MALMÖ UNIVERSITY

# Open-source algorithm for wearables in healthcare-applications

Open-source algoritm för aktivitetsmätare inom hälso- och sjukvård

Anna Klingberg Brondin

Marcus Nordström

**Degree:** Bachelor 180 ECTS
**Major:** Computer Science
**Program:** Computer Engineering
**Date:** June 2020

**Supervisors:** Dario Salvi
& Carl Magnus Olsson
**Examiner:** Reza Malekian

## Abstract

In today's society, it is quite common to track your own health with the use of a wearable device. These devices track physical activity and physiological signals. This is a concept that could be used in healthcare-applications as well. The main issue with this lies in the fact that commercially available devices send personal data to their own servers. The goal of this thesis is therefore to set in motion a project to build an entirely open-source firmware for smart watches for use in healthcare, where reliability, accuracy and privacy are driving quality attributes. This thesis covers a step counting algorithm in addition to the firmware for the watch. To speed up the process of developing the algorithm, an existing algorithm for smartphones is used as a starting point. This algorithm is rewritten, optimized for wearable devices and tested with an existing dataset. The firmware is built with an existing RTOS implementation and the algorithm is integrated into it. To test the firmware, 10 participants conducted several test scenarios, both on a treadmill and on mixed terrain. The results of this were a median accuracy of 92% and could be improved further with more optimizations with a larger dataset. The source code is publicly available on GitHub [1].

Keywords: *Step counting, wearable, open-source, privacy, healthcare*

## Sammanfattning

I dagens samhälle har aktivitetsmätare blivit allt vanligare för att logga vår fysiska hälsa. Detta är något som även skulle kunna gynna och användas i sjukvårdssammanhang. Problemet är att dagens kommersiella aktivitetsmätare sparar personlig information och privat data på sina egna servar. Syftet med denna avhandling är därför att påbörja ett projekt där all mjukvara är open-source och där tillförlitlighet, noggrannhet och integritet är drivande attribut. En stegräknaralgoritm implementeras för smartklockor som ska kunna användas inom vården. Utvecklingen av algoritmen bygger på en existerande stegräknaralgoritm för smartphones som har skrivits om och optimerats för inbyggda system, så som aktivitetsmätare. Mjukvaran är konsturerad med ett realtidsoperativsystem där algoritmen är integrerad. För att testa algoritmen har 10 deltagare genomfört ett antal tester, både på löpband men också utomhus på varierat underlag. Den slutliga noggrannheten resulterade i en median på 92% och skulle kunna förbättras genom att vidareutveckla optimeringen med hjälp av större datamängder. Källkoden är tillgänglig för allmänheten på GitHub [1].

## Acknowledgement

We would like to thank our supervisors Dario Salvi and Carl Magnus Olsson for their generous support and feedback. This thesis would not have been possible without them. We would also like to thank our teacher Magnus Krampell for always answering questions and coming up with new ideas to move the project forward. Finally, thanks to our participants who generously helped us with testing.

## Glossary

**Black box** An opaque system where only inputs and outputs are visible
**MEMS** Microelectromechanical systems
**RTOS** Real time operating system
**IoT** Internet of things
**FPU** Floating point unit, a processors unit for floating point arithmetic
**CORDIC** Coordinate rotation digital computer
**GUI** Graphical user interface

# Contents

## List of Figures

## List of Tables

# 1 Introduction

## 1.1 Background

In today's modern society, it is quite common to track your own health with the use of a wearable device [2]. These devices are able to quantify physical activity using quantities like steps, calories and walked distance. Some devices are even able to track physiological signals like heart rate, heart rate variability and blood oxygen saturation [3].

The use of wearable devices has increased as an attractive complement to clinical diagnosis and follow up. Tracking physical activity and physiology can help capture the progression of symptoms over time or evaluate the therapy on an individual basis [4]. Studies have shown that information from interactive wearable monitoring is valuable for both patients and health professionals and can improve health care [4]. Notwithstanding their potential, two issues hinder their use in healthcare; accuracy and transparency.

Although studies about accuracy of common devices are being conducted [5], most of the commercial products are sold as a *black box* of which no details are published about the algorithms they employ nor the accuracy they achieve. If the source code of these algorithms were made public, e.g. through an open source license, researchers could test the algorithms accuracy under different circumstances.

Considering patients' personal data, the need of secure and privacy-respecting solutions is important. Most of the commercial devices connect to a companion mobile phone application that obliges users to accept a poor user friendly privacy policy [6]. This makes these devices not suitable for use in healthcare applications. This problem could also be partially solved with an open source approach. There is, in fact, conviction within the computer security community that published designs, protocols and even source code contributes to improved security [7]. Open-source removes the dependence on a single party to decide in favor or against a certain system by enabling users to evaluate the security by themselves [7]. It also enables users to patch software if there are bugs and submit changes to a central repository so other users can update their systems to include the patch. An open source software would allow users and researchers to develop custom secure and privacy-respecting firmware on the devices for use in health care.

## 1.2 Purpose

This thesis is developed within an initiative that aims at building an entirely open-source firmware for fitness trackers and smart watches for use in healthcare, where reliability, accuracy and privacy are driving quality attributes. The project will rely on existing programmable commercial devices as much as possible. The major limitation of existing solutions is that they usually lack well-tested algorithms for extracting data from the raw sensor data (e.g. from accelerometers or photoplethysmography). The aim of our thesis is to port an algorithm, particularly for extracting steps from accelerometry, to work in embedded systems. The algorithm is integrated into a wearable device as a starting point for the initiative's firmware.

## 1.3 Research questions

The main research question for this thesis is:

**RQ1:** How can an open-source algorithm for step counting, with a known accuracy for use in healthcare-applications, be implemented?

To answer the main question these questions are formulated:

**RQ1.1:** How can steps in fitness trackers be counted from raw accelerometery data?
**RQ1.2:** How can the algorithm be made portable? (See 2.2 for definition)
**RQ1.3:** How can the step counting accuracy of the algorithm be determined?
**RQ1.4:** How can the algorithm be optimized and fine tuned in order to reduce battery drain?

## 1.4 Limitations

For the purposes of this study, a complete set of possible algorithms for step counting is not presented or tested. Instead the focus has been on one approach which has been shown as particularly suitable in a related study [8], as well as implemented [9] and shown to be computationally efficient in smartphones.

# 2 Theory

This section gives an overview of relevant theoretical aspects of this thesis. With this, the reader will be provided with enough information to follow the discussion further in this paper.

## 2.1 Accelerometry

An accelerometer is a device that measures either static or dynamic acceleration. Static is for example gravity and dynamic can be vibrations or a movement. There are two types of accelerometer sensors; capacitive or piezoelectric. Capacitive uses fixed plates with a plate hung by a spring in the middle. To get the acceleration you read out changes to the capacitance [10]. Piezoelectric uses masses hung by a spring and covered in a piezoelectric material, this material produces a voltage based on the mechanical stress i.e. acceleration [10]. Accelerometers often have three of these inner sensors on three orthogonal axes in space $(x, y, z)$ and sometimes $x$ and $y$ are combined [11]. The accelerometers that are common in smartphones and wearables are so tiny they are considered *MEMS* (Microelectromechanical systems). The size of the entire sensor, with MEMS and the electronics for measurement is often less than $3mm^2$ [11].

Acclerometers comes in a wide variety of sensitivity and accuracy. Sensitivity is measured in G-forces and typical ranges are 2-16G but there exists accelerometers for crashes that can record up to 250G. The accuracy is often associated with the sensitivity as the lower the sensitivity, the higher the precision in the measurement [10].

## 2.2 Software portability

Software portability is defined as the effort of adapting software to new platforms and environments within reasonable limits [12]. C is a programming language that was first standardized in 1989; *C89.* This version is also known as *ANSI C* and many compilers for different environments and CPU-architectures support it. Two of the most common compilers are GCC [13] and Clang [14] and they support ANSI C. Because of this wide support for different environments, C can be compiled to many different devices. As a result of this C is considered a highly portable language.

## 2.3 Open-source

Open source software is a way of writing source code and leaving it available for other people. Because of different licenses, an open source software permit others to study, use, change and share the source code in a modified or unmodified form [15]. There is a spectrum of how strict the different licences are. The strongest, GNU AGPLv3, has many conditions to follow e.g. modifications needs to be released under the same license, a copy of the license has to be included with the software and if the software is distributed (even via a network) the source code needs to be available. In the other end of the spectrum, with no conditions whatsoever, is the Unlicence [16]. Choosing the right licence could be important.

Writing an open source project also includes criteria that needs to be followed. This criteria includes e.g. the program must include source code, modification under the same terms as the licence of the original software, free redistribution and must not discriminate against persons, groups or fields of endeavor [15].

## 2.4 RTOS

RTOS (Real Time Operating System) is an operating systems that work with real-time applications i.e. those that need to process data and react in real time. RTOSes are designed with reliability in mind and are therefore deterministic [17]. RTOSes are split into two categories; hard and soft. Hard RTOSes are completely deterministic and always meet deadline whilst soft RTOSes sometimes do not meet deadlines. Hard real time is much harder to develop and is used in very critical applications. Soft real time is more widespread and often used in $IoT$-applications (Internet of Things) as there exists a need in these devices to do certain tasks at a certain rate e.g. communicating with other devices or reading a sensor value [18]. Soft real time performance are measured in terms of time consistency. The metrics that are relevant are listed below.

- **Interrupt latency**, the time from an interrupt happens to when the handling is invoked.
- **Threads fly-back time**, the time to return to a thread from an interrupt.
- **Context switch time**, the time to switch between threads.

The variability in the system response time is called jitter. There are several things that can affect the jitter including time consistency mentioned above and things like if several threads needs to share the same resources. RTOSes are known for their high degree of reliability, there are certain design choices that can be made to increase this e.g. minimizing dynamic memory allocation and reeling on static allocations. The implementation is made with this in mind. [17]

# 3 Related work

This section presents related work based on a literature study, targeting step counting algorithms and open source wearables. This literature study yielded two main types of relevant references to our study; (1) studies of open source wearables and step counting algorithms, and (2) studies sharing technical concerns relevant to the implementation and testing of our work. These references are presented in situ within section 4 and 5 where their relevance to this particular study can also be seen, in addition to the section below for the background information to the problem area.

## 3.1 Open-source wearables

Current research points to a lack of open-source wearables in the market and in research. While some open-source solutions for medical devices have been proposed, the number of implementations is fewer. Bhat et al. [19] have introduced a suggestion for an open-source platform for wearable health monitoring. In their research they present challenges and solutions for wearable health, summarized in Figure 1.

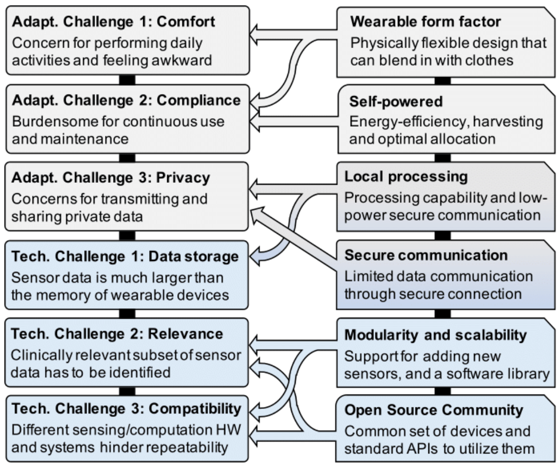
Adapt. Challenge 1: Comfort
Concern for performing daily activities and feeling awkward
Adapt. Challenge 2: Compliance
Burdensome for continuous use and maintenance
Adapt. Challenge 3: Privacy
Concerns for transmitting and sharing private data
Tech. Challenge 1: Data storage
Sensor data is much larger than the memory of wearable devices
Tech. Challenge 2: Relevance
Clinically relevant subset of sensor data has to be identified
Tech. Challenge 3: Compatibility
Different sensing/computation HW and systems hinder repeatability
Wearable form factor
Physically flexible design that can blend in with clothes
Self-powered
Energy-efficiency, harvesting and optimal allocation
Local processing
Processing capability and low-power secure communication
Secure communication
Limited data communication through secure connection
Modularity and scalability
Support for adding new sensors, and a software library
Open Source Community
Common set of devices and standard APIs to utilize them

**Figure 1:** Challenges and solutions of wearable health platforms by Bhat et al. [19]

Several of the mentioned challenges, e.g. privacy, data storage and compatibility, are useful to consider in this research to achieve the purpose and answer the research questions. Among the list of proposed solutions, energy-efficiency and opens-source community are related to what this work is about.

## 3.2 Step counting algorithms and testing accuracy

In a study done by Brajdic and Harle [8] several step counting algorithms have been compared. Authors concluded that the best accuracy and performance are obtained by the windowed peak detection algorithm. This algorithm was later further optimized in the study by Salvi et al. [9]. They analyzed different methods for each stage of the windowed peak detection algorithm and implemented it in Java for use in smartphone applications. Authors have also made their implementation open-source and released an associated dataset with raw accelerometry and ground truth steps.

For what regards the testing of step counting algorithms, Cho et al. [20] evaluate their algorithm's accuracy by testing the step counting in different wrist positions and different walking speeds on a treadmill. Case et al. [5] tested commercially available fitness trackers for step counting accuracy by walking 500 and 1500 steps on treadmills. In Brajdic and Harle [8], instead, the researchers encouraged users to walk at varying speeds without a treadmill in order to get more realistic walking samples while video recording the activity.

The results from research done by Brajdic and Harle [8] and the implementation by Salvi et al. [9] is useful for this project. The way of testing the algorithm to evaluate the accuracy, other studies' methods are interesting. Methods such as using a treadmill in various speeds and not using a treadmill in addition to different wrist positions are useful to get a complete picture of the accuracy and a point of comparison with existing accuracy data.

# 4 Method

The method adopted in this project is based on the framework described by Nunamaker et al. [21]. The Nunamaker method is chosen due to it being systematical and having an ability for iteration which is utilized when designing the system. Our approach is mapped in relation to Nunamaker et al. in Table 1.

**Table 1:** The method description and our approach

| Nr | Process | Issue | Our approach |
|---|---|---|---|
| 1 | Construct a conceptual framework | •State a meaningful research question<br>•Investigate the system functionalities and requirements<br>•Understand the system building processes/procedures<br>•Study relevant diciplines for new approaches and ideas | •General problem statement from previous research<br>•Literature study, based on the identified key concepts of the problem area<br>•Problem breakdown into subproblems and then tasks |
| 2 | Develop a system architecture | •Develop a unique architecture design for extensibility, modularity, etc.<br>•Define functionalities of system components and interrelationships among them | •Evaluate the selected algorithm<br>•Choosing a device for testing algorithm.<br>•Choosing a RTOS with an implementation for the selected device.<br>•Programming environment and tools |
| 3 | Analyze and design the system | •Design the database/knowledge base schema and processes to carry out system functions<br>•Develop alternative solutions and choose one solution | •Optimizing algorithm for embedded devices<br>•Selecting a suitable open-source licence |
| 4 | Build the prototype system | •Learn about the concepts, framework, and design through the system building process<br>•Gain insight about the problems and the complexity of the system | •Porting algorithm<br>•Testing the RTOS implementation<br>•Integration of algorithm |
| 5 | Observe and evaluate the system | •Observe the use of the system by case studies and field studies<br>•Evaluate the system by laboratory experiments or field experiments<br>•Develop new theories/models based on the observation and experimentation of the system's usage<br>•Consolidate experiences learned | •Fine-tuning of parameters<br>•Testing ground truth device to verify accuracy<br>•Testing algorithm with existing dataset<br>•Testing accuracy with participants |

## 4.1 Construct a conceptual framework

The first step in the research process is to define the general problem. Based on the problem, a research question and subquestions are set up to solve the problem. A literature study is made to better understand the aim of the project. Databases as IEEE Xplore and ACM Digital Library as well as Google Scholar are used to find published research reports. By using selected keywords the results are limited to relevant articles in the area of step counting algorithms, wearable devices, open-source projects and privacy issues. Beyond that, a broader search on internet is conducted to find relevant theoretical aspects to better understand the used software and hardware. Used keywords are:

- step counting
- pedometer
- algorithm
- accuracy
- wearable
- health
- open-source
- privacy

Sorting out relevant research article is done by reading the abstracts to get an overview of the contents. For those papers that seemed to be relevant, the methods, results and discussions is read as well. Useful articles are saved and organized in folders in a tool called Zotero. The folders; Open-source wearable, privacy issues, step-counting and healthcare, helps to make sure that all areas are covered of the literature study.

In addition to the literature research, to get a better overview what has to be done, a problem breakdown is made. The main problem have been broken down to subproblems and tasks and are presented in a function tree [22, p.43] (see Figure 3).

## 4.2 Develop a system architecture

This step involves the design of a system architecture. The existing architecture by Salvi et al. [9] forms the basis of this project since, according to the literature study, they have used the most optimal algorithm. Each step of the algorithm has been evaluated, their study is open-source and there is an implementation (although for smartphone applications). This is used as a base to expand the algorithm's use for working in several different devices with smaller processors and bigger limitations e.g. without a floating point unit and with less available RAM. Furthermore they have proved that their work shares the quality attributes; reliability and accuracy.

During this phase, a device with an existing RTOS implementation needs to be chosen as well. It is also important to select a suitable programming environment and tools to aid the development and debugging in the later stages.

## 4.3 Analyze and design the system

To analyze and design the system the implementation given by Salvi et al. [9] is examined to determine what needs to be modified to work in an embedded environment. The first modification is the language. Their implementation is written in Java, running in a virtual machine. A virtual machine takes the bytecode and interprets it into machine code. This is a performance overhead that can be removed by using a language that compiles directly into machine code. Therefore we chose C as our language because it compiles directly to machine code for many different processor architectures, which makes it highly portable as well. Another consideration for optimization into embedded devices is that some embedded processors lack an *FPU* (Floating Point Unit) and instead emulate this, which takes many CPU-cycles to compute. Our implementation thus removes all need for floating point arithmetic, in order to further optimize performance.

Since the implementation is open-source, a suitable license needs to be chosen. There exists a wide range of licenses from permissive to restrictive. One resource to find a fitting license is *choosealicense.com* [16].

## 4.4 Build the (prototype) system

To build the system, an iterative process is used to first port the algorithm to C, and then further optimize it for use in embedded systems. Similarly, the code for the wearable device is also further developed using an iterative process, where the algorithm is integrated into the RTOS with other features such as the display, graphics, Bluetooth and the accelerometer drivers.

## 4.5 Observe and evaluate the system

The algorithm itself is tested for accuracy during this stage with an existing dataset [23]. The dataset is gathered using a smartphone and through the Android API, which translates the raw accelerometry data into force. The dataset was acquired holding the phone in different positions. The algorithm is then integrated into the firmware for the watch. After the integration, validation testing is conducted with the same tests as Cho et al. [20] with different wrist positions and walking speeds on a treadmill. To get more realistic data that can be applied in everyday usage, the method proposed by Brajdic and Harle. [8] is used. Their approach is to test outside with different people that walk 500 steps with a varied speed.

For validating, the test is based on 8 different scenarios listed below. Each part is done at 500 steps with 10 participants. The participant will wear 2 PineTime watches on the same arm. One watch with the C implementation and one watch with the built-in step counter.

- Pocketed hands, hands in pockets at 4.5 $^{km}/_{h}$ on a treadmill.
- Grabbed phone, arms half way extended and stationary at 4.5 $^{km}/_{h}$ on a treadmill.
- Folded arms, arms crossed on chest at 4.5 $^{km}/_{h}$ on a treadmill.
- Slow walking, arms swinging at 2.5 $^{km}/_{h}$ on a treadmill.
- Normal walking, arms swinging at 4.5 $^{km}/_{h}$ on a treadmill.
- Fast walking, arms swinging at 6.5 $^{km}/_{h}$ on a treadmill.
- Normal running, arms swinging at 8.5 $^{km}/_{h}$ on a treadmill.
- Normal walking on mixed outdoor terrain.

**Ground truth**

To have a reliable ground truth, a Polar stride sensor [24] placed on participants' shoe is used in each test as well as an observer whom counts the steps to 500. The sensor has no step counter but is counting the cadence every second. Cadence is the number of times per minute the foot with the sensor hits the ground. The number of steps is therefore counted by multiplying total duration with 2 times average cadence. To make sure the Polar stride sensor is a reliable ground truth, the wearable is tested by one participant, outside on mixed grounds (gravel and asphalt). The value of ground truth is at last calculated as the average of 500 and the stride sensor value. To then compare the output of our implementation to the ground truth, *percentage error* is used (see Formula 1).

$$Error\ (\%) = \left| \frac{detected\ value - ground\ truth}{ground\ truth} \right| \cdot 100 \tag{1}$$

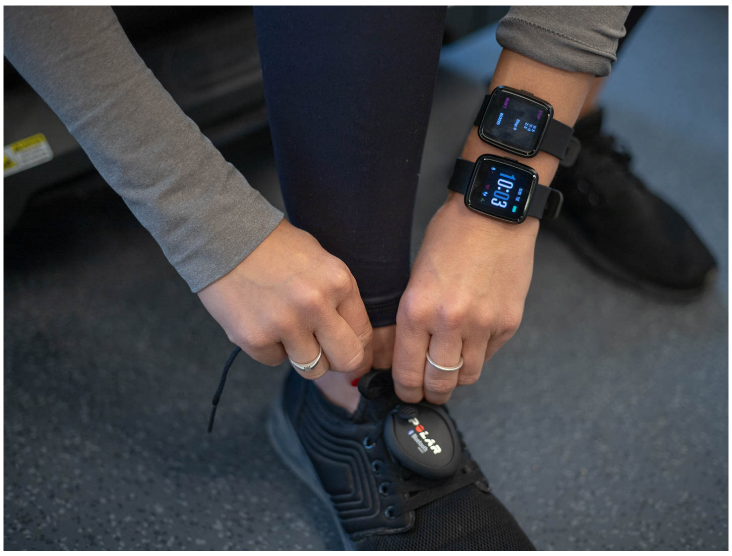

**Figure 2:** Test equipment

## 5 Results

### 5.1 Construct a conceptual framework

Based on the literature study described in section 4.1, five references make up the core foundation for the conceptual framing of this study (Table 2).

**Table 2:** An overview of related work and relevance to this study

| Author | Content | Relevance |
|---|---|---|
| Bhat et al. [19] | An open source platform for use in healthcare applications | Challenges and solutions to common issues in wearables |
| Brajdic & Harle [8] | Compared different algorithms for step counting | An overview for choosing algorithm |
| Salvi et al. [9] | Implemented windowed peak detection and optimized it | A starting point for this study |
| Cho et al. [20] | Evaluated accuracy in their algorithm using different wrist positions | Used for testing the implementation |
| Case et al. [5] | Compared different commercially available fitness trackers | A comparison to the wearable in this study |

Since this research project is an extension of a study done by Salvi et al. [9] their research paper is useful as a starting point for this study. The paper is published in 2018 and could be found on IEEE Xplore with keywords; algorithm, step counting, accelerometer and accurate. Searching for step counting, health and algorithm, using the ACM Digital Library reveals the most cited research articles as the article by Brajdic and Harle [8]. This paper is also used as a basis in the study by Salvi et al.

The article by Baht et al. [19] shows up in the top results on IEEE Xplore, when sorting on relevance and using keywords as Open-source, wearable and health. The paper is published 2019 and presents ideas and solutions for an open-source platform for wearable health monitoring. The authors have, however, not implemented the platform they propose.

Another article is found on IEEE Xplore with the keywords; pedometer, algorithm and wearable. Cho et al. [20] did a study where the testing part is relevant for this project. Some relevant articles were also found when searching in Google Scholar. The research by Case et al. [5] is one of them by using keywords as accuracy, wearable and health.

Using the literature identified through our survey of related work, the problem at hand could be broken down and defined more clearly in terms of what needed to be done. From a software development perspective, the result from this problem breakdown is represented by a function tree and is presented in Figure 3. The red box is the main problem, while the yellow boxes are subproblems and the green boxes represent the associated tasks.

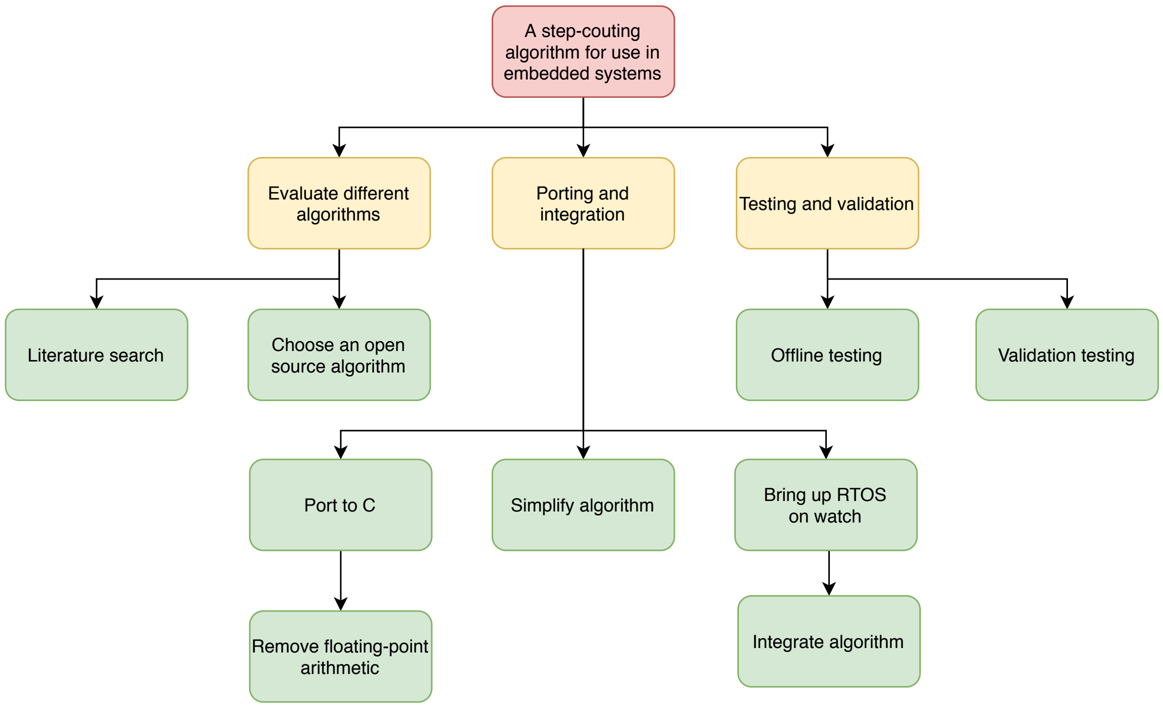

**Figure 3:** Main problem broken down in a function tree

## 5.2 Develop a system architecture

For the wearable device a PineTime [25] is chosen because it is able to run custom built open-source software. The accelerometer inside PineTime has built-in intelligence for step counting that will be used as a comparison to the implemented algorithm. There exists an implementation of the Zephyr RTOS [26] for this wearable. Zephyr is a modern RTOS built with security and safety in mind, therefore it shares the quality attributes (reliability and privacy) with our project. Zephyr is described in more detail with its context in a later section (see 5.4.2).

The project is built with two compilers; GCC[13] and Clang[14]. The compilers support different features which is useful for the optimization. Clang is useful to find memory and address leakage as well as undefined behaviour e.g. integer-overflows. GCC is able to provide performance statistics by compiling each function with an call to *mcount*. Mcount is a function that counts the number of calls and execution time a function has, the tool *gprof* is then used to present this data in an human readable form.

To handle version management, git and GitHub are utilized. Git includes submodules, meaning it is possible to have repositories of code inside other repositories. Figure 4 shows how the project is built. The red box is an external dependency, the yellow box is the algorithm and the green boxes are repositories used to develop the algorithm.

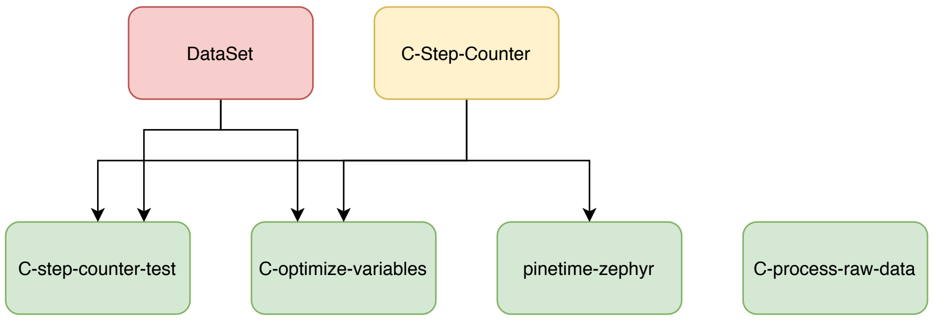

**Figure 4:** The project is built with a submodule and several repositories

## 5.3 Analyze and design the system

The implementation is licensed as open-source and MIT licence is chosen because it is a permissive license. This means that anyone is allowed to distribute, modify, use it commercially or in private applications if they include a copy of the licence. It also includes that the software is distributed as it is, meaning we provide no warranty and are not liable for any issues experienced by those that use our code.

To design the system, strategies for optimising the algorithm is needed. The first is to remove the assumption that a FPU is available. Square roots are only supported for floating point in the standard C library *Math.h*. Therefore an integer square root is needed. This proved to be a challenging task where *CORDIC* (Coordinate rotation digital computer) [27] needed to be utilized. CORDIC is a way to implement many functions using only addition, subtraction, bitshift and table lookup, which is further used to calculate the square root [28]. Although it is not exact, to increase the accuracy of CORDIC it needs more iterations, and therefore takes longer to compute. A midpoint between accuracy and performance therefore needed to be made.

The algorithm needs buffers between each of the stages (see Figure 5) as some of them needs old samples as well. To optimize this, efficient buffers between each stage is needed and since this algorithm is used in RTOS-applications, static is preferred to dynamic allocations. An existing ring buffer implementation, with MIT licence, was found [29]. As we share the same licence, the implementation could be integrated into this project seamlessly.

## 5.4 Build the (prototype) system

The algorithm was developed using an iterative approach starting from the Java implementation by Salvi et al. [9]. The first iteration was to rewrite the algorithm to C. The next iteration was to remove floating point arithmetic and then the last iteration of the algorithm was to optimize it for use in embedded systems. This is described in more detail later in this chapter.

The code for the PineTime was also developed using an iterative approach where the Zephyr RTOS implementation [30] was the starting point. This was then evaluated to work on our watch. The next iteration was to integrate the algorithm into Zephyr and the last iteration was to test and validate that the algorithm was correctly integrated.

### 5.4.1 The algorithm

The algorithm consists of several discrete parts as described by Salvi et al. [9] and shown in Figure 5. Their original implementation uses independent threads with synchronized lists between them. Our implementation had to be optimised to work in embedded systems and therefore the algorithm does not use threads.

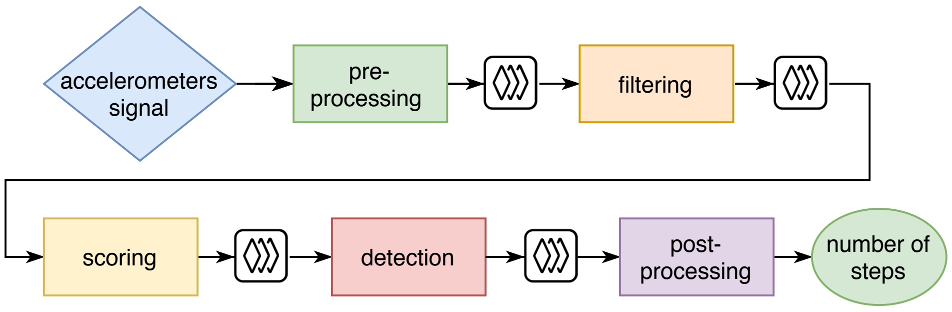

**Figure 5:** Block diagram of the algorithm used by Salvi et al. [9]

**Pre-processing**

The pre-processing takes the three orthogonal axes $(x, y, z)$ as 32-bit signed integers, from the accelerometer and combines these into one value as shown in Formula 2. This value is henceforth called the magnitude. Optionally, if the sampling method presents jitter, the magnitude values can be linearly interpolated if a flag is set.

$$m_i = \sqrt{x_i^2 + y_i^2 + z_i^2} \tag{2}$$

Since the standard C library only supports square roots for floating points, the calculation of the magnitude is done by using CORDIC square root.

**Filtering**

Since the frequencies in the data that are relevant for the algorithm are below $3Hz$ as established by Hoeger et al. [31], a low-pass filter is used to cut off the noise from higher frequencies. This is implemented using a *FIR-filter* in both the original and this implementation, both utilized a Gaussian window function with a length of 13 and a standard deviation of 0.35. The main difference is that the C implementation uses pre-calculated coefficients.

**Scoring**

To make detection easier in the next block, a scoring stage is used to exaggerate the peaks of the signal. The implementation uses mean difference to find the peaks in magnitude, shown in Formula 3 where $x$ is the content of the buffer and $N$ is a fixed value considered for optimization, described below.

$$p_i = \frac{\sum_{k=-N,k\neq i}^{N}(x_i - x_{i+k})}{2N} \tag{3}$$

**Detection**

This stage is used to statistically determine outliers in the data i.e. the peaks. This is done by calculating the mean $\mu$ and standard deviation $\sigma$ of each sample, then taking the difference between the current magnitude $x_i$ and the mean and comparing it to the standard deviation times a threshold $th$. The threshold is a fixed value considered for optimization, described below. If the difference between the current magnitude and the mean is bigger than the standard deviation times the threshold a peak is detected. The entire formula (4) is described below.

$$x_i - \mu > \sigma \cdot th \tag{4}$$

Given that the threshold is a floating point number, we replaced this formula with two integers; $th_{whole}$ and $th_{frac}$ according to Formula 5:

$$x_i - \mu > \sigma \cdot th_{whole} + \frac{\sigma}{th_{frac}} \tag{5}$$

**Post-processing**

Post-processing handles false positives from the previous stage by having a sliding window of fixed size $t_{window}$ and only using the maximum value within the window. The implementation uses $t_{window} = 200ms$ that corresponds to 5 $^{\text{steps}}/_{\text{second}}$, which is well above the normal walking speed according to Hoeger et al.[31], but $t_{window}$ is still considered for optimization.

### 5.4.2 Integration of the algorithm in a smartwatch firmware

The firmware was developed using an iterative approach where the Zephyr Operating System was used for the PineTime smartwatch. The algorithm was integrated into the operating system and other essential tools were developed in addition to step counting such as Bluetooth communication and the display. Bluetooth is used to send raw data from the device and the display is used to start, stop and reset the algorithm in addition to see the current amount of steps.

**Initial tests of Zephyr RTOS**

During this iteration the focus was to validate the Zephyr implementations [30] ability to provide the necessary functionality to support our project. Mainly get Bluetooth, display and accelerometer drivers in working order. The tests concluded that the driver for the BMA421 accelerometer was incomplete and did not provide any data. This was fixed by using the drivers from another PineTime project and modifying those. Bluetooth and display both worked but using both at the same time caused a spike in RAM usage to over 90%. We tried to minimize the RAM usage but as soon as the Bluetooth is enabled it allocates a lot of RAM. Therefore we decided to split the project into two different firmware. One firmware for data collection with only accelerometer and Bluetooth enabled to send the raw data to a phone for collection to later be optimized. The other firmware is for testing the algorithm with accelerometer and screen enabled. A *GUI* (graphical user interface) was created for the testing firmware with the raw values, current step count, time since boot in milliseconds, a stop button and a reset button (see Figure 6).

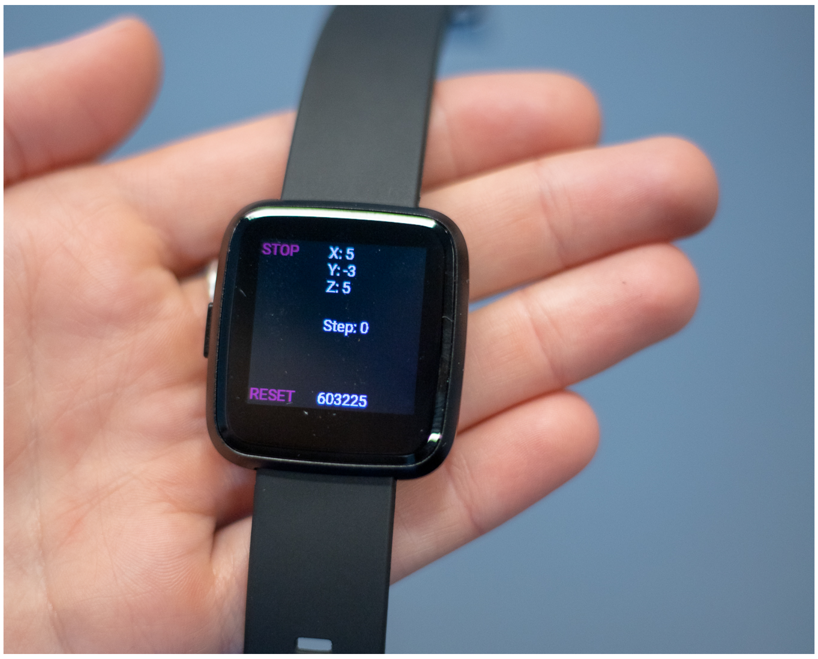

**Figure 6:** The GUI for the algorithm testing firmware

**Integration of the algorithm**
To integrate the algorithm the first step was to optimize it for PineTime. The firmware to send raw accelerometer data over Bluetooth was utilized and then captured on a smartphone using the nRF Connect app [32]. To save battery, two samples at time were sent over Bluetooth, thus buffering one sample and sending only when the second sample is available. Table 3 shows how the 20 byte packet is structured. Time is in milliseconds since boot. The signed values had their byte order rearranged as the watch uses big endian and the computer uses little endian. This was done to make it possible to restore them in the computer to a readable CSV format using *C-process-raw-data* to further process it into *C-optimize-variables* to get the optimized constants for the algorithm. *C-process-raw-data* is a script written in C that takes the logs from the nRF connect app and processes them into a CSV file. *C-optimize-variables* is a grid search to optimize the algorithm parameters where a set range of the values are specified and the algorithm is running with all possible combinations of them to find optimal solutions. The parameters that are optimized are $N$ in the scoring stage, $th_{whole}$ and $th_{frac}$ in the detection stage and $t_{window}$ in the post-processing stage.

The algorithm is integrated in the main loop of Zephyr, a temporary metric for difference in timing was made and displayed on the screen to see the jitter. Since this value was small, linear interpolation was disabled.

**Table 3:** Bluetooth packet structure

| | | | |
|---|---|---|---|
| Unsigned 32-bit Time | Signed 16-bit X | Signed 16-bit Y | Signed 16-bit Z |
| Unsigned 32-bit Time2 | Signed 16-bit X2 | Signed 16-bit Y2 | Signed 16-bit Z2 |

## 5.5 Observe and evaluate the system

Before integrating the algorithm into the watch, several tests were done on the computer using a test program that runs the algorithm with an existing dataset [23]. The dataset includes several tests from different wearing position. Table 4 shows how the C implementation corresponds to ground truth compared with the Java implementation. The accuracy in all following tables are calculated as 100 - *percentage error*.

**Table 4:** Results from the offline testing.

| **Position** | **C implementation** | | **Java implementation** | |
|---|---|---|---|---|
| | Accuracy (%) | | Accuracy (%) | |
| | Median | Min/Max | Median | Min/Max |
| Hand | 88 | 80 / 99 | 94 | 67 / 99 |
| Armband | 89 | 82 / 97 | 94 | 91 /99 |
| Back pocket | 96 | 82 / 100 | 96 | 87 / 99 |
| Front pocket | 97 | 88 / 100 | 97 | 85 / 100 |
| Neck pouch | 98 | 83 / 99 | 99 | 86 / 100 |
| Purse | 84 | 62 / 90 | 87 | 59 / 97 |
| **Total (average)** | **92** | **80 / 98** | **95** | **79 / 99** |

To test the integration, smaller walks of roughly 50-100 steps were captured. The data was then used to optimize parameters. The main problem encountered here was hypersensitivity, which produced many false positives. To oppose this, the sensitivity was extended from $2G$ to $16G$ and resolution decreased from 12 bit to 10 bit by removing the last 2 bits with bit shifting. These two changes made the accelerometer less sensitive which produced better results. To increase power efficiency the sample frequency was decreased to 50Hz without a significant drop in accuracy. This is shown in Table 5, noteworthy is that the accuracy is optimized for just that data. This is why the accuarcy is extraordinarily high. Grid search is used to optimize the algorithm parameters where a set range of values are specified and the algorithm is running with all possible combinations of them to find optimal solutions. The parameters that are optimized are $N$ in the scoring stage, $th_{whole}$ and $th_{frac}$ in the detection stage and $t_{window}$ in the post-processing stage. The final constants are scoring window size $N = 40$, detection threshold $th_{whole} = 2$, $th_{frac} = 2$ and post processing time $t_{window} = 200ms$.

**Table 5:** Results from the optimization

| | **Accuarcy (%)** |
|---|---|
| 16G, 12 bit, 100Hz | 95.1 |
| 16G, 10 bit, 100Hz | 99.6 |
| 16G, 10 bit, 50Hz | 99.2 |

The Polar stride sensor was tested to verify its accuracy. One participant performed 4 walking and 3 jogging tests outside on various ground to collect data. The error is the difference between steps as counted manually and the steps reported by the sensor. Results from the tests are presented in Table 6 below.

**Table 6:** Results for verifying accuracy in Polar stride sensor.

| | **Steps** | **Accuracy (%)** |
|---|---|---|
| Walking | 45 | 91.1 |
| Walking | 181 | 100 |
| Walking | 335 | 99.7 |
| Walking | 500 | 99.6 |
| Jogging | 70 | 98.6 |
| Jogging | 97 | 97.9 |
| Jogging | 326 | 99.7 |

## 5.6 Final main result

10 participants performed 7 walking and 1 running tests to collect data. The participants are 6 females and 4 males and their age is between 21 and 60 with a median of 25. The built-in step counter refers to the step counter in the BMA421 accelerometer. The test results were summarized by calculating the median, minimum and maximum accuracy of the different speeds and arm positions. With this, final values are presented as the *total* in Table 7.

**Table 7:** Results from PineTime testing.

| | **C implementation** | | **Built-in step counter** | |
|---|---|---|---|---|
| | Accuracy (%) | | Accuracy (%) | |
| | Median | Min/Max | Median | Min/Max |
| Pocketed hands | 91 | 77/98 | 99 | 91/100 |
| Grabbed phone | 84 | 53/99 | 99 | 87/100 |
| Folded arms | 94 | 70/100 | 99 | 97/100 |
| Slow walking | 82 | 41/96 | 71 | 22/97 |
| Normal walking | 95 | 80/99 | 99 | 93/99 |
| Fast walking | 92 | 83/97 | 98 | 93/100 |
| Normal running | 97 | 84/100 | 97 | 93/100 |
| Walking outdoor | 98 | 95/99 | 99 | 95/100 |
| **Total (average)** | **92** | **73/99** | **95** | **84/99** |

# 6 Analysis and Discussion

## 6.1 Motivation and approach to the study

Even if step-counting is an established technology, there is continuous innovations in this area. However, even though a number of wearables and other products are becoming open-source, algorithms for processing their data are not following the same tendency. This development is therefore needed to widen the availability of open source tools for health and well-being. For research purposes, having full access to the source code as well as integration with an open source based wearable provides transparency and control, allowing future research to expand or adapt the implementation as needed.

The reason for using Nunamaker et al. [21] as a basis for the methodological framework was the systematic construction and the ability for iterations. Compared to *Design Science Research Methodology* which has a similar construction, Nunamaker et al. is not as elaborate and gives more leeway to explore and customize the methodology for the specific project.

The development environment, RTOS and tools have been adequate for our purposes. We encountered issues related to the RTOS implementation and the missing accelerometer driver. This could have been avoided by using an RTOS with this already integrated. Since the thesis was developed with a limited budget, free alternatives were used e.g. for performance measurement we utilized gperf, but there exists proprietary tools such as Arm MAP that are much more advanced and give a better overview.

## 6.2 Analysis of result

The aim of this thesis was to implement and test an open-source algorithm for step counting with a known accuracy. The results showed that an accuracy of 73% to 99% was possible in the final implementation, depending on the pace and position of the wearable (e.g. pocketed hands, folded arms, or free swinging). These results are promising as a first implementation, as commercially available wearables have been shown to have an accuracy range between 77.3% and 98,5% [5]. While further optimization is needed, the open source based approach we have taken is well positioned to invite such work.

The results of offline testing in Table 4 displays slightly different results than the original Java implementation. We believe this to be rooted in the conversion from floating point to integer and the CORDIC square root algorithm as it is a balance between exact results and performance. It is possible to increase the accuracy in this, but as the step counting algorithm also had a goal to be as efficient and portable as possible in our implementation, this also affected the final accuracy of our tests. Subsequently, the slightly lower accuracy than Salvi et al. [9] may also be attributed to an active design choice to promote efficiency and portability.

In Table 6 the Polar stride sensor is accuracy tested to be used as the ground truth device. This is a useful device for this purpose as it is designed solely for step analysis and not as sensitive for different walking positions as the smartwatch we have worked with. The first test produced an accuracy much lower than expected (91.1%). As we suspected that the reason for this was loss of Bluetooth signal to the phone, we monitored this connection in all further tests and observed no more unexpected low results.

Kang et al. [33] observe that a used method does not always perform best in every walking activity. Based on our results in addition to results from Cho et al. [20], Case et al. [5] and Salvi et al. [9] there are in every study one or several activities that exhibit lower accuracy than the others, which confirms this observation. It also means that our wide range of accuracy is a common limitation in most implementations.

Table 7 presents some interesting variation in the accuracy, mainly related to *slow walking*, which has the lowest values both for the algorithm (82%) and the built-in step counter (71%). In the study by Cho et al. [20] where different velocities and wrist positions were tested, their algorithm showed a higher accuracy in the slow walking test (99.1%). Most of the devices tested by Cho et al. showed high accuracy but *STK smart band* and *iPhone 6 health app* produced similar results to ours. The STK smart band had an accuracy in 68% and the iPhone 6 health app 60%. A possible explanation for our result is that the data used to optimize our algorithm was collected with a mixed walking speed that did not include slow walking at 2.5 $^{km}/_{h}$. The low accuracy regarding to slow walking, with our algorithm but also some other commercial available devices, may therefore affect people with slower walking because of age or some disabilities.

The constants of the algorithm could be further improved for the slow walking scenario. The FIR filter coefficients are not of importance for this since it is a low-pass filter and therefore let through even the slowest of walking. The sliding window is one of the main constants that could be a source of error, we utilized a window size of 40 and a sample frequency of 50 $Hz$ meaning that 0.8 *seconds* are processed. If the $^{\text{steps}}/_{\text{seconds}}$ is slower than that the peaks will not be as visible. Detection threshold is probably the constant of greatest importance for accuracy in every scenario, Wong et al. [34] present in their study a hypothesis that the heel strike might not be as strong when walking slower, therefore producing a lower signal to noise ratio. In our algorithm this means that the threshold is not reached and the step is not counted. The last constant is the time threshold and since this is to make sure false steps are not counted, it is not of importance for slow walking. Since the PineTime watch did not present a suitable way to run the algorithm and Bluetooth at the same time, data was not collected that could be plotted to show this. This discussion is based from a theoretical standpoint.

Furthermore lower accuracy in *grabbed phone* was also of notice, with an accuracy of 84% for the algorithm. A possible explanation for this is the static position of the arms whilst holding a phone. Since the resolution was lowered and the range increased of the accelerometer, it is possible that the signal became too weak in this static position.

Another inconsistency from Table 7 is *walking outdoor*, where both the algorithm and the built-in step counter displayed much better results than the other categories (98% and 99%). Our algorithm had not just the best but also the smallest range in the outdoor test, the lowest accuracy was 95% and the highest 99%. This could also be because of the type of data we used for optimization. All the collected data for optimization was collected outdoor and is essentially what the algorithm was made for.

**Table 8:** Reflection points from result

| **Test** | **Part** | **Reflection** |
|---|---|---|
| Offline | Tests in general | Not as accurate as Java implementation because of integers |
| Polar | First test | Not as accurate as the others because of possible lost BLE connection |
| PineTime | Slow walking | Not as accurate as the others because of not beeing optimized for that slow velocity |
| PineTime | Grabbed phone | Not as accurate as the others because of a too stationary wrist position |
| PineTime | Walking outdoor | More accurate than the others because of similarity to the optimization tests |

The utilized accelerometer Bosch BMA421 has built-in intelligence for step detection and counting and is used as a point of comparison. In the conducted tests the accuracy of the built-in step counter ranged from 84% to 99% which is higher than the one produced by our algorithm. However, that algorithm is not open-source and only exists in certain Bosch Sensortec products, but is still valuable as a comparison.

The possible sources of error is the CORDIC square root algorithm and the constants of the step counting algorithm. The CORDIC square root was the fastest algorithm we could find, but also not the most accurate. It is possible to increase the accuracy of it, but that would affect CPU utilization and therefore power consumption. The constants could have been optimized further with more participants to get a greater variety in the data.

# 7 Conclusions

## 7.1 Answering the Research questions

Summarizing this project, the aim was to answer the main research question:

> **RQ1:** How can an open-source algorithm for step counting, with a known accuracy for use in healthcare-applications, be implemented?

The subquestions were set to be a guideline, helping us to answer the main question. To build an algorithm for step counting, we needed to be able to count steps by converting and processing raw accelerometery data. That was done by answering subquestion 1.1:

> **RQ1.1:** How can steps in fitness trackers be counted from raw accelerometery data?

The subquestion have been answered in the result section where we describe how the accelerometer's three signals converts into one value, passing through a FIR-filter to cut off noise, exaggerate the peaks before outliers are detected. If the time difference between two outliers are bigger than a set value, a step is counted (see 5.4.1).

To make the algorithm work in many different devices even those with smaller processors, it needs to be portable. Hence, subquestion 1.2 was formed.

> **RQ1.2:** How can the algorithm be made portable?

To answer this we chose to work with the programming language C as it is a quite old and stable language with support for many embedded processors. A consideration for portability was that not all embedded devices contains a FPU, because of this, only integers was utilized (see 4.3 and 5.3).

The algorithm also needed a known accuracy, which means we wanted to find a metric which describes the overall reliability of the algorithm's results. Subquestion 1.3 is devised to answer this.

> **RQ1.3:** How can the step counting accuracy of the algorithm be determined?

By utilizing both offline testing with an existing data set and setting up different test scenarios for 10 participants to conduct, data was collected to determine the algorithm's accuracy. By comparing the algorithm's value with ground truth, the accuracy was calculated as 100 - *percentage error* and are summarized in Table 4 and Table 7.

Since the algorithm will be used in battery powered devices, it is of importance to try to reduce the battery drain. Considering this, we formulated subquestion 1.4.

> **RQ1.4:** How can the algorithm be optimized and fine tuned in order to reduce battery drain?

This was done by using only integers instead of floating points, which will improve performance in some devices (see 4.3), striking a balance between accuracy and performance in the CORDIC algorithm (see 5.3) and lastly decreasing the sample frequency (see 5.5). Noteworthy is that this is only speculative as a way to measure battery usage was not presented.

The conclusion of this thesis is an open-source algorithm with a median accuracy of 92% (average min/ max was 73%/99%) in the PineTime implementation.

## 7.2 Contribution

As stated in 1.2 this thesis was developed within an initiative that aims at building an entirely open-source firmware for fitness trackers and smartwatches for use in healthcare. This thesis has laid the ground work for further developments towards this goal. The source code [1] is available on GitHub and is published under MIT Licence.

## 7.3 Future work

This thesis covers the basic integration on the watch in addition to the step counting algorithm. Future work will be to further optimize the algorithm e.g. by improving the FIR filter coefficients, optimizing the constants with help from more participants and in different scenarios. The different scenarios should include the ones where the algorithm performed the worst i.e. slow walking and grabbed phone. To further develop the firmware, battery drain needs to be measured. Since the PineTime watch did not present a suitable way to measure this, a more suitable device or method needs to be chosen.

Since the initiative this thesis was developed in aims to build an entire firmware for fitness trackers. Further developments includes more algorithms for e.g. photoplethysmography, an companion app for the patient to see their data and maybe also a cloud service so a physician can monitor their patients.

## References

[1] A. Brondin and M. Nordström. (2020). C-step-counter, [Online]. Available: https://github.com/Oxford-step-counter/C-Step-Counter (visited on 05/27/2020).

[2] J. McCarthy. (2019). One in five U.S. adults use health apps, wearable trackers, [Online]. Available: https://news.gallup.com/poll/269096/one-five-adults-health-apps-wearable-trackers.aspx (visited on 03/02/2020).

[3] Fitbit. (2019). Fitbit charge 3, [Online]. Available: https://www.fitbit.com/us/products/trackers/charge3 (visited on 03/02/2020).

[4] A. Ozanne, D. Johansson, U. H. Graneheim, K. Malmgren, F. Bergquist, and M. A. Murphy, "Wearables in epilepsy and parkinson's disease—a focus group study", *Acta Neurologica Scandinavica*, vol. 137, no. 2, pp. 188–194, 2018, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ane.12798, ISSN: 1600-0404. DOI: 10.1111/ane.12798. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/ane.12798 (visited on 03/11/2020).

[5] M. A. Case, H. A. Burwick, K. G. Volpp, and M. S. Patel, "Accuracy of smartphone applications and wearable devices for tracking physical activity data", *The Journal of the American Medical Association*, vol. 313, no. 6, pp. 625–626, Feb. 10, 2015, ISSN: 1538-3598. DOI: 10.1001/jama.2014.17841.

[6] R. Goyal, N. Dragoni, and A. Spognardi, "Mind the tracker you wear: A security analysis of wearable health trackers", in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ser. SAC '16, event-place: Pisa, Italy, New York, NY, USA: Association for Computing Machinery, 2016, pp. 131–136, ISBN: 978-1-4503-3739-7. DOI: 10.1145/2851613.2851685. [Online]. Available: https://doi.org/10.1145/2851613.2851685.

[7] J.-H. Hoepman and B. Jacobs, "Increased security through open source", *Communications of the ACM*, vol. 50, no. 1, pp. 79–83, Jan. 1, 2007, ISSN: 0001-0782. DOI: 10.1145/1188913.1188921. [Online]. Available: https://doi.org/10.1145/1188913.1188921 (visited on 02/15/2020).

[8] A. Brajdic and R. Harle, "Walk detection and step counting on unconstrained smartphones", in *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, ser. UbiComp '13, Zurich, Switzerland: Association for Computing Machinery, Sep. 8, 2013, pp. 225–234, ISBN: 978-1-4503-1770-2. DOI: 10.1145/2493432.2493449. [Online]. Available: https://doi.org/10.1145/2493432.2493449 (visited on 05/26/2020).

[9] D. Salvi, C. Velardo, J. Brynes, and L. Tarassenko, "An optimised algorithm for accurate steps counting from smart-phone accelerometry", in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Honolulu, Jul. 2018, pp. 4423–4427. DOI: 10.1109/EMBC.2018.8513319.

[10] T. K. (2013). Accelerometer basics, [Online]. Available: https://learn.sparkfun.com/tutorials/accelerometer-basics/all (visited on 03/03/2020).

[11] C. I. St.J. Dixon-Warren. (2010). The evolution of compact three axis accelerometers, [Online]. Available: https://memsblog.wordpress.com/2010/11/22/the-evolution-of-compact-three-axis-accelerometers/ (visited on 03/03/2020).

[12] Techopedia. (2011). What is portability?, [Online]. Available: https://www.techopedia.com/definition/8921/portability (visited on 03/03/2020).

[13] Free Software Foundation Inc. (2020). GCC, the GNU Compiler Collection, [Online]. Available: https://gcc.gnu.org/ (visited on 05/26/2020).

[14] LLVM. (2020). Clang, [Online]. Available: https://clang.llvm.org/index.html (visited on 03/11/2020).

[15] O. S. Initiative. (2020). Open source initiative, [Online]. Available: https://opensource.org/ (visited on 03/12/2020).

[16] GitHub. (2020). Licenses, [Online]. Available: https://choosealicense.com/licenses/ (visited on 03/12/2020).

[17] C. EmbeddedWare. (2017). Rtos concepts, [Online]. Available: https://web.archive.org/web/20200319002655/http://www.chibios.org/dokuwiki/doku.php?id=chibios:articles:rtos_concepts (visited on 05/27/2020).

[18] IntervalZero. (2017). Why a real-time operating system is a necessity for iot, [Online]. Available: https://www.intervalzero.com/rtos/real-time-operating-system-necessity-iot/ (visited on 03/17/2020).

[19] G. Bhat, R. Deb, and U. Y. Ogras, "OpenHealth: Open-source platform for wearable health monitoring", *IEEE Design Test*, vol. 36, no. 5, pp. 27–34, Oct. 2019, ISSN: 2168-2364. DOI: 10.1109/MDAT.2019.2906110.

[20] Y. Cho, H. Cho, and C.-M. Kyung, "Design and implementation of practical step detection algorithm for wrist-worn devices", *IEEE Sensors Journal*, vol. 16, no. 21, pp. 7720–7730, Nov. 2016, ISSN: 2379-9153. DOI: 10.1109/JSEN.2016.2603163.

[21] J. F. Nunamaker Jr, M. Chen, and T. D. Purdin, "Systems development in information systems research", *Journal of management information systems*, vol. 7, no. 3, pp. 89–106, 1990. DOI: 10.1080/07421222.1990.11517898.

[22] K. Österlin, *Design i fokus för produktutveckling : varför ser saker ut som de gör?*. Liber ekonomi, 2003, ISBN: 91-47-06535-4.

[23] Oxford-step-counter. (2017). Dataset, [Online]. Available: https://github.com/Oxford-step-counter/DataSet (visited on 05/12/2020).

[24] Polar. (2020). Stride sensor bluetooth® smart, [Online]. Available: https://www.polar.com/uk-en/products/accessories/stride_sensor_bluetooth_smart (visited on 03/17/2020).

[25] PINE64. (2019). Pinetime, [Online]. Available: https://www.pine64.org/pinetime/ (visited on 02/12/2020).

[26] The Linux foundation projects. (2020). Zephyr project, [Online]. Available: https://www.zephyrproject.org/ (visited on 03/09/2020).

[27] J. Volder, "The CORDIC computing technique", in *Papers presented at the the March 3-5, 1959, western joint computer conference on XX - IRE-AIEE-ACM '59 (Western)*, San Francisco, California: ACM Press, 1959, pp. 257–261. DOI: 10.1145/1457838.1457886. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1457838.1457886 (visited on 04/23/2020).

[28] C. É. Luxembourg. (2002). Square-root based on cordic, [Online]. Available: https://www.convict.lu/Jeunes/Math/square_root_CORDIC.htm (visited on 04/23/2020).

[29] A. Kalør. (2014). Ring-buffer, [Online]. Available: https://github.com/AndersKaloer/Ring-Buffer (visited on 03/30/2020).

[30] najnesnaj. (2020). Pinetime-zephyr, [Online]. Available: https://github.com/najnesnaj/pinetime-zephyr (visited on 03/11/2020).

[31] W. W. K. Hoeger, L. Bond, L. Ransdell, J. M. Shimon, and S. Merugu, "ONE-MILE STEP COUNT AT WALKING AND RUNNING SPEEDS", *ACSM's Health & Fitness Journal*, vol. 12, no. 1, pp. 14–19, Feb. 2008, ISSN: 1091-5397. DOI: 10.1249/01.FIT.0000298459.30006.8d. [Online]. Available: https://journals.lww.com/acsm-healthfitness/Fulltext/2008/01000/ONE_MILE_STEP_COUNT_AT_WALKING_AND_RUNNING_SPEEDS.7.aspx (visited on 03/06/2020).

[32] N. semiconductor. (2016). Nrf connect for mobile, [Online]. Available: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-mobile (visited on 04/24/2020).

[33] X. Kang, B. Huang, and G. Qi, "A novel walking detection and step counting algorithm using unconstrained smartphones", *Sensors (Basel, Switzerland)*, vol. 18, no. 1, Jan. 19, 2018, ISSN: 1424-8220. DOI: 10.3390/s18010297.

[34] C. K. Wong, H. M. Mentis, and R. Kuber, "The bit doesn't fit: Evaluation of a commercial activity-tracker at slower walking speeds", *Gait & Posture*, vol. 59, pp. 177–181, Jan. 1, 2018, ISSN: 0966-6362. DOI: 10.1016/j.gaitpost.2017.10.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0966636217309669 (visited on 06/01/2020).