0}^t a(x) dx \quad \text{Eq. (11)}$$

In the above equation the velocity is calculated in relation to time starting with the initial velocity v_0 at the timestamp t_0 . The acceleration $a : [t_0, t_1] \rightarrow R$ indicates an acceleration value for each time step between t_0 and t_1 . The velocity equation is the first part of the whole calculation in order to find the distance, the other consists of the following formula [15], [36].

$$s(t) = s_0 + \int_{t_0}^t v(x) dx \quad \text{Eq. (12)}$$

In this equation s_0 is the distance travelled at t_0 , which is ideally the starting position of the phone. So after integrating the acceleration twice, the result should be the travelled distance or preferably a three dimensional point in space [37].

The following Listing 2 on the next page shows a pseudo code of this simple calculation. The purpose is to show, how fast and easy the calculation could be done, if everything works as intended.

Listing 2: Pseudo code of a simple calculation for distance.

```

01 arrayVel;           //Create array for velocity values
02 arrayDist;         //Array for distance values
03
04 for each sample i do
05     dt = oldTime - newTime    //if constant sampling use 1
06
07     arrayVel[i] = previous velocity + (acceleration +
08         previous accelreation) / (2 * dt)
09
10     arrayDist[i] = previous distance + (distance +
11         previous distance) / (2 * dt)

```

Despite these equations seem fairly simple or at least easy to implement, they are practically not useful and even not possible. The error, which propagates after each integration is horrible as well as some other problems, which are explained in the next subitem.

Previously there is a short introduction to the general accelerometer sensor behaviour together with an explanation of how to interpret the raw data and its format.

Sensorbehavior

Accelerometers are not able to distinguish between acceleration as a result of gravity and acceleration as a result of movement. This results in an acceleration of -9.81 m/s^2 in resting position along the z-axis. So, even if the phone is lying flat on the table, the z-axis reads the gravitational force, which has to be excluded from the further calculation. The measurement of acceleration is performed on the x-, y-, and z-axis (see 2.2.2.1 Accelerometer basics).

Programmatically the values can be accessed over the values array, which is returned by the sensor event [38]. The array contains the following values:

Listing 3: Reading sensor values showcase.

```

01 Sensorevent with „values“ array {
02     float x = values[0];
03     float y = values[1];
04     float z = values[2];
05 }

```

The following Figure 27 shows the measured data of a linear accelerometer sensor. Although it is the x-axis, which is described, the basic idea is the same for the other axes as well as the normal accelerometer sensor.

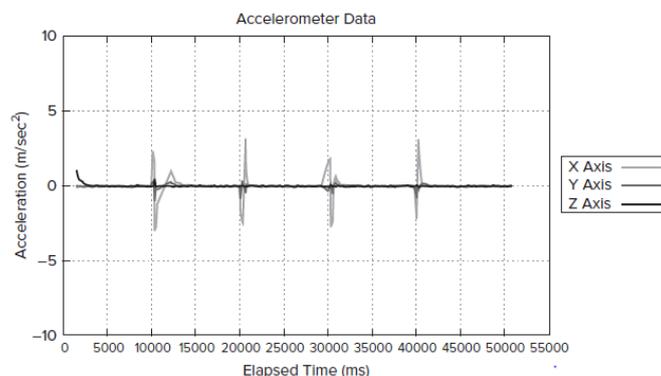


Figure 27: Sensor data from moved x-axis.

As the accelerometer sensor provides the direction of movement (+ and -) it is easy to see where the phone was moved along the x-axis. In Figure 27 a movement from left to right is shown as sharp change in the acceleration value in the positive direction, followed by a sharp change in negative direction. The positive value indicates a force was applied in positive direction (acceleration) and the negative value indicates a force being applied in the negative direction of the x-axis (deceleration). The acceleration happens four times each at 1s, 2s, 3s and 4s, indicated through the peak in positive and negative direction [35].

Linear Accelerometer

As already mentioned, the basic accelerometer is a hardware sensor, which measures the acceleration along three axes. On one axis, the z-axis, the sensor includes the gravitational force, which can be filtered out with various filters like a low pass filter or high pass filter, but can also be avoided directly by using androids LINEAR_ACCELERATION type for the sensor. Though using this sensor type sounds fairly obvious and good, it must be said that it is not as accurate as it might seem, because there are some simple problems, when subtracting the gravity.

The linear acceleration is basically sensor fusion, which uses the accelerometer and orientation to know where the gravity is directed, in order to remove it. The sensor delivers data, that was processed by a high pass filter, so that the gravity or any other slowly changing acceleration cannot pass through the filter. To compute the linear acceleration from device to device, various sensors are used. Some devices use the accelerometer plus the magnetometer and others use the accelerometer plus the gyroscope [39].

Sensor delay

Another important part of the android accelerometer is the sensor delay. The sensor delay is used to adjust the sampling rate of the sensor. There are four sensor delays available in standard android, which set the respective sampling rate for the sensor:

- SENSOR_DELAY_NORMAL (200 microseconds delay)
- SENSOR_DELAY_GAME (20000 microseconds delay)
- SENSOR_DELAY_UI (60000 microseconds delay)
- SENSOR_DELAY_FASTEST (0 microseconds delay)

So, for example, if the sensor acquired the data with a sampling rate of SENSOR_DELAY_NORMAL, it would deliver values at 200 Hz, so every 5ms. The value of 5ms results of the fact, that the frequency f is the reciprocal of the period T [40, p. 157].

$$f = \frac{1}{T} \rightarrow T = \frac{1}{f} \quad \text{Eq. (13)}$$

The sensor delay can also be adjusted manually, which would greatly enhance the algorithm and the algorithmic calculation in the following implementations. There is an issue though with the sensor rate, as the delay, which is specified, is only suggested to the android operating system. The android operating system and other running applications, like phone calls, can alter this delay, which results in slight changes for each sampling rate. These changes can accumulate up and contribute largely to the calculation error.

In the algorithms, the delay was adjusted to each smartphone, because as primary researches with various smartphones have shown, each smartphone reacts differently to the fixed or given sensor delays. According to the real sampling rate, which slightly varies from the adjusted one, the algorithm is programmed (most of the algorithms heavily rely on the sampling frequency).

4.1.2 Issues, Limitations & Hints

Along all the features and possibilities of digital or analogue accelerometers come various limitations or issues. Some are mentioned already during the other topics of this thesis, but the major issues will be described in more detail now. Also some hints and ideas for future purpose are given, though most of them are used during this project and thesis.

The main problem upon using the above mentioned equations is, that accelerometers are bad at dead-reckoning (continuous position determination). Accelerometers have some noise which varies from smartphone to smartphone as each has its own manufacturer and device type. Theoretically the equations could be used to determine the position of an object, in this case the smartphone in a three dimensional space. In practice the function of $a(t)$ cannot be ascertained as the acceleration values are measured at discrete moments. This means, the sensor delivers a value at a fixed time stamp, which results in a time series of a sequence of quantities. As a result of this behaviour the normal formula for calculating the distance out of the acceleration can only be applied piecemeal. The results of each piece are added up in order to determine the final position. The easiest way is to assume, that the acceleration a is constant during a time interval of t_i and t_{i+1} .

Another issue of analogue and even digital accelerometers is, that they have some noise. The noise can be filtered of course with various filter types (explained in the following subitem) but normal accelerometers produce raw data, which is not filtered or smoothed. This noise will normally result in a non-zero mean, if the noisy values are used in the formula. So when integrating the acceleration signal, the non-zero mean is continuously added and accumulates in the resulting velocity signal and later of course in the distance integration. This behaviour is called sensor drift, as the integration starts fairly well, but quickly accumulates the errors and the resulting values drift away.

Also considered must be the usage of the Pythagorean theorem in order to compute the value or magnitude of the acceleration. As the direction of acceleration is not relevant (it does not matter if the device is accelerating or deaccelerating), the following formula should be applied to the raw values, before the integration or further proceeding happens.

$$\text{magnitude} = \sqrt{x^2 + y^2 + z^2} \quad \text{Eq. (14)}$$

By using this equation, also the facing of the smartphone is irrelevant. So if the reanimating person is putting the phone face down on the chest of the reanimated person, the values still will be positive, because of squaring. For a better imagination, why the above formula should be used, a driving car can be imagined. What if, the car is accelerating with $5m/s^2$ for 1s and then suddenly deaccelerates with $5m/s^2$ for 1s – the change would be zero. However, if the value of acceleration is used, something else is computed, as the value would not be negative.

One more obstacle is, that unless the device is accelerating or deaccelerating at all points in time, a constant non-zero velocity and a constant zero velocity will both contribute nothing to the double integration. Therefore it is impossible to tell a non-zero velocity from a zero velocity, and therefore a calculated distance is in fact useless. For example, if acceleration is measured on a device lying flat on the table, it will measure $x = 0, y = 0, z = -g$, where g is the constant acceleration due to the gravity (if no linear acceleration sensor is used). If the measurement is done while the device is traveling at a constant speed of 10 meters per second for example in the x, y or z direction it will also measure $x = 0, y = 0, z = -g$ because it is technically not accelerating. The device is traveling at a constant speed and therefore not accelerating – so the part of the travel, where it moves at constant speed, is not contributing to the integrated measurement [35].

4.1.3 Possible Solutions & Filters

As the problems are clear and described, there have to be solutions in order to avoid some of the issues or limitations. Of course, many more than the following described solutions exist and some even are not that useful for the specific problem in this thesis. The major problem of different sensor types, some measure more accurate, some not, is hard to overcome, as every manufacturer has his own implementation. If all accelerometers would have nearly the same standard, the problem could be solved by the root. Anyway, if not there, it is still possible to enhance or standardize the output in some ways. The following methods shall give only a short introduction and will not be further explained in detail, as this would consume too much time. The presented methods are heavily inspired by the fantastic book “Professional Android Sensor programming” by Milette & Stroud [35].

In order to remove the noise, as well as the gravity, often filtering is used. With the filters described below the raw values can be smoothed and the gravity factor is removed. However, it is better to use the linear accelerometer of the android system (the fused sensor), if present and available, as here the gravity is already removed and the values are much smoother. After the gravity is removed or the values are read and filtered with a respective filter, it is best to calculate the magnitude of the acceleration values (see 4.1.2 Issues, Limitations & Hints) before continuing with further calculations. The reason is, that it is not necessary in this thesis to know the direction of the accelerating smartphone, because the reanimating person could place it slightly crooked or facing down and if the algorithm only relies on one axis, the results would be distorted. This allows also to set a threshold for a minimal acceleration in order to exclude an incidental acceleration by a person who bumps the phone. Only real acceleration, which is produced by a shake or push on the smartphone needs to be detected.

Generally, it is best to filter output from individual sensor readings or fuse the results from multiple sensor. Even if sensor fusion is used, it is good to filter the results further or work with other techniques to reduce the errors. First the filters for specific problem types will be described, then some other techniques to further enhance the output of the sensors (either single or fused ones).

Before the filters are described, a short list of possible error types is given for a better understanding (most of them have been covered already in the previous sub items):

- **Systematic errors**
Systematic errors are constant changes in accuracy or precision¹⁰ of the sensor due to external or internal influence. For example, if the phone is placed near a magnet, the sensors, that rely on magnetic fields, are disturbed.
- **Noise**
A noise is a random influence on the generated sensor value.
- **Drift**
A drift describes a slow wandering of the data away from the real world value. The best example here is the double integration of the acceleration values, which drifts away due to accumulation of errors.
- **Offset**
If the output is not equal to zero, when the measured property should be zero, the sensor has an offset.
- **Time delays**
As android is a no real time operating system, some values can be delayed, which results in incorrect timestamps – one of the major problems for the algorithms.

¹⁰ The actual value the sensor tries to measure and the one it measures are often different. High accuracy means the value is close to the actual one, where high precision means the values are clustered around a particular value, whether it is the actual one or not [35].

The biggest issue is still the integration error and the accumulation of it during the integration. Like the acceleration example shows, the error will grow exponentially even if the device is not moving, as the sensor continuously produces drift over time and includes noise, which adds up to the final result [41]. In order to reduce this and the other errors filtering is a good approach. Following filters are commonly used for filtering various types of sensors. During this thesis they are used in some of the algorithms in order to filter the raw accelerometer data.

Low-Pass Filter

Low pass filtering is, as the name suggests, a filter, that lets low frequencies pass and cuts off high ones (based on digital signal processing theories). This filter type is ideal for eliminating noise above a certain frequency. The implementation of this filter can be found in the following sub item (4.2 Implemented Algorithms) as well as the other filter implementations in java. With the α the filter can be fine-tuned in a range of 0 to 1. On the one side, if alpha is very small, the effect of the filter is small, but the changes in the input are allowed to build up quickly. On the other side, if alpha is close to one, the effect of the filter is large and the input is allowed to build up very slowly. For this filter the desired time constant and sample rate should be known, because this allows to pick an appropriate alpha value. As the sensors in Android treat the requested sample rate only as suggestion, it becomes hard to actually fix a sample rate and time constant (however the approximate rate is often good enough) [35], [42].

High-Pass Filter

Again, as the name suggests the high pass filter lets high frequencies pass and reduces the amplitude of signals, which have a frequency that is lower than the threshold frequency. A high pass filter can be used to measure the real acceleration of the device, as he eliminates the contribution of the force of gravity. Again the implementation of a simple high pass filter in java can be found in the appendix of the document.

Simple moving average (SMA)

The moving average provides a better smoothing against single data point spikes, that occur due to drift or noise. The SMA is often called the running average, as it finds the arithmetic mean of the most recent k values in a set of data, that continuously increases. The variable k stands for the size of the “window”, which is basically the area that is averaged. This method, though needs at least the first k values to work and for the first values normally zero is used. As the values are collected fairly fast with the accelerometer sensor, this little disadvantage will not take into account that hard. The implementation allows to push new values to the array of already collected values and continuously returns the average.

A few things are to note at this algorithm, because if the window size is large, a sudden change in data values may take too long to be seen in the moving average and the calculation cannot be done, until there is sufficient data available. On the other side the smoothing parameter or window size must be chosen large enough to adequately smooth the data.

Kalman filter

This filter can provide excellent signal processing results, but is complicated to implement for even the simplest examples. The algorithm is fed with noisy measurements, some predictions about how the measurement’s true value is behaving and some information about the forces, that are causing the system to change. This filter type is extremely flexible and can be used to smooth high-frequency noise or to isolate a periodic signal. If functionality beyond simple smoothing and simple high-pass filtering is needed, a Kalman filter will give the best result [43].

Re-zeroing

This simple but effective method can be used, if there is an offset present, that is affecting the application. During the algorithm development this method is used as the user has to lay the smartphone flat on the chest of the victim, which is the starting position. By using the re-zeroing it is possible to find an initial zero point, which is important for the algorithm or further calculations, like the displacement. If no re-zeroing happens, the smartphone would produce values around zero, as the chest of the person is not totally flat. This little offset accumulates over time and heavily increases the overall error of the calculation.

The re-zeroing is some kind of calibration, which needs to happen before any further calculation is performed. Some may think it would take a long amount of time, but the accelerometer is fast enough in reading values, that the required amount is easily collected in not nearly half a second. The implementation will be explained during one of the algorithms in the following sub items.

4.1.4 Research Insights

Based upon former researches and a lot of trial and error with various accelerometer applications and implementations, a short list of important insights was created. The listed points include a software design consideration for the algorithms [38].

1. As the accelerometer signal is usually not noise free it has to be filtered. A possible filter could be the moving average, like the low-pass filter. The processed value is the result of averaging a certain amount of samples.
2. Even with previous filtering the data can contain some errors, either due to mechanical noise or any other disruption. Therefore, it is often useful to implement another filter after filtering the raw data. Depending of the filtered data samples, a window is produced, which contains the real data samples (the simplest window cuts off all measured values above and under a certain threshold, e.g. all values smaller than -2 and greater than $+2$).
3. It is good to specify a no movement state by using the re-zeroing method in order to calibrate the phone for a starting position. The calibration routine is done upon starting the application and specifies the starting point (ideally this is the chest of the reanimated person).
4. The sampling frequency should be as high as possible, ideally the fastest available, which is normally 0ms delay. Although the algorithm has to be adjusted specifically to the real sampling frequency, as android takes the provided sampling frequency variable only as suggestion and the actual one is often different. A faster sampling rate implies more accurate results, as the error is reduced by using much more information.
5. The time between the samples must be the same, which is important for the algorithmic calculation. Best to use is a fixed time interval, in which the current acquired value is passed to the calculation.
6. Using the linear accelerometer is better for this kind of application, as the computation time is kept low. The implementation of android performs the filtering of the gravity and returns a nearly zeroed value for all three axes of the accelerometer.
7. Further noise reduction could be achieved by using sensor fusion with other sensors, like the magnetometer or the gyroscope. Even other sensors, like the GPS or the location service of android could be used for further enhanced results and position determination – generally contradicting to the thesis concept, as only the accelerometer low cost sensor should be used.

4.2 Implemented Algorithms (self-developed and existing)

This part or subitem of the thesis is one of the major ones, as it includes some of the tested algorithms as well as the one, which is used during the field studies. Of course there are many more ways to get the location of the smartphone as well as the frequency in which it is moving, but the aim is to use the low cost accelerometer sensor, which is available in nearly every smartphone nowadays. Each algorithm is first described theoretically and later on the implementation is shown in a short listing or pseudo code.

In general, the basic application stays always the same, only the data calculation and further processing from the point, where the accelerometer raw data is gathered, will be described. Also the transmission format to the server will be shown, or which information is transmitted. The most interesting part is the calculation itself, which will be analysed for each shown algorithm. For each implementation various fix parameters are defined, which are often adapted in each subsection of this point (in each algorithm). The problems during each experiment and the results will slip into the next experiment. Each further test gives more and more information about the overall process behind the distance and frequency detection. The following Figure 28 shows the basic structure for each presented experimental application:

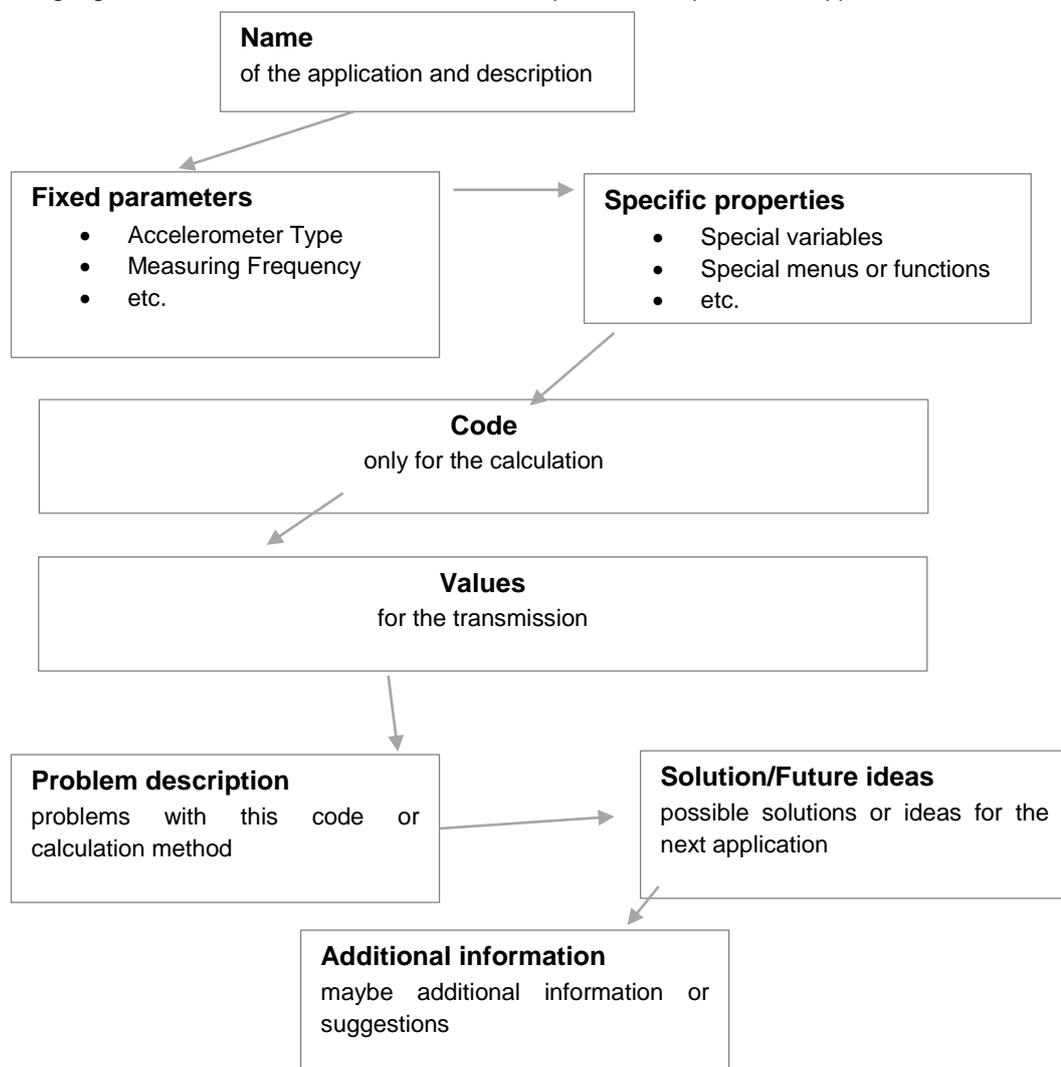


Figure 28: Flow of the algorithm description in the following part of this thesis.

4.2.1 Experimental App: Filtering & Integration

The following explained implementation of an algorithm tries to get information about the chest compression depth, as well as the frequency, by using two simple filters as well as a slightly changed double integration of the acceleration. This algorithmic implementation is inspired by the following fantastic article “Development of Android Based Chest Compression Feedback Application Using the Accelerometer in Smartphone” [21], which also has already been mentioned in the related work. Unfortunately, it was not possible to fully reproduce the described results and the algorithmic implementation, as some accelerometers use an additional pressure switch in order to detect the starting point of each compression. Regrettably, this switch is not implemented in normal smartphones. A chest compression motion is a recurring action, where only the peak-to-peak distance is the only valid information. Peak-to-peak is from positive to negative, as this represents theoretically a push with a release afterwards.

As already mentioned, accelerometers are not very handy and often produce noise, as well as drift, which should be removed before further calculations are performed. Therefore, a low pass filter was applied to the raw data in order to remove the noise and later a high pass filter for the drift.

Before any further explanation can be done, the used hardware has to be mentioned. The phone used is a Google Nexus 5. In order to determine the real sampling frequency, the application “[Sensor Rate Checker](#)” was used, which reads out the real sampling frequency based on the given measuring parameter. Of course the sampling rate changes also on using the phone and background tasks, but it is possible to see at least the average offset from the real sampling rate. Some parameters have to be mentioned for this algorithmic implementation, as well as some special parameters.

Fixed parameters:

IP = “91.219.68.209”	This stands for the address of the server at the university of St. Pölten.
Port = “80”	With this variable the port of the server is described.
TYPE_ACCELEROMETER	Upon using this parameter, the normal accelerometer sensor is used for the data acquisition, which still contains the gravity.
SENSOR_DELAY_FASTEST	In order to acquire more data points for the filtering and calculation, the fastest sampling rate is chosen. Though 0ms is suggested to the operating system, it will not directly perform the sampling at this time.

Special properties:

$\alpha_1 = 0.3$	By using such a low value for α_1 , it is possible to enhance the smoothing, as the noise of the accelerometer was very high. This variable is used for the low pass filter.
$\alpha_2 = 0.8$	α_2 describes the smoothing parameter for the high pass filter, which is closer to one, as the drift of this specific sensor (in the used smartphone) did not have that much drift.

Code:

The following code snippets are used during calculation and rely heavily on previous parameters, like the gathered acceleration values or arrays, to store them. In order to implement them in another application, it is necessary to also register the sensor as well as to allocate the appropriate arrays for storing the data. Within this application the calculation process was mainly done in the `OnSensorChanged` Method, which is fired every time the sensor reads new data or acquires a new value. The values are stored in the `event` variable, which is provided by the method. The first code part in Listing 4 describes the low pass filter, which is used during the calculation.

Listing 4: Low pass filter implementation.

```
01 private float[] lowPass(float x, float y, float z, float a) {
02     float[] filteredVal = new float[3];
03     filteredVal[0] = x * a + filteredVal[0] * (1 - a);
04     filteredVal[1] = y * a + filteredVal[1] * (1 - a);
05     filteredVal[2] = z * a + filteredVal[2] * (1 - a);
06     return filteredVal;
07 }
```

After filtering the raw data, the values are processed further to the high pass filter, seen in Listing 5, in order to remove the drift of the phone. Again an example implementation of the high pass filter is shown in the Listing 5 below (other implementations are possible).

Listing 5: High pass filter implementation.

```
01 private float[] highPass(float x, float y, float z, float a) {
02     float[] filteredVal = new float[3];
03     float[] gravity = {0, 0, 0};
04     gravity[0] = ALPHA * gravity[0] + (1 - a) * x;
05     gravity[1] = ALPHA * gravity[1] + (1 - a) * y;
06     gravity[2] = ALPHA * gravity[2] + (1 - a) * z;
07     filteredVal[0] = x - gravity[0];
08     filteredVal[1] = y - gravity[1];
09     filteredVal[2] = z - gravity[2];
10     return filteredVal;
11 }
```

Upon using the high pass filter, it is possible to remove the gravity, which is factored out of the values. By using the linear accelerometer, it would be possible to work with already filtered values, but during this algorithm implementation it was intentional, as the filter methods should be evaluated.

When both filters have been applied, the calculation is performed. During the `OnSensorChanged` event the above shown filters are called and applied to the raw data. The values are then stored in an array whereby then further calculations are performed. As the sensor acquires multiple values, once the `OnSensorChanged` event is fired, it is possible to perform the calculation inside the function (at least theoretically). Normally it would be more practical to wait for some values or to gather them, until enough are there in order to further process them, but the sensor reads a lot of values, once the event is fired. The following Listing 6 below shows the main calculation method, which is performed every time the sensor acquires some new values.

Listing 6: Main calculation method upon acquiring new sensor values.

```

01 public void onSensorChanged(SensorEvent event) {
02     if (started) {
03         time = System.currentTimeMillis(); //Timestamp of event
04
05         if ((time - timeOld) > getNumberOfMeasures()) { //Proceed
only every X seconds
06             dT = (time - timeOld) / 1000f; //Milliseconds to seconds
07             timeOld = time; //Old time update
08             timeCount++; //Counting to seconds
09
10             float[] values = event.values; //Copy the raw values
11             accelValsLow = DataManipulation.lowPass(values[0], values[1],
values[2], 0.3f); //Apply low pass filter
12             x = accelValsLow[0];
13             y = accelValsLow[1];
14             z = accelValsLow[2];
15
16             accelValsHigh = DataManipulation.highPassFilter(x, y, z,
0.8f); //Apply high pass filter
17             x = accelValsHigh[0];
18             y = accelValsHigh[1];
19             z = accelValsHigh[2];
20
21             magnitude = (float) Math.sqrt(x * x + y * y + z * z);
22             calcValueStack.add(magnitude); //Add the magnitude to list
23             float[] calcAccelVals = {x, y, z};
24
25             //Perform the calculation
26             for (int i = 0; i < calcAccelVals.length; i++) {
27                 totalAccel[i] += calcAccelVals[i];
28                 velocity[i] = initVelocity[i] + (totalAccel[i] * dT);
29                 distance[i] = (initVelocity[i] * dT) + 0.5f *
totalAccel[i] * dT * dT;
30                 initVelocity[i] = velocity[i];
31             }
32
33             //DETERMINE If distance greater than starting position and see
if 5cm were traveled
34             ...
35     }

```

The above Listing 6 shows the major part of the code containing the approach to find out the displacement of the smartphone. The code was interrupted at line 34 as there would have been a simple metric comparison between the starting position of the phone and the final position, which is stored in the 3-dimensional array `distance`. The name `distance` may be misleading, as it contains theoretically only the final position or actual position of the phone, but the name was chosen before the distance was calculated. On line 05 an if-statement is used in order to calculate at a fixed timestamp. This is needed, as the calculation should have discrete time stamps. It is also used in order to count up to seconds, which is necessary for the visualization, but not for the algorithm (therefore not shown).

After both filters have been applied, the resulting values are used for the calculation. Each array in this calculation process starting at line 26 and ending at line 31 is 3-dimensional. At first the total acceleration is counted up, which never gets erased and increases by the time [44]. This is the special modification of the base algorithm:

$$s = v_0 \cdot dt + \frac{1}{2} \cdot a \cdot dt^2 \quad \text{Eq. (15)}$$

By tracking the previous acceleration of all 3 axes, it is possible to make each following measurement integration based on instantaneous acceleration or change in acceleration. Theoretically the values in the array `totalAccel` should go from zero to a max value and then back to zero upon moving the device. Ideally equal parts are added due to acceleration and deceleration of the smartphone. A short explanation to this idea is, that if the integration is performed without including the previous acceleration, this will end up with a value that does not include all the information [44].

As the resulting values in the `distance` array are in the unit meters, a conversion to centimetres (multiplying by 100) must be performed in order to make a declaration about the compression depth. To calculate the frequency, the occurrences of “right” pushes are counted, which are recognized, once the distance is greater than 5cm. The counting continues for a specific amount of time, for example 5s. After these 5s the number of correct pushes is counted up to 60s in order to check, if the frequency is right. Once again the right frequency according to the ERC standards would be 100 pushes in 60s. [10] The following formula is used on the website, after the specific amount of time has passed:

$$\frac{(\text{correct pushes} \cdot 60s)}{\text{time in s}} \quad \text{Eq. (16)}$$

For example, if the time is 5s, as mentioned above and the number of correct pushes in this time is eight, the resulting frequency for 60s would be 96, which is already pretty good and near the perfect amount of 100 pushes per minute.

Values:

For the visualization on the website the above described code needs to transmit the actual distance data as well as the frequency and the time. For the time in order to be displayed appropriately the code implements a simple if-statement, which counts up to 1s total. The sensor triggers for example every 5ms, which can be adjusted or read out at the beginning of the application. So once the sensor triggered 200 times and one second has passed, this information is sent to the server, where further processing happens. Listing 7 on the next page shows the transmission of the values. From line 01 to 08 the time counting is performed, which is of course not very precise, but gives some pretty good results, which are close to real world values. In this example the website updates every 5s, so once they have passed, the frequency based on the above mentioned formula is calculated.

Line 13 creates a JSON object, which contains the filtered z-values of the smartphone, in order to visualize the moving curve of the phone on the website. The z-values are used, as the user normally pushes the phone away from him and towards him, once he performs the reanimation. Of course, this is not the depth, but it can give at least a hint about how hard and fast he pushes. For this visualization it was enough, in order to send the results to the website. On line 14 another JSON object is created, which contains the information about the distance, as well as the frequency, once it has been calculated (every 5s), as well as the time that has passed.

Listing 7: Time dependent transmission of the sensor values.

```
01 if (timeCount == 20) { //1s passed
02   measuredTime += 1; //after 5s update on server
03   timeCount = 0; //reset counter
04 }
05 if (measuredTime == 6) { //after 5s (6 as we start from 1)
06   frequencyReal = (frequency * 60) / (measuredTime - 1);
07   measuredTime = 1; //reset time count
08 }
09
10 //Data gets sent over the socket to the server
11 //-----
12 try {
13   socket.emit("SensorData", new JSONObject().put("time",
time).put("push", z));
14   socket.emit("AdditionalInformation", new
JSONObject().put("freq", frequencyReal).put("dist",
distanceDiff).put("timeMeasure", measuredTime));
15 } catch (JSONException e) {
16   e.printStackTrace();
17 }
```

Problem description:

By using this experimental algorithm, it was possible to detect that accelerometers and the smartphones do not always perform as intended. As already mentioned several times, it is nearly impossible to adjust the sampling rate to a fixed amount, as the provided value is only a suggestion for the android operating system. Of course this algorithm or implementation could be more refined, especially in the calculation in the `OnSensorChanged` method or the filters. Also the general integration is problematic, as the algorithm should work for at least ten minutes, but the values drift away already fairly fast, even by taking the total acceleration into account, which should reduce the error, as well as the two applied filters. Accelerometers are generally very precise, but bad at dead reckoning [44]. In order to get more optimized values, it is good to use sensor fusion, or more precisely, the linear acceleration of the smartphone. By filtering the raw values from the normal accelerometer, the drift and noise can be reduced, but not as well as by using the already implemented sensor fusion of the android operating system. The problem is, that not all older phones are able to perform the sensor fusion, as some of the sensors are missing for example.

Future ideas:

As the above shown code is only an example of an implementation, there are no real future plans for this application. Also the approach with total acceleration taken into account is not really helpful, as the phone has no real starting position and always a different initial velocity. Of course this method could work, but based on the research and given hardware, it was not possible to achieve the wanted results. For the next applications the lessons learned have been, that every accelerometer and smartphone is different. Also the calibration at the beginning, e.g. a starting point or continuous recalibration is important in order to maybe reduce the drift or errors produced by the smartphone. The noise still contained in the data adds up fairly fast, as the noise has a non-zero mean, which continuously adds to the velocity integration and distance integration.

4.2.2 Experimental App: Re-calibration, Speed & Magnitude

In contrast to the algorithm approach described previously, which deals with the double integration and some mechanics to filter the raw data, the following algorithm is based on the speed and the magnitude of the acceleration values. This means, no double integration is performed, which should theoretically at least reduce the possible errors, though the raw values of the accelerometer are still very noisy and drift away.

It is important to mention, that speed in this context does not mean the actual physical speed, which can be calculated by integrating the acceleration values, but rather the speed, that is produced between two accelerations, once the device is shaken. Basically this algorithm approach is a simple shake gesture detector, but much more adapted to the current reanimation problem. The idea behind this application is, that if the reanimating person is pushing the phone as hard as he or she can, the speed or the acceleration will drastically increase. This increase in acceleration between two measured acceleration values can indicate either a push or release with the phone. Of course by using this method, it is impossible to make a point about the distance travelled with the phone, but the major idea was to find the right frequency. If the reanimation guidelines were ranked in a ranking system, the right frequency would take the 1st place. Therefore, the major focus was on finding the right frequency for this approach and only count it once the reanimating person pushes hard enough. So according to numerous primary tests with the application and calibration of the used testing device, a Nexus 5, it was possible to develop at least another way for detecting not the depth of the reanimation, but maybe the frequency.

In order to measure the power of the shake, not only two values could be used, as the accelerometer sensor reads much too often and maybe the user is still pushing further, but the calculation already thinks that this is the maximum. If only the absolute difference between two acceleration values (at two timestamps) were taken, a local maximum could be reached and a global one neglected, as maybe the user is pushing further.

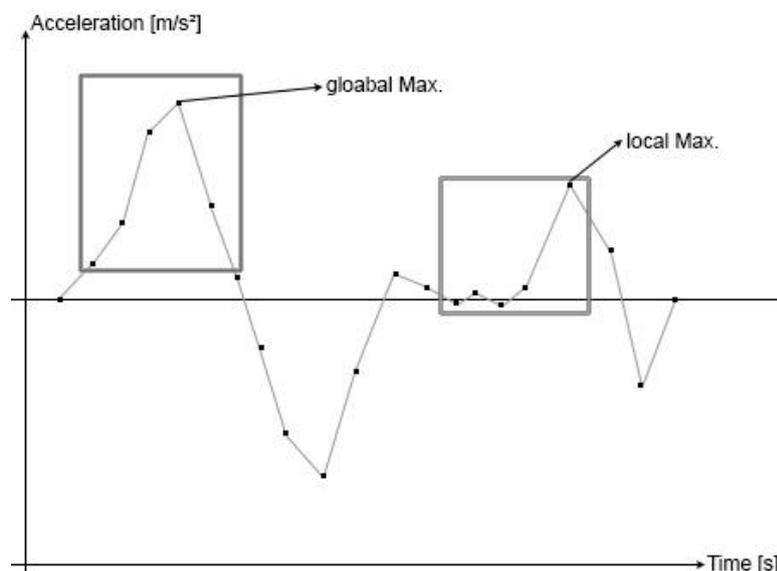


Figure 29: Maxima of acceleration function for one axis.

The Figure 29 above shows schematically the acceleration along the x-axis with time and the local maximum and global maximum. Of course, the same applies to the negative direction of the acceleration values (local and global minimum).

So the speciality of this algorithm implementation is the recalibration of the sensor or the values and the usage of an average measurement of n measured acceleration values.

Each metric distance between two acceleration points at time $t_i - t_{i-1}$ is calculated and saved into an array. After n of those metric distances have been collected, the average of the sum is formed and compared to a pre-defined threshold. If the value exceeds the threshold, the push (which is represented by the n values in the array) is counted. The threshold and the number of averaged values can be adjusted programmatically, but were chosen due to primary research and tests with a reanimation phantom and two professional paramedics. According to these tests, the values have been adjusted for the given smartphone. First the major fixed parameters are shown again, which have not changed since the last application but are necessary.

Fixed parameters:

IP = "91.219.68.209"	This stands for the address of the server at the university of St. Pölten.
Port = "80"	With this variable the port of the server is described.
TYPE_LINEAR_ACCELEROMETER	Upon using this parameter, it is possible to directly get the filtered raw values from the accelerometer. This sensor is a fused sensor, which means, it consists of several other sensors in order to process out the drift and noise. By using this sensor, the minimum operating system version must be higher, as it is a newer implementation.
SENSOR_DELAY_FASTEST	Again this parameter is used, as the reading should be as fast as possible and the timestamps are controlled programmatically. Though 0ms is suggested to the operating system, it will not directly perform the sampling at this time.

Special properties:

pushThres = 1.2	This value indicates the minimum force, that has to be applied to the phone in order to detect the shake or gesture as a push. The value starts at a pre-defined number (according to primary research) and gets dynamically adjusted during the application runtime. After a specific amount of measurements have been taken, the value is recalibrated dynamically during the runtime. The intention behind this recalibration is to adjust the calculation algorithm to the user, as some users may push harder or weaker than others.
timeThres = 500	With this value it is possible to adjust the amount of time the user has to perform the push. The default is set to 500ms, but can be adjusted before the application is launched.
magnitudeThres = 5	With this threshold the other functionality of the algorithm is checked, because it was necessary to also include the general movement. So every time the user pushes harder than this value a parameter is counted up. This is used to detect if the user is moving the phone in any way.

Also new in this application and algorithm is the menu, which allows the user to adjust the parameters of the calculation prior to the launch. After the application has been launched, it is impossible to change the parameters until finishing it. Of course the menu is not intended to be used by a layperson, but it allows far more freedom while testing the application.

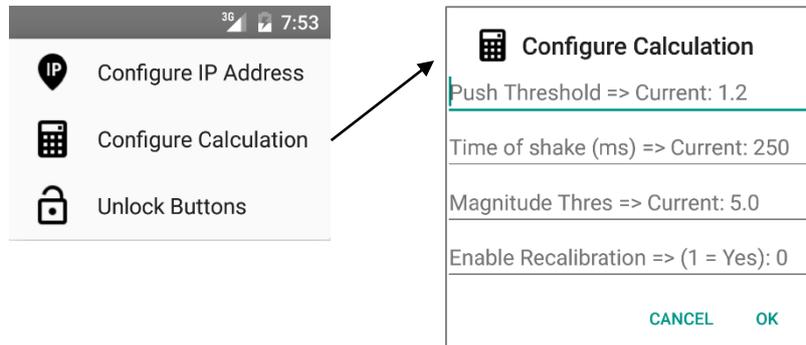


Figure 30: Application menu and algorithm calibration submenu.

Figure 30 above shows the general application menu, which is still the same as in the normal application, but also has a specific submenu, that allows more customization for the user. The menu or input fields always show the current value as a hint. By switching the last menu point it is possible to control whether the recalibration of the push threshold should happen or not.

Code:

Again the following code parts are taken out of the full application and need the whole environment to function appropriately. The major parameters are described separately as well as the arrays that are important. In the first code snippet the recalibration function is presented.

Listing 8: Recalibration method for the pushes.

```

01 private void recalibrateCalculation(ArrayList<Float> list) {
02     float sum = 0.0f;
03     float mean = 0.0f;
04     int count = 0;
05     int countThres = 0;
06
07     for(Float f : list) {
08         sum += f;
09     }
10     mean = sum/list.size();
11
12     for(int i = 0; i < list.size(); i++) {
13         if(list.get(i) > mean) {
14             count++;    //Count amount greater mean
15         }
16         if(list.get(i) > pushThres) {
17             countThres++;    //Count amount greater threshold
18         }
19     }
20
21     if(count + list.size * 0.1 > countThres) {
22         pushThres = mean + 0.2;    //Adjust threshold
23     }
24 }

```

Listing 8 above shows the recalibration method for the algorithm, which needs a list of values, where the new mean value is calculated from. The list of values contains values, where each value stands for the average of n acceleration distances (difference between an actual and a previous one). The list consists for example of 200 values, which have been calculated over the past few seconds. First the function calculates the sum of all values and later the mean, before proceeding in the lines 12 to 18 with the counting part. The first if-statement counts the number of values, that are greater than the mean value of all values. The second if-statement counts the number of values, which are greater than the actual defined push threshold. Later those two numbers are compared in line 21, where a little offset is added to the first count of values in order to neglect the standard derivation. If the statement evaluates to a true value, the new push threshold is adjusted by adding an offset to the pre-calculated mean of values.

The intention is to dynamically react to the pushing behaviour of the reanimating person. For example, the person is reanimating not hard enough, but the frequency would be good and the algorithm says the frequency is not right as the push is not deep enough, it would sophisticate the result. Even if the person is not pushing hard enough, the frequency, if it is right, is often half the battle. Sometimes the people also change their pushing habit during the reanimation, as they push one minute harder and the other minute softer. In order to detect the movement in any situation the idea is to adjust the values on the run. The following code shows the implementation of the algorithm and the usage in the running program.

Listing 9: Major part of the algorithm is described.

```

01 public void onSensorChanged(SensorEvent event) {
02     if (started) {
03         time = System.currentTimeMillis(); //Timestamp of event
04
05         if((time - timeOld) > getNumberOfMeasures()) {
06             timeOld = time; //Old time update
07             timeCount++; //Counting to seconds
08
09             x = event.values[0];
10             y = event.values[1];
11             z = event.values[2];
12
13             if(valueStack.size() > (int)getNumberOfMeasures()) {
14                 valueStack.remove(0); //Remove old ones
15             }
16
17             float speed = Math.abs((x + y + z) - (xOld + yOld +
18 zOld));
19             valueStack.add(speed);
20
21             xOld = x;
22             yOld = y;
23             zOld = z;
24
25             float sumOfValues = 0.0f; //Store sum of speed
26             float avgOfValues; //Store average of speed
27
28             ...

```

```
...
26
27     for(float f : valueStack) {
28         sumOfValues = (sumOfValues + f);    //Find sum
29     }
30     avgOfValues = sumOfValues /
(int)getNumberOfMeasures();
31     averageStack.add(avgOfValues);
32
33     if(measuredTime % 5 == 0 && recalibrateCheck == 1) {
34         if(averageStack.size() > 200) {
35             recalibrateCalculation(averageStack);
36             averageStack.clear();
37         }
38     }
39
40     if(avgOfValues >= pushThres) {
41         frequencyReal++;    //Count up the frequency
42         valueStack.clear();
43     }
44     ...
```

In Listing 9 above the major part of the algorithm is shown, without the time counting that happens past line 44. As already explained in the previous application example, the time count is just used in order to get an approximate measurement about the running time of the application and the algorithm and to update the information on the website once every few seconds. Also the second if clause, where the time difference is calculated, is nothing new. This is an approach to overcome the vague sampling rate of the sensor and the time stamps.

As this application uses the linear accelerometer sensor from android, theoretically no filtering is required, as the values should be near zero and the gravity is factored out. On line 17 the speed is calculated, which is the absolute distance between the actual acceleration values and the previous acceleration values on all three axes. Then those values are added to an ArrayList called `valueStack`, which can be adjusted by the time threshold (the time in milliseconds to perform a shake). So if this time threshold is set to 500ms, the array has the size of five, as the value is divided by 100. The division results from the fact, that every 100ms a value is an appropriate way to get enough information.

At the beginning the array contains only one value and slowly builds up to the maximum size, but anyway the average is calculated in line 30. In the beginning the not totally filled array can be neglected, as it rapidly fills up to the maximum amount and always clears unnecessary or old values. Only the latest n values are viewed, where n again depends on the time threshold. After the average is calculated, the threshold may be adjusted, if the requirements on line 33 are met. The first requirement is a way to perform the recalibration only every 5s, as the user could unwittingly change his pushing behaviour and not stay the same over time. This is simply a way to ensure, that the recalibration is eligible along the other requirement, that checks, if it is anyway wanted. Once the recalibration is performed, the recalibration stack is cleared in order to collect another 200 values. The condition of minimum 200 values is not really necessary, as the sensor collects far more values over five seconds but implemented, in order to ensure, that at least 200 values are used, if somehow the sensor collects less.

After a possible recalibration the average over the `valueStack` is compared to the `pushThres`, which indicates the minimum pushing power the user has to have. If the power or push was hard enough, a variable is counted up, that stands for the frequency. The rest of the code is equal to the previously described application, as again here the frequency is extrapolated to 60s and it is checked, if the user is reanimating in the optimal frequency of 100 pushes per minute.

Values:

This part also only slightly differentiates from the transmission of the previous application. For the time, the code implements again a simple if-statement, which counts up to one second total. The counting statement is again adjusted to the sampling rate of the sensor, though still only an approximation, as the sensor is not performing as often as wanted. In Listing 10 below the values for the transmission are shown, which are passed to the server as two JSON objects. The separation in two JSON objects is done, as the second one, with the Tag "AdditionalInformation" only needs to be sent once the seconds are counted up and not every time a new value arrives. The first one is sent every time a value arrives in order to visualize the movement towards and backwards the reanimating person.

Listing 10: Transmission of the acquired data.

```
01 ... Performing time count
02
03 try {
04   socket.emit("SensorData", new JSONObject().put("time",
time).put("z", z));
05   socket.emit("AdditionalInformation", new
JSONObject().put("freqR", frequencyReal).put("timeMeasure",
measuredTime));
06 } catch (JSONException e) {
07   e.printStackTrace();
08 }
09
10 ... Closing of function body
```

Problem description:

Generally, this algorithm was more an idea or other way to calculate the frequency based on the acceleration and not the distance, as the distance calculation is nearly impossible with the given tools (just a low cost smartphone accelerometer sensor). The real problem with this algorithm approach is, that the acceleration is changing over the reanimation time rapidly, as the user performs the reanimation either in a harder, faster, slower or weaker way and the difference or distinction between those is very difficult. Although the recalibration is a good idea, the function could be much more refined, as the values are often changing much faster than the recalibration happens.

Future ideas:

For the future only the gained knowledge about the distance between two acceleration values will be necessary and useful as well as the recalibration. The attempt is though not really expedient and was not further followed, as the results often varied much more and were not as adequate as wanted. The main result was, that the best idea for further applications is, to concentrate more on the right frequency, based on the acceleration readings.

4.2.3 Final App: Peak detection & frequency

This application is one of the final results of this thesis as well as the corresponding developed algorithm. The application itself is used during the studies with laypersons and trained professionals and visualizes the frequency of the reanimation on the corresponding LifeStream website. After 15 seconds (an adequate update time) the frequency on the website is updated based on the average reanimation frequency during this time. The frequency is calculated for these 15 seconds or any other interval on the smartphone and later transmitted to the server along with other values. The visualization of this algorithm is also explained here, as this algorithm is used during user studies.

First and foremost, the application idea and algorithm is explained, which stands behind the application. Starting with the required minimum operating system. This application requires at least Android 4.4 in order to make use of the android linear accelerometer, which is used in order to work with already optimized values (regarding the noise and drift). Furthermore, the application requires at least Android 6.0 to use the Toolbar. If not, the menu will not appear as it should, or behaves differently, which is not that harmful, as the user anyway normally has no time during a time critical situation, to tweak the algorithm with the menu.

For the calculation the linear acceleration sensor is used, which was already processed out the force of gravity from the raw values. This algorithm is fairly simple and not fully finished, concerning the distance detection, but fully functional in order to find the right frequency, which is sometimes even more important. The optimal frequency of 100 pushes per minute should theoretically be achieved by pushing always at least five centimetres into the chest of the victim. Often enough though, the reanimating person is not hitting those five centimetres and pushes either too hard or too weak, but still in the right frequency, which is also already very effective. As the distance detection with the given sensor and technology is not really possible the approach with frequency seemed more promising as well as an approximation of the distance based on the z-axis acceleration. As the performed reanimation of the user normally changes over time, especially when the power ceases, the frequency detection is very difficult. The requirement to the algorithm must be to detect hard pushes as well as faint pushes. Therefore, some kind of peak detection is implemented. According to previous studies and a lot of acceleration data logging and plotting an idea arose. Once the acceleration values or signal traverse the zero line, a change in acceleration happens and a peak can be detected.

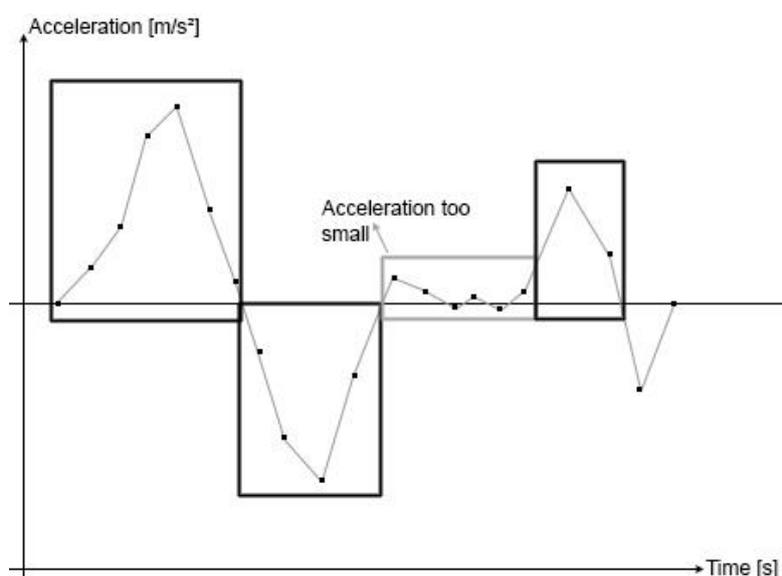


Figure 31: Peak detection and minimum threshold (along z-axis).

So no matter how weak or hard the user pushes, the peak can be detected by its zero-line crossing and change of acceleration, of course with a minimal threshold of applied acceleration. As Figure 31 on the previous pages shows, each peak can be detected independently, of the applied force. The grey box indicates an interval, where the acceleration is too small and the threshold is not hit by the curve. These small peaks are not concerned during the algorithm. Again the most important parameters and properties are described.

Fixed parameters:

IP = "91.219.68.209"	Again this address represents the address of the server at the university of St. Pölten. The web address is http://lifestream.fhstp.ac.at
Port = "80"	With this variable the port of the server is described. Of course, this could be changed into any other port that is available.
TYPE_LINEAR_ACCELEROMETER	The fused sensor is used again, as he processes the gravity out and filters the raw acceleration signal very well.
SENSOR_DELAY_GAME	This time a slightly slower sampling rate is chosen, as the peaks should be detected as well as possible. The delay is defined for 20.000 microseconds, but again this is only a suggestion to the android operating system. The time counting if-statement during the code is used to time the value processing somehow.

Special properties:

pushThres = 1.0	This value indicates the minimum force that has to be applied to the phone in order to detect the peak of the acceleration signal. The value starts at a pre-defined number (according to primary research) and can be adjusted by the application menu.
timeThres = 100	During this algorithm or application this variable is used to check, whether 100ms passed (as 100 is defined) or not, before further calculations are performed.
magnitudeThres = 5	With this threshold the other functionality of the algorithm is checked, because it was necessary to also include the general movement. So every time the user pushes harder than this value, a parameter is counted up. This is used to detect, if the user is moving the phone in any way.
negativeThres = -0.5	This special variable is used to detect or mark the point when the acceleration changes into negative direction. Upon reaching a value of -0.5 or smaller than this the direction change of the acceleration signal can be detected.

As already mentioned, this application also implements the menu in order to fine tune the algorithm. The menu is slightly changed in relation to the other menus, but generally equal.

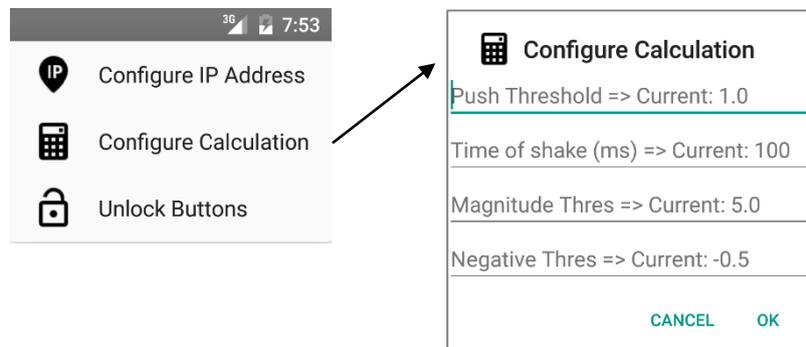


Figure 32: Acceleration menu and submenu with options.

Figure 32 shows the menu for the application, which has basically stayed the same. Only the submenu has changed and gives new customization options. The main functionality of unlocking the buttons and eventually configure the IP address still stays the same.

Code:

The code parts below are taken out of the main application and show the peak detection as well as the rest of the code and the time calculation in more detail as it is used for the server or website. The most important part this time is the self-developed peak detection, which is taken out and described separately and in more detail prior to the rest of the algorithm.

Listing 11: Peak detection algorithm.

```

17 //Perform peak detection
18 if(measuredTime % 15 == 0) {
19     minus = true;
20     for (int i = 1; i < calcValueStack.size() - 1; i++) {
21         if (calcValueStack.get(i) < negativeThres) {
22             minus = true;
23         }
24         if (calcValueStack.get(i - 1) <
calcValueStack.get(i)) {
25             if (calcValueStack.get(i) > calcValueStack.get(i
+ 1)) {
26                 if (calcValueStack.get(i) > pushThres &&
minus == true) {
27                     frequencyReal++;
28                     minus = false;
29                 }
30             }
31         }
32     }
33 }
34 calcValueStack.clear();
35 }

```

Listing 11 above shows from line 17 to 35 the peak detection, that is taken out of the whole code in order to explain it in a more detailed way and previous to the rest. Starting with line 18, first the time is checked, if already 15 seconds have passed.

The amount of 15 is chosen, as it seems appropriate for the website to show the updated frequency. So only if this amount of time passed, the collected values are taken and the peaks are counted within this interval. A Boolean variable `minus` is used in order to determine whether the acceleration crossed the zero line and goes into the negative direction or not. Only the positive peaks are interesting, as those are the result of pushing the smartphone down, if it is lying flat on the chest of a victim. During the for-loop, where all values are passed, each is checked on line 21, if it is smaller than the negative threshold. Once value is found that is smaller than the negative threshold, the Boolean variable `minus` is set to `true` again, which indicates a change or acceleration in the negative direction.

The next three if-statements are capsuled and the first one starting at line 24 checks if the previous value is smaller than the current one. If that is true, the second if-statement inside the first one is evaluated, which checks, if the current value is greater than the next one. The last if-statement inside the second one checks, if the value is greater than the push threshold and then increments a variable, that counts the frequency or peak, because once inside the third if-statement, a peak is found. After the for-loop finishes the stack of values is cleared and the values are dumped. So after the first 15 seconds the algorithm counts how many peaks have been produced, independently on how deep or hard they have been.

The next code parts show the overall implementation and usage of the above described peak detection during or inside the `onSensorChanged` method.

Listing 12: Sensor method with peak detection and time measurement.

```

01 public void onSensorChanged(SensorEvent event) {
02     if (started) {
03         time = System.currentTimeMillis(); //Current time
04
05         if((time - timeOld) > timeThres) { //Control time
06             timeOld = time; //Old time is new one
07             timeCount++; //Counting to seconds
08
09             float[] values = event.values; //Highpass filter
10             float[] filteredValues =
DataManipulation.highPass(values[0], values[1], values[2], 0.8);
11
12             x = filteredValues[0];
13             y = filteredValues[1];
14             z = filteredValues[2];
15             calcValueStack.add(z); //Add z value only
16
17 - 35     //PEAK DETECTION
18
19             magnitude = (float) Math.sqrt((x*x) + (y*y) + (z*z));
20             double accelDiff = Math.abs(magnitude -
magnitudeOld); //Acceleration change
21             magnitudeOld = magnitude; //Old is new one
22
23             if(!sensorInitialized) { //If sensor not used
24                 magnitudeOld = magnitude; //Last magnitude is
old one
25                 sensorInitialized = true;
26
27             }
28         }
29     }
30 }
31
32 ...

```

```
44         } else { //If sensor is used
45         if(accelDiff > magnitudeThres) //If diff greater
46         {
47             frequency += 1; //Count every movement
48         }
49
50         //Count time dependent on smartphone
51         if (timeCount == 50) { //Count 1s
52             measuredTime += 1;
53             timeCount = 0;
54         }
55         if(measuredTime == 16) { //Reset after 15 seconds
56             measuredTime = 1;
57             frequencyReal = 0;
58             frequency = 0;
59         }
60
61         ... Transmission code
```

Listing 12 above and on the previous page is not much different than the previously described methods, but combines a lot of features. First after the initial check if the application is started, the values are copied into an array and filtered with the high pass filter on line ten. After they have been filtered and saved into the respective variables *x*, *y* and *z*, the *z*-value is used for the peak detection. Therefore, the value is saved in an Arraylist called *calcValueStack*, which dynamically grows over time and includes the acceleration values along the *z*-axis. The *z*-axis is used, as it is important to get information about the negative and positive acceleration. By using the magnitude, the negative values would be removed by the Pythagorean theorem. After the peak detection is performed, the magnitude is calculated and used to determine the absolute distance between the current magnitude and the previous one. Although this information is not really used on the website, it is still important in order to count any movement and see the difference between the real peak detection along the *z*-axis and the overall movement of the phone.

From line 51 to 59 the time is calculated again by using the sampling rate of the phone and some counting variables. During this application and the studies, a Nexus 5x was used with the newest android operating system. Prior researches with the phone had shown, that the sampling frequency is around 20ms. As a result of these 20ms sampling frequency the first if-statement on line 51 checks whether the *timeCount* is 50 or not, because 50 times 20 results in 1000ms or one second. If one second passed, another variable is increased, the *measuredTime*, that counts the seconds and the time counter is rest to zero. The second if-statement on line 55 checks, if the variable *measuredTime* has reached a value of 16 (not 15, as the variable starts to count from one). The variable needs to start from one, as the server displays the time and uses it for further calculations. If the variable started from zero, the server and the website would produce an error, as there some calculations with the time are performed. Also the *frequency* is reset to zero, which is used during the magnitude difference check and more important, the *frequencyReal* is reset to zero. During the next 15s the frequency, that has been calculated with the peak detection, is shown on the website and updates only after the next calculation has been performed and sent to the server.

Values:

For the transmission the same structure is used as in the described algorithms previously, only the transmitted values have changed. The transmission includes a simple calculation this time in order to extrapolate the frequency to 60s. Therefore, the counted frequency is simply multiplied by four in order to count up to one minute. The rest of the extrapolation is performed on the website.

Listing 13: Data transmission with multiplied frequency.

```
01 ... Performing time count
02 try {
03     socket.emit("SensorData", new JSONObject().put("time",
time).put("magnitude", z));
04     socket.emit("AdditionalInformation", new
JSONObject().put("freq", frequency).put("freqR",
frequencyReal*4).put("timeMeasure", measuredTime));
05 } catch (JSONException e) {
06     e.printStackTrace();
07 }
08 ... Closing of function body
```

Problem description:

Once again this algorithm is of course not fully perfect, but at least stable enough along the rest of the application to work for ten minutes continuously or more. That is already an amazing outcome and the user studies, that have been performed with a layperson, have shown, that the algorithm works even compared to a professional reanimation phantom that is used to calculate the chest compression and frequency (see [11], [12]). The major problem is still the noisy signal and the sensor as well as the smartphone. The time count is only estimated and not totally equal to the real time and the sampling frequency often behaves differently from phone to phone. Anyway the peak detection works, as intended, even with a changing reanimation behaviour during the ten minutes. One problem though still is the repositioning of the phone, as some of the reanimating persons have to adjust it, if the smartphone slides away during the reanimation. Though, only the z-axis is taken for the calculation and the horizontal or vertical movement should not harm the peak detection, the repositioning still produces errors. These errors can result in strange acceleration signals and therefore in a wrong frequency.

Future ideas:

Although this algorithm is finished or nearly finished for the frequency detection the distance detection is still problematic. The final insight on the ideas or future work is given in Chapter 5 of this thesis, but the next step would be to find the distance. As the distance calculation only works for a short period of time and the error drifts away exponentially, the idea is to perform the distance calculation (by double integrating the acceleration signal) only every 15 seconds for the amount of values that are currently used during the peak detection.

Somehow a fusion of the first described application during this thesis and this one by using the filters and the peak detection and only calculate the distance during the next 15 seconds or any other time that is adjusted in the code.

The approach was not pursued further, as the project already finished by performing the user tests and by showing the current results to the project partners. A future collaboration could of course include the implementation of the described idea and the appropriate testing.

Additional information:

This time the visualization on the website is presented as well, as the results of the algorithm are very promising and used during a test with laypersons. The corresponding LifeStream website shows the reanimation curve, which is the push along the z-axis (later replaced by the distance) to make a point about the depth somehow. The frequency is shown to the right in the information box and has three colour codes.

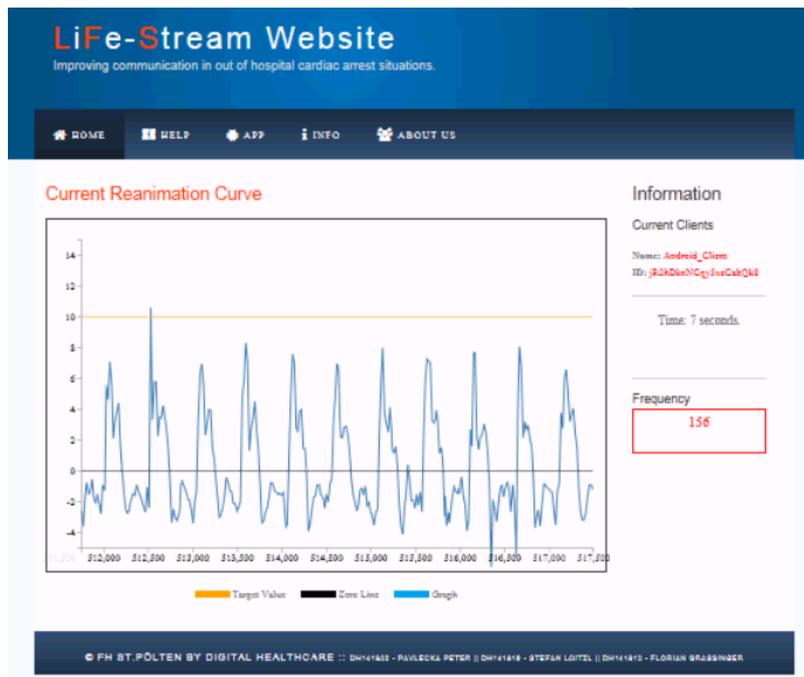


Figure 33: LifeStream website with too high frequency over the last 15 seconds.

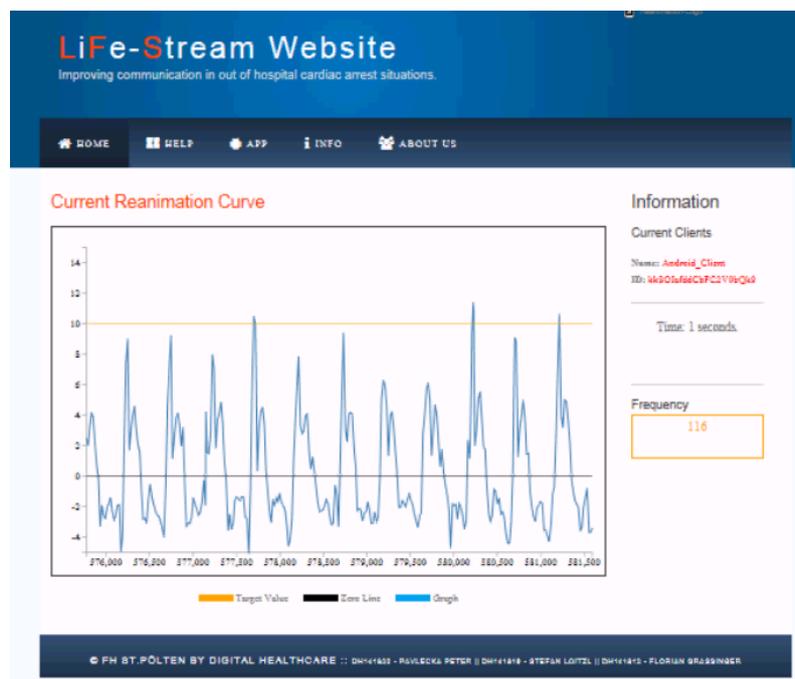


Figure 34: LifeStream website with not so good frequency over last 15 seconds.

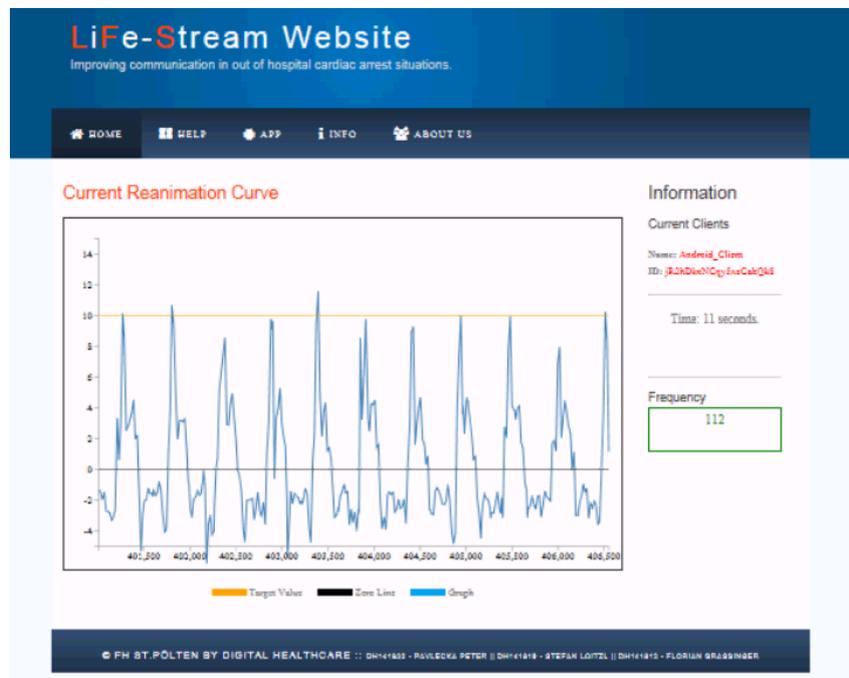


Figure 35: LifeStream website with nearly perfect frequency.

The colour codes for the frequency are the following (based on the range of the frequency):

- Red = Indicates a very bad frequency (all under 90 or above 130)
- Yellow = Indicates an average frequency (from 90 to 100 & 120 to 130 pushes)
- Green = Indicates an optimal frequency (from 100 to 120 pushes)

In the graph the z-axis is visualized, as the movement with the smartphone happens along this axis. By visualizing it, it is possible to give at least a hint about the reached depth or pressure that is applied to the smartphone.

4.2.4 Further Possible Methods

Additionally, to the previous described algorithms a few other possible solutions are available, but not all implemented, either due to lack of time or chance of success. For the sake of completeness, they shall be mentioned here or described shortly:

- **Re-zeroing**

The idea behind this method is fairly simple but effective, if the sensor raw values contain any kind of offset, that is affecting the application. Before performing further calculations with the sensor values, the recalibration factor or array is calculated. This is simply done by measuring N values, store them in an array and find the mean values. So in practice for example 50 acceleration values of x, y, and z are stored in an array and the mean x, y and z-values are calculated. Then, after new values have been read, the mean values are subtracted each time from the new values. By using this method, an offset can be removed from the sensor values and each value is technically re-zeroed. This method was implemented in one of the applications, but did not remove the drift or errors efficiently enough.

- **Kalman filtering**

By using this filter excellent signal processing results can be achieved, but it is very complicated to implement and not suitable for a simple and fast smartphone application. The algorithm takes noisy measurements, some predictions on how the measurement's true value is behaving and some forces, that are causing the system to change. The Kalman filter estimates a process by using feedback control systems. It estimates the process state at some point of time and then obtains feedback in the form of noisy measurements. For the algorithm to function appropriately two kinds of equations are necessary. The time update measurements, which can also be described as the predictor equations and the measurement update equations or corrector equations. In the actual implementation of the filter usually the measurement noise covariance is measured before the filter is applied. Also the process noise must be measured prior to the filtering.

This filter type is extremely flexible and can be used to smooth high-frequency noise or to isolate a periodic signal. If functionality beyond simple smoothing and simple high-pass filtering is needed, a Kalman filter will give the best results. An actual implementation of the filter though is not really possible with the given software and hardware, as the smartphone needs the processing time for the calculation and the data transmission. Also two kinds of equations are necessary and two inputs, but the smartphone produces roughly only one type of input – the acceleration values.

The filter operates by producing a statistically optimal estimate of the system state based on the taken measurements. In order to achieve this the system needs to know the noise of the input to the filter and also the noise of the system itself, which is technically very sophisticated [45].

- **Verlet integration**

Another possible solution for the positioning could be the Verlet method to integrate the devices acceleration to a position. It is a numerical method, that is used to integrate Newton's equations of motion in order to find the position out of the acceleration. Often it is used to calculate trajectories of particles in video games or movement simulations. A much more detailed information about the method can be found on the following two references: "Computer Experiments on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules" [46] and "A Simple Time-Corrected Verlet Integration Method" [47].

Although the method seems very complex, the functionality and stability is great, compared to other algorithms like the Euler Backward Propagation. The major problem with this algorithm and why it is not used, is explained in the following.

There are two criteria to make the verlet algorithm exact. One is the constant acceleration, which is not present in this particular android application, as the acceleration is always changing (even from plus to minus) and the other one is even more problematic for the android implementation, because it is the constant timestamp. As already mentioned several times, the android operating system is not producing real values or usable values by measuring the sampling frequency. Often the values are changing depending on the current running tasks or other things the operating system has to perform. Of course there are workarounds present for the presented problems, like the time corrected verlet integration, but still not suitable for the current application or problem of this thesis, as acceleration is changing constantly as well as the sampling rate.

5 Evaluation of Chest compression depth algorithm Approaches for the Mobile First Responder Client

In this chapter the previously shown applications of Chapter 4 are combined into a final result and evaluation of the performed work. During the work on the project “LifeStream” numerous applications were developed with different algorithms and implementations of those. Not every algorithm is functional and works, but the final results have shown, that it is still possible to make a point about the frequency somehow. Also the distance detection is theoretically possible, but generally not achievable with the defined resources and technologies. More insight in the problems and future plans is given in Chapter 6, which deals with the limitations and discusses more details.

5.1 Aim of this Investigation

Once again the aim of this thesis was to find a possible solution for detecting the chest compression depth and chest compression rate by using a low cost smartphone sensor – the accelerometer. Upon detecting or calculating the frequency and the depth with the smartphone, the information is transmitted to the server (this happens nearly continuously) and from there redirected to the visualization website.

The information than can be used by the professional or dispatcher, who views the website to guide the reanimating layperson through the reanimation. Of course nowadays the reanimations are often guided by the dispatcher (which follows a global guideline), but only with audio feedback and not visually. The reanimating person is led to the right frequency by a simple counting scheme, that is recited by the dispatcher. The dispatcher has the possibility to lead the reanimating person into the right direction but, is not able to control the depth or frequency visually. Often enough people are counting in another frequency than they push, e.g. they count more slowly than they actual move their hands. By using the website, it is possible to give visual feedback about the frequency and maybe in future implementations also about the depth.

5.2 Description of Method of this Examination

The following subitem describes the methods used during the problem examination as well as the methods to test the results produced during the algorithm development. Also the previous made milestone plan is discussed and the general field of research. As a reminder the thesis mainly deals with the algorithm for CCR and CCD detection as well as the transmission of the gathered information and not directly with the overall project and its realization.

5.2.1 Field of Research

The overall project is situated in the Human Computer Interaction (HCI) field of research, or more specifically in the medical application sector and human dispatch life support. The project combines two great fields of research, the technology or software development and the medical sector or dispatch life support. A combination of these two research fields lead to the fantastic project “LifeStream”, which tries to combine already present technology, that is easy accessible to everyone and medical information, in order to enhance the dispatch life support. As the medical field of research is somehow precarious in terms of application development, the application is defined as gadget for dispatch life support and not as medical application.

5.2.2 Course of time

For the thesis, as well as for the project, a timeframe of two years was planned including its development and the contacting of the project members. Mostly the time schedule was followed appropriately and nearly every milestone was achieved. In the following Table 3 the project planning is shown, as well as an annotation to each milestone.

Table 3: Time schedule for the project.

Estimated Date	Milestone	Annotation
9 th November 2015	-) finished research -) basic algorithm concept -) further development of application	At this point the milestone was already achieved and some algorithms had been implemented.
30 th November 2015	-) first implementation of algorithm -) visualization and transmission of information to the website	Here already some other algorithms had been tested and the visualization worked as well. Still in time according to the time schedule.
21 st December 2015	-) validation of the algorithm -) further research -) optimizing data transmission -) writing down results	Also this milestone was still in time as a lot of testing was performed, but the algorithm was not that useful and was dropped later on.
18 th January 2016	-) finalization of data transmission & algorithm -) implementation finished -) writing on thesis	Here the time plan was already complicated, as the writing of the thesis started later and the implementation continued.
22 nd February 2016	-) thesis finalization -) final tests and validation -) visualizing of data	Also this timestamp could not be met, as the algorithms had to be further developed.

Of course the previously made time schedule was not totally achieved, as some unexpected problems occurred during the development or some more insight was gained. For example, the previously developed or found algorithm, that is based on the double integration, did not work as intended, especially with the low cost accelerometer sensor in smartphones. Therefore, also the tests of the algorithms reverted back to March or April, where the first functional algorithm was found for frequency detection and also used during the user tests.

5.3 Final Application Results

In this subsection the final results of the applications, as well as the algorithm development, are pooled. There are numerous applications that have been developed and no less algorithms that have been tested or viewed, but not all lead to a meaningful result. All in all, the detection of the chest compression rate is somehow possible with the given hardware and surroundings, but the chest compression depth detection is far more complicated and nearly impossible with the accelerometer sensor of the smartphone. On the next pages Table 4 lists a short description of the used algorithm, the problems that occurred and why it was not that useful and finally the results. Not all applications are listed in the table and also not all are described in more detail in this thesis (except the good and functional ones).

Table 4: Results of developed applications and methods.

Application: Simple Double Integration	
Algorithm/Method	Simple double integration of the acceleration values. Based on the formulas from 4.1.1, where the general idea is presented. The displacement (change in position with view on the direct way between two points) should be gathered by simply integrating the acceleration values twice. The used sensor was the basic accelerometer.
Problems	Various problems occurred during the calculation, like the drift of the sensor and the general noise that is included in the data. Although even a basic filter has been applied later on, the noise of the raw values still had a non-zero mean. Also the time measurement and sampling frequency is problematic.
Results	<ul style="list-style-type: none"> • Accelerometers in smartphones are noisy and drift. • Sampling rate is hard to configure at fixed time stamps. • Filtering is necessary (noise and drift). • Simple double integration is not possible. • No statement about chest compression rate or depth possible.
Application 2: Double Integration with total acceleration	
Algorithm/Method	Double integration with total acceleration taken into account and further filtering by using the high pass filter primary and later a low pass filter in order to reduce the noise and drift of the raw values. The sensor used is till the basic accelerometer without any sensor fusion. For a more detailed description view 4.2.1.
Problems	As already discovered in the previous application, the sensors work not very precisely and still often contain a lot of noise and drift, that accumulates up during the integration over time. Also the time factor is problematic for the acceleration, as the smartphone produces no continuous time stamps, that have the same distance.
Results	<ul style="list-style-type: none"> • The basic accelerometer sensor is not very useful. • The fused linear accelerometer should be used, that already contains zeroed values in idle state. • Double integration works, but not over a longer amount of time, as the values and time are not stable enough.
Application 3: Magnitude Threshold	
Algorithm/Method	This application uses a completely different approach in order to detect the frequency of the pushes. The magnitude of the acceleration in x, y, z-axis is calculated and checked, whether a threshold is passed or not.
Problems	Although the linear acceleration was used, the idea was not very helpful, as the magnitude does not give a very good statement about the movement.
Results	<ul style="list-style-type: none"> • Movement detection with magnitude is possible, but not good. • The linear accelerometer is better for further development.

Application 4: Metric distance between acceleration	
Algorithm/Method	<p>Based on the previous application three, the idea here was to measure the metric distance between the acceleration signals and make a point about the frequency. The idea also involves the accumulation of the values into a set of values and to perform the calculation later. This should reduce the outliers, as the arithmetic middle of the value set is taken for the calculation.</p> <p>Once the acceleration between two points is calculated and the arithmetic middle of the value set, the result is compared to a predefined threshold. The threshold of a minimum acceleration or speed can be adjusted dynamically based on the reanimation behaviour of the reanimating person. For more information, see the following Chapter 4.2.2.</p>
Problems	<p>Problematic in this approach was the fact, that the values still were very noisy although the linear accelerometer was used. Also the idea of measuring the acceleration distance between two points is not that practical and only useful for frequency detection.</p>
Results	<ul style="list-style-type: none"> • Accelerometers are not very handy especially for distance detection. • The frequency detection works already pretty well, but is not able to detect a change in frequency, once the strength of the push changes.
Application 5: Final Version	
Algorithm/Method	<p>The final application, that is also used during the user tests and studies, implements some kind of peak detection. Each movement that is above a certain threshold is recognized, even if the strength of the pushes changes over time.</p> <p>The peaks are detected with a self-developed algorithm, that detects the change into the positive or negative direction of the acceleration signal. A detailed explanation is given in 4.2.3 Final app: Peak detection & frequency.</p>
Problems	<p>This algorithm also has to deal with the general problems like all the others. The accelerometer in the smartphone and the operating system generally are very unhandy and operate in a wayward manner.</p> <p>Once the sampling frequency is set, the operating system only takes it as suggestion and not directly as fixed value. Also the algorithm still lacks some further checks, as the accidental movement with the phone is also detected, but works for most of the cases.</p>
Results	<ul style="list-style-type: none"> • The application detects the frequency very well and is nearly equal to a professional reanimation phantom that is used for training purpose. • Further investigation is necessary and it is possible to enhance the application by implementing the displacement detection. • Displacement always can be detected over a short amount of time (ideally only over the current value set).

6 Discussion and Conclusion

The aim of this thesis was to show the made researches, discuss the LifeStream platform with the client and server architecture, show the design and development and of course find a possible implementation for chest compression rate and chest compression depth detection by using a low cost smartphone sensor – the accelerometer. Also the optimized transmission and server development, together with the visualization of the gathered information were part of this thesis, though even more important for the whole project. Ideally every layperson should be able to use the application and perform a guided cardiopulmonary resuscitation (CPR) with a better outcome for the patient. By using the developed application, the dispatcher, who instructs the layperson, is able to further guide him or her during the reanimation, as he gets continuous visual feedback about the current situation. Every smartphone nowadays is able to run applications that are based on integrated sensors and the accelerometer sensor is an old sensor, that is already available since the beginning of smartphones. By using this low cost sensor, it should be possible to spread the application around and make it usable for nearly everyone (also due to the simplicity of the app and the slim user interface).

During the work on this thesis and the algorithms for CCR and CCD detection a lot of projects have been viewed and lots of tests were performed with the accelerometer. The tests have shown, that the accelerometer and especially the whole system, where it is integrated, are very different and often rely heavily on the hardware and the algorithm. Some algorithms are much easier to implement, but far not as accurate as others, who are more complex, but need much more data, hardware or information to work. As already mentioned in previous chapters, the accelerometer sensor is often enough not able to measure values without noise and drifts away after some time. The continuous error, which has a non-zero mean, adds up to further calculations that are performed with the raw values.

After further researches visual inspection of acceleration data and various smartphones, a possible solution came up at least for this problem – the linear accelerometer. In contrast to the normal accelerometer, which measures the gravitational force along the z-axis (the axis where the phone is moved towards and away from the user), the linear accelerometer is a fused sensor, that consists of several other sensors in the smartphone. The linear accelerometer has already well optimized filter methods, which are far more accurate than self-implemented ones. For example, in some implementations the linear accelerometer consists of the normal accelerometer and the magnetometer and in other implementations a gyroscope for orientation detection is added. Common to all of them is the fact, that they still contain noise and drift, though the linear accelerometer filters out the gravitational force of the z-axis. Another problem is, that the linear accelerometer is only available in devices with the latest versions of Android version and not all smartphones contain the necessary sensors for the sensor fusion.

Further tests revealed more problems with the acceleration values and the time stamps the sensor is producing. The main algorithm, which double integrates the acceleration signal for example, or some other algorithms, are dependent on time. Though the sampling frequency of the sensor can be adjusted by four fixed parameters or individually, the operating system takes the value for the sampling frequency only as suggestion and performs other tasks in the background. Often enough the sensor samples slightly slower than the actual sampling frequency as other tasks are more important for the operating system in the background (these tasks cannot be stopped, as they are system tasks). So for the algorithm or any calculation it is problematic to rely on fixed time intervals, as they often are slightly shorter or longer. The errors are adding up over time and contribute heavily to the whole calculation.

A possible solution was to wait before calculating and estimating depth. At least 100ms proved to be useful. Still a problem is, that some time stamps are above the 100ms or greater and also the fact, that numerous important values are lost during the defined pause. During peak detection this can be fatal, as a global maximum could be skipped.

Of course, these sensor and device specific problems have been compared to other works, like PocketCPR [20] or an interesting study called “Development of Android Based Chest Compression Feedback Application Using the Accelerometer in Smartphone” [21]. The PocketCPR device allows a direct feedback for the reanimating person, but not the dispatcher. Despite intense researches it was not possible to find further information about the method for depth and frequency sensing. At least more insight for the sensor was gained, as the sensor in this device is more advanced and specifically made for the purpose of reanimation, but no further information about the type and model could be gained.

Also the study was not really helpful, although they used the accelerometer sensor in smartphones, but they applied further filtering methods and used other sensors in order to stabilize the results. Also the application was not tested the full ten minutes continuously, which is important for real life applications. Nevertheless, it was very helpful for the algorithm development, as the filter methods have been reproduced slightly differently and some ideas were gained.

The final results in the previous chapter showed, that there are several possible solution approaches, but not all are very useful or lead into the right direction. During the project and especially in this thesis it turned out, that the frequency detection is much easier than the continuous position determination, especially in smaller unit ranges (like cm's). The accelerometer is not only very noisy and drifts, but also the measurement is often different. The equations, that are theoretical possible or at least physically, are not practical in real world applications and especially in smartphone applications that try to determine the position. Unless the smartphone is accelerating in one direction only, the calculations will fail normally without including a set of gyroscopes and far more accurate sensors. Accelerometers measure the acceleration in a body fixed reference frame, where normally displacement in earth fixed reference frames is necessary.

Therefore, it is not possible to only integrate the accelerometer twice and find the displacement, except it is rotated into the earth fixed frame, before the integration takes place. All this works theoretically, assuming that the current available accelerometer sensors are perfect, however they are far away from being perfect, actually. There are two major errors present, where one is the bias¹¹ on the accelerometer and the other an initially present tilt error. These errors add up during the integration for position determination and sophisticate the results. As double integration is the only way to find the displacement out of the acceleration, the sensor has to be improved or further sensors need to be included in order to correct the bias error [49].

During the thesis it became clear, that with the given premises of only using the low cost accelerometer sensor in smartphones, it is not possible to make a sturdy point about the displacement (at least for the required ten minutes continuously). Nevertheless, it is possible to make a point about the current reanimation frequency very well by using the developed peak detection algorithm. Even a position determination could be possible by using the peak detection and the current viewed values during the peak detection (a so called window of values) for the integration. As the values always are restricted to a certain amount and interval a double integration of those values would contain less errors, that could add up over time.

¹¹ A measurement error that remains constant in his magnitude for all observations. Some kind of systematic error, that is non changing [48].

Nevertheless, the error is still present, due to already mentioned reasons. The implementation of the double integration after the peak detection was not part of this thesis anymore, as the frequency detection works very well and was used during the final studies for the application and the overall project. On the next page the main research question and its sub questions are once again shown and examined to answer with the findings during this thesis work.

Question 1

Can low-cost and mobile sensors like the accelerometer in mobile devices (e.g. smartphones) provide high-density data for real-time CPR signal processing and data transmission?

Answer

According to the researches made during the project and the whole thesis, this question has to be split based on the feasibility. The previous results or algorithms showed, that the frequency detection is easier achievable in contrast to the depth detection with the defined methods or setup. Some algorithms are more stable in sense of frequency detection than others, like the peak detection for example. However, in other cases robustness in depth and frequency detection could not be achieved.

The data acquisition of the smartphone is very high and provides a lot of data for further usage and calculations. Although the arrival of the data is dependent on the sampling rate, which for her part is depending on the operating system. The data comes in irregular intervals which makes a calculation or further usage very difficult and requires a lot of further processing. Also the calculations are limited due to the fact, that the accelerometer in smartphones is not developed for accurate position determination.

Generally, it is possible to claim, that the low cost accelerometer sensor in mobile phones or any multi modal devices is good for simple measurements or measurements, that do not require a too high accuracy. A simple frequency detection, which is basically a movement recognition or peak detection, where the peaks are produced by moving the phone is easily possible and applicable for the LifeStream project.

On the other hand, the numerous tests and prototypic algorithm implementations showed, that a precise depth detection is not possible by just using the devices accelerometer sensor. The measured raw values are too noisy and a precise calculation cannot be performed due to physical and technical limitations. At least not in the required unit range of centimetres, which would be necessary for the CCD detection.

Question 1.1

Which real-time algorithms can be used for frequency detection (chest compression rate = CCR) and distance detection (chest compression depth = CCD)?

Answer

Various algorithms have been tried out during this thesis. Though the best one, that is used during the user test, is the peak detection, which makes a point about the current reanimation frequency. Compared to other algorithms and tests, the peak detection is the most promising for the CCR detection, where CCD detection is not really possible with the defined technology.

Question 1.2

Is it possible to make a testimonial evidence about distance detection with the accelerometer (low cost sensor)?

Answer

Based on the researches and explanations during this thesis it is theoretically possible, but the sensor produces too much noise and drift and the smartphone sampling rate is also inappropriate. Therefore, no testimonial evidence about continuous distance sensing is possible over ten minutes or more. However, from peak to peak distance sensing could be feasible, if further researches and testing is performed. Some of the algorithms for distance calculation are able to perform well for a short amount of time, before the error increases.

Question 1.3

How can the application be optimized, as well as the data transmission, to ensure a fluent transmission and appropriate visualization on the client side (website)?

Answer

While working on the project and thesis continuous updates have been made for the application. For example, the full screen mode is implemented (as far as possible) and the sensor usage is stopped, once the application stops or the transmission is shut down. The data transmission uses the JSON format for small and short packages that are continuously sent. The transmission is performed by using TCP, which ensures that the packages are delivered to the server. For further stability all processes are terminated appropriately and the calculation is optimized as well.

7 Summary

Aim of this thesis was to find a possible algorithm in order to detect important parameters, the chest compression rate and frequency, during a cardiopulmonary resuscitation in a cardiac arrest situation. The reanimation is performed by a layperson who receives guidance over the phone from a dispatcher, until the ambulance arrives. Often enough the layperson reanimates in the wrong frequency and not deep enough, which is not promoting for the overall good outcome of the situation. The right frequency of 100 pushes per minute and the optimal depth of 54 mm is important according to medical guidelines.

The thesis and the research during the thesis evolved around a project called "LifeStream" which is a collaboration between the university of applied sciences (FH St. Pölten), the medical university of Vienna and Notruf NÖ GmbH. The idea was or is to support laypersons and dispatchers by using the smartphone, which is already in use for the call, for further information gain during an ongoing reanimation process. While the person is reanimating, the is able to give further feedback by using the developed smartphone application, as it transmits information about the reanimation process to a corresponding website.

The project started by developing a simple application for acceleration detection and a server to display it. Later a website was developed in order to visualize the parameters of the reanimation and the current reanimation curve (moving curve of the phone). During the work on the project and thesis numerous other projects have been viewed, which deal with similar subjects, but not exactly the same. Based on the researches, it was possible to gain further insight in the smartphone, sensor and server thematic. The server and website are not directly part of this thesis, nevertheless work very well and are able to display the data in nearly real time. The server accepts information from mobile clients on the one side and deploys the information on the other side to connected website clients, that view the information.

The smartphones collect the acceleration data in real time and process it further. The values are for example filtered before further calculations are performed or stored for later usage. Most of the developed algorithms use the linear accelerometer sensor of the smartphone, which already filters out the gravity component, that is affecting the z-axis and smooth the values. All calculations for the algorithms are performed by the smartphone, as the way to the server and back would be too long, if the calculation was based on the previous and next values. Each time a new value arrives, it is stored in an array or container for further calculations, that are based upon more values. If the data were transferred to the server, the time would also slightly shift, even more than it is actually doing due to the fact that the sampling rate of the smartphone is not exact.

During the thesis work and research, it became more and more clear, that admittedly it is theoretically possible to calculate the displacement out of the acceleration, but not practically. The accelerometer sensor of the smartphone is very noisy and drifts over time, the continuous non-zero error mean adds up during the calculation very fast. As the acceleration values need to be integrated twice in order to get the displacement, the error is increasing even faster. The smartphone accelerometer sensor is not made for precise position determination, as its normal purpose is to detect whether the user is performing a shake or any other gesture.

The accelerometers, that are implemented in nowadays smartphones and older ones, are not that precise and even though the linear accelerometer is used, the values are often enough not accurate. The linear accelerometer even often consists of different parts, depending on the manufacturer of the smartphone. As the linear accelerometer is a fused sensor, out of others, the other sensor can be the gyroscope or the magnetometer or any other useful sensor for filtering out the gravity and smoothing the values.

Although it still was possible to gain at least information about the frequency, the phone is moving and the reanimating person is pushing. By implementing some sort of peak detection algorithm, it is possible to measure the frequency independently of the actual compression depth. Tests with real users have shown, that during an ongoing reanimation process for ten minutes, the user is continuously changing his or her reanimation behaviour, as time passes by. At the beginning most people are reanimating much faster, as they still have more power reserves and near the end the reanimation is much weaker, but still the frequency is right. The algorithm detects these changes in the reanimation behaviour and is not fixed to a certain minimum push, that needs to be achieved. Even a weak push in the right frequency can contribute at least a little to the overall outcome [50].

All in all, the project has opened a fantastic possibility for future research and development. For example, the peak detection could be further refined and the double integration performed only during the short period of the current peak detection. Before the next update arrives, the values are removed and therefore also possible errors. The project gave a fantastic insight into sensor programming, server programming and website visualization and is an ambitious project, which definitely should be further refined and investigated.

8 Reference List

- [1] M. Ljunggren, M. Castrén, and K. Bohm, "Abstract 231: The Effect of Dispatcher Call Processing Interval and Ambulance Response Interval for Survival From Cardiac Arrest," *Circulation*, vol. 128, no. Suppl 22, pp. A231–A231, Jan. 2016.
- [2] M. Levy, D. Yost, R. G. Walker, E. Scheunemann, and S. R. Mendive, "A quality improvement initiative to optimize use of a mechanical chest compression device within a high-performance CPR approach to out-of-hospital cardiac arrest resuscitation," *Resuscitation*, vol. 92, pp. 32–37, Jul. 2015.
- [3] "Sudden Cardiac Arrest Treatment | Sudden Cardiac Arrest Foundation." [Online]. Available: <http://www.sca-aware.org/sudden-cardiac-arrest-treatment>. [Accessed: 17-Jul-2016].
- [4] L. Wik, J. Kramer-Johansen, H. Myklebust, H. Sjørebø, L. Svensson, B. Fellows, and P. A. Steen, "Quality of cardiopulmonary resuscitation during out-of-hospital cardiac arrest," *JAMA*, vol. 293, no. 3, pp. 299–304, Jan. 2005.
- [5] P. Eisenburger and P. Safar, "Life supporting first aid training of the public—review and recommendations," *Resuscitation*, vol. 41, no. 1, pp. 3–18, 1999.
- [6] American Heart Association, "About Cardiopulmonary Resuscitation (CPR) (continued)," 2012. [Online]. Available: https://www.heart.org/idc/groups/heart-public/@wcm/@ecc/documents/downloadable/ucm_438917.pdf. [Accessed: 31-Mar-2016].
- [7] B. Roessler, R. Fleischhackl, H. Losert, C. Wandaller, J. Arrich, M. Mittlboeck, H. Domanovits, and K. Hoerauf, "Cardiopulmonary resuscitation and the 2005 universal algorithm: Has the quality of CPR improved?," *Wien. Klin. Wochenschr.*, vol. 121, no. 1–2, pp. 41–46, Jan. 2009.
- [8] C. Ro, "The 'chain of survival' concept: how it can save lives.," *Heart Dis. Stroke J. Prim. Care Physicians*, vol. 1, no. 1, pp. 43–45, 1992 1991.
- [9] J. P. Nolan, G. D. Perkins, and J. Soar, "Chest Compression Rate Where Is the Sweet Spot?," *Circulation*, vol. 125, no. 24, pp. 2968–2970, Jun. 2012.
- [10] G. D. Perkins, A. J. Handley, R. W. Koster, M. Castrén, M. A. Smyth, T. Olasveengen, K. G. Monsieurs, V. Raffay, J.-T. Gräsner, V. Wenzel, G. Ristagno, and J. Soar, "European Resuscitation Council Guidelines for Resuscitation 2015," *Resuscitation*, vol. 95, pp. 81–99, Oct. 2015.
- [11] P. Pavelcka, "LiFeStream - Validation of a mobile application for visual assessment of cardiopulmonary resuscitation (CPR) acceleration data compared with a life-support manikin sensory," Masterthesis, FH St. Pölten, St. Pölten, 2016.
- [12] S. Loitzl, "Validating a sensor-based mobile health service with real-time data acquisition for improving MPDS communication for lay-rescuer in out-of-hospital cardiac arrest based on a randomized simulation trial," Masterthesis, FH St. Pölten, St. Pölten, 2016.
- [13] "What is data streaming? - Definition from WhatIs.com," *SearchNetworking*. [Online]. Available: <http://searchnetworking.techtarget.com/definition/data-streaming>. [Accessed: 04-Apr-2016].
- [14] J. Rybach, *Physik für Bachelors: mit 92 durchgerechneten Beispielen, 176 Testfragen mit Antworten sowie 93 Übungsaufgaben mit kommentierten Musterlösungen*, 2., Aktualisierte Aufl. München: Fachbuchverl. Leipzig im Carl-Hanser-Verl, 2010.
- [15] G. Elert, "The Physics Hypertextbook," *The Physics Hypertextbook*. [Online]. Available: <http://physics.info/>. [Accessed: 06-Apr-2016].

- [16] "Accelerometer & Gyro Tutorial," *Instructables.com*. [Online]. Available: <http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/>. [Accessed: 08-Apr-2016].
- [17] E. W. Weisstein, "Pythagorean Theorem." [Online]. Available: <http://mathworld.wolfram.com/PythagoreanTheorem.html>. [Accessed: 11-Apr-2016].
- [18] C. Woodford, "How accelerometers work | Types of accelerometers," *Explain that Stuff*, 30-Jun-2015. [Online]. Available: <http://www.explainthatstuff.com/accelerometers.html>. [Accessed: 11-Apr-2016].
- [19] "Node.js Tutorial – Step-by-Step Guide For Getting Started." [Online]. Available: <http://www.airpair.com/javascript/node-js-tutorial>. [Accessed: 19-Apr-2016].
- [20] ZOLL Medical Deutschland GmbH, "PocketCPR - ZOLL Medical," 2016. [Online]. Available: <http://www.zoll.com/de/produkte/pocketcpr/>. [Accessed: 11-Apr-2016].
- [21] Y. Song, Y. Chee, J. Oh, and S. Lee, "Development of Android Based Chest Compression Feedback Application Using the Accelerometer in Smartphone," *Proc. Int. Conf. Biomed. Eng. Syst.*, Aug. 2014.
- [22] G. D. Perkins, C. Augré, H. Rogers, M. Allan, and D. R. Thickett, "CPREzy: an evaluation during simulated cardiac arrest on a hospital bed," *Resuscitation*, vol. 64, no. 1, pp. 103–108, Jan. 2005.
- [23] "CPREzy CPR Assistance Device." [Online]. Available: <http://site.jjstech.com/pdf/FirstVoice/CPREZ01.pdf>. [Accessed: 16-Apr-2016].
- [24] "Usage Scenarios: An Agile Introduction." [Online]. Available: <http://agilemodeling.com/artifacts/usageScenario.htm>. [Accessed: 20-Apr-2016].
- [25] F. Flake and D. Buers, Eds., *60 Fälle Rettungsdienst*, 2. Aufl. München: Elsevier, Urban & Fischer, 2011.
- [26] "What is Application Program Interface (API)? Webopedia Definition." [Online]. Available: <http://www.webopedia.com/TERM/A/API.html>. [Accessed: 22-Apr-2016].
- [27] "Package Index | Android Developers." [Online]. Available: <http://developer.android.com/reference/packages.html>. [Accessed: 25-Apr-2016].
- [28] "Android Services - Tutorial." [Online]. Available: <http://www.vogella.com/tutorials/AndroidServices/article.html>. [Accessed: 25-Apr-2016].
- [29] V. Grec, "gesture builder – Android Research Blog," 10-Jan-2012. .
- [30] Y. Golecha, "How Do Annotations Work in Java? - DZone Java," *dzone.com*, 22-Feb-2014. [Online]. Available: <https://dzone.com/articles/how-annotations-work-java>. [Accessed: 27-Apr-2016].
- [31] "MongoDB Explained in 5 Minutes or Less," *www.credera.com*, 30-Jan-2014. .
- [32] "jQuery Fundamentals :: A guide to the basics of jQuery." [Online]. Available: <http://jqfundamentals.com/>. [Accessed: 28-Apr-2016].
- [33] M. Bostock, "D3.js - Data-Driven Documents." [Online]. Available: <https://d3js.org/>. [Accessed: 28-Apr-2016].
- [34] "JSON." [Online]. Available: <http://www.json.org/>. [Accessed: 29-Apr-2016].

- [35] G. Milette and A. Stroud, *Professional Android Sensor Programming*. John Wiley & Sons, 2012.
- [36] P. Becker, "Erfassung und Verarbeitung von Sensordaten," Hochschule Bonn-Rhein-Sieg, 2013.
- [37] K. Seifert and O. Camacho, "Implementing positioning algorithms using accelerometers," *Free. Semicond.*, 2007.
- [38] D. Acharjee, A. Mukherjee, J. K. Mandal, and N. Mukherjee, "Activity recognition system using inbuilt sensors of smart mobile phone and minimizing feature vectors," *Microsyst. Technol.*, May 2015.
- [39] "accelerometer - Android TYPE_LINEAR_ACCELERATION sensor - what does it show? - Stack Overflow." [Online]. Available: <http://stackoverflow.com/questions/7858759/android-type-linear-acceleration-sensor-what-does-it-show>. [Accessed: 07-May-2016].
- [40] L. Tan, *Fundamentals of Analog and Digital Signal Processing*. AuthorHouse, 2008.
- [41] "newtonian mechanics - Calculation of Distance from measured Acceleration vs Time - Physics Stack Exchange." [Online]. Available: <http://physics.stackexchange.com/questions/36312/calculation-of-distance-from-measured-acceleration-vs-time>. [Accessed: 02-Aug-2016].
- [42] "Android Accelerometer." [Online]. Available: <http://www.kircherelectronics.com/blog/index.php/11-android/sensors/7-android-accelerator>. [Accessed: 13-May-2016].
- [43] K. Sloth, "TKJ Electronics » A practical approach to Kalman filter and how to implement it," 10-Sep-2012. .
- [44] "accelerometer - Calculating distance using Linear acceleration android - Stack Overflow." [Online]. Available: <http://stackoverflow.com/questions/12926459/calculating-distance-using-linear-acceleration-android>. [Accessed: 23-May-2016].
- [45] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," 24-Jul-2006. [Online]. Available: http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf. [Accessed: 01-Jun-2016].
- [46] L. Verlet, "Computer 'Experiments' on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules," *Phys. Rev.*, vol. 159, no. 1, pp. 98–103, Jul. 1967.
- [47] J. Dummer, "A Simple Time-Corrected Verlet Integration Method." [Online]. Available: <http://lonesock.net/article/verlet.html>. [Accessed: 10-Mar-2016].
- [48] "bias error," *TheFreeDictionary.com*. [Online]. Available: <http://encyclopedia2.thefreedictionary.com/bias+error>. [Accessed: 16-Jul-2016].
- [49] "newtonian mechanics - Calculation of Distance from measured Acceleration vs Time - Physics Stack Exchange." [Online]. Available: <http://physics.stackexchange.com/questions/36312/calculation-of-distance-from-measured-acceleration-vs-time>. [Accessed: 12-Jun-2016].
- [50] R. Greif, "'Push hard and fast' – Cardiopulmonary resuscitation from 2010 on," *Trends Anaesth. Crit. Care*, vol. 1, no. 4, p. 175, Aug. 2011.

9 List of Tables

Table 1: Challenges and solutions for server.....	20
Table 2: Related work and state of the art.	22
Table 3: Time schedule for the project.....	71
Table 4: Results of developed applications and methods.....	72

10 List of Figures

Figure 1: Structure of the thesis.....	10
Figure 2: Simple visualization of a travelled distance. Note: not the direction is important, only the way.	13
Figure 3: Simple visualization of displacement. This time the direction is important, as it is a vector quantity.	14
Figure 4: The relation or difference between distance and displacement.	14
Figure 5: Velocity and speed relation.....	15
Figure 6: Accelerometer explained, with a ball inside a box [16].	17
Figure 7: Moving the ball inside the box [16].	18
Figure 8: Ball in resting position with gravity [16].	18
Figure 9: Coordinate system and force vector [16].	19
Figure 10: PocketCPR by Zoll [17].....	22
Figure 11: PocketCPR for Droid by Zoll [17].	23
Figure 12: Monitoring screen App [21].	23
Figure 13: CPREzy device and included stuff [23].	23
Figure 14: Reanimation process visualization.	25
Figure 15: Reanimation with phone and LifeStream-App.	26
Figure 16: App Mock-up.	28
Figure 17: First implementation of the draft.	30
Figure 18: Circle gesture with "Gesture Builder".	31
Figure 19: Gesture design implementation of the app. Design made by Melanie Kain & Nina Hladil.....	32
Figure 20: Final used application design.	33
Figure 21: Final application result screen and saving.	34
Figure 22: Subarea of the application - menu.	34
Figure 23: Fullscreen and back button overridden.....	35
Figure 24: Data transmission schematic.	36
Figure 25: Website design and home page view.	37
Figure 26: Partial visualization example on the website.	40
Figure 27: Sensor data from moved x-axis.	43
Figure 28: Flow of the algorithm description in the following part of this thesis.	49
Figure 29: Maxima of acceleration function for one axis.	55
Figure 30: Application menu and algorithm calibration submenu.	57
Figure 31: Peak detection and minimum threshold (along z-axis).	61
Figure 32: Acceleration menu and submenu with options.	63
Figure 33: LifeStream website with too high frequency over the last 15 seconds.	67
Figure 34: LifeStream website with not so good frequency over last 15 seconds.....	67
Figure 35: LifeStream website with nearly perfect frequency.	68

11 List of Formulas

Eq. (1) Distance Formula with continuous acceleration.....	13
Eq. (2) Distance Formula without continuous acceleration.....	13
Eq. (3) Displacement in three dimensional space.....	14
Eq. (4) Speed calculation.....	15
Eq. (5) Speed calculation alternate form.....	15
Eq. (6) Velocity calculation.	15
Eq. (7) Acceleration calculation formula (constant approach).....	16
Eq. (8) Formula for calculating the average acceleration.....	16
Eq. (9) Pythagorean theorem in 3D.	19
Eq. (10) Newton's second law of motion.....	19
Eq. (11) Integration to get velocity out of acceleration.	42
Eq. (12) Second integration in order to get distance out of velocity.	42
Eq. (13) Frequency to period relation.	44
Eq. (14) Calculation of the magnitude.....	45
Eq. (15) Modification of base formula for distance calculation out of acceleration.....	53
Eq. (16) Formula to count measured frequency within some seconds up to 60 seconds.	53

12 List of Listings

Listing 1: Default data package.	39
Listing 2: Pseudo code of a simple calculation for distance.	43
Listing 3: Reading sensor values showcase.	43
Listing 4: Low pass filter implementation.	51
Listing 5: High pass filter implementation.	51
Listing 6: Main calculation method upon acquiring new sensor values.....	52
Listing 7: Time dependent transmission of the sensor values.	54
Listing 8: Recalibration method for the pushes.....	57
Listing 9: Major part of the algorithm is described.....	58
Listing 10: Transmission of the acquired data.	60
Listing 11: Peak detection algorithm.....	63
Listing 12: Sensor method with peak detection and time measurement.	64
Listing 13: Data transmission with multiplied frequency.....	66

13 List of Abbreviations

A

API *Application Programming Interface*

C

CCD *Chest Compression Depth*

CCR *Chest Compression Rate*

CPR *Cardiopulmonary resuscitation*

CSS *Cascading Style Sheets*

csv *Comma-Separated Values*

D

DOM *Document Object Model*

E

EMD *Emergency Medical Dispatcher*

EMS *Emergency Medical Service*

ERC *European Resuscitation Council*

G

GPS *Global Positioning System*

H

HCI *Human Computer Interaction*

HTML *Hypertext Markup Language*

HTTP *Hypertext Transfer Protocol*

I

IDE *Integrated Development Environment*

IP *Internet Protocol*

J

JSON *JavaScript Object Notation*

L

LTE *Long Term Evolution, Long Term Evolution*

S

SMA *Simple Moving Average*

SQL *Structured Query Language*

SVG *Scalable Vector Graphics*

T

TCP *Transmission Control Protocol*

U

UDP *User Datagram Protocol*

14 Appendix

Included with this thesis is a CD with the following components:

- All described applications and the final application.
- Server and website folder structure for an easy deployment to every NodeJs runtime environment.
- This thesis in .pdf format.