Tampereen yliopisto

Timo Onnia

DEVELOPMENT AND EVALUATION OF A HEART RATE SENSOR SOFTWARE FOR TEAM SPORTS MONITORING

The Faculty Council of Computing and Electrical Engineering
Master’s thesis
May 2019

ABSTRACT

Timo Onnia: DEVELOPMENT AND EVALUATION OF A HEART RATE SENSOR SOFTWARE FOR TEAM SPORTS MONITORING
Master of Science Thesis
Tampere University
Master's Degree Programme in Electrical Engineering
May 2019

Mankind has always been passionate about competing in sports of all sorts. First known sports originate thousands of years back in time. Despite the long history of sports, the last decades and years of technological improvements are now changing the nature of sports as we know it. Better technology has provided us ways to measure, quantify and analyse athlete performance in more detailed and profound ways than ever before. These tools have led to even more disciplined training programs, pushing the limits of athletes further and further. However, especially in team sports the best exercise programs and appropriate loads of training are hard to estimate. Every athlete and team are unique, as well as the training scenarios and games where the athletes push themselves. Therefore, there is an ever-increasing need for even better and more versatile monitoring tools to meet the needs of team sports.

In this thesis the objectives were to develop and evaluate a next generation elite team sports wearable heart rate monitoring and sensor solution for the heart rate and sports analytics company Firstbeat Technologies Oy. The work consists of evaluating an existing sensor system solution Suunto Movesense and developing an application on top of it to meet the needs of demanding team sports. The developed sensor software integrates an existing physiological algorithm monitoring library into the Movesense sensor, enabling wireless real-time monitoring of athletes with analytics being run on the sensor itself. The real-time analytics of each athlete is broadcasted over BLE advertising to a client device responsible for capturing and visualizing the data to team coaches. RR-intervals of measured ECG signal are also internally saved to non-volatile memory. This ensures that no single beat is lost, making all data accessible during and after the exercises. Client communication APIs for managing the sensor and data transfers were built on top of Movesense libraries.

As part of the thesis, performance of the ECG R-peak detection algorithm of the Movesense sensor was also measured and evaluated against Firstbeat's Bodyguard 2 monitoring device in a controlled field test. Suitability of the Movesense sensor and final software are also evaluated against predefined use cases.

Development of the sensor software was performed in an iterative manner by continuously evaluating the sensor's performance based on feedback from on-going team sports field test reports. Summaries from field-test reports are provided and analysed as part of this thesis, to evaluate the sensor software development process and the performance of the sensor. Reports of memory usage and power consumptions are reported. A simple RRI-signal generator was also devised and used for testing and development purposes. Preliminary plans for continuous integration (CI) pipeline with hardware-in-the-loop simulation testing are briefly discussed.

The result of the thesis project is a market-ready consumer product Firstbeat Sports sensor, which meets the needs and requirements set by the business. The reported results show that the sensor can reliably work in the challenging use cases of professional team sports. Future work and challenges of the development process are discussed at the end of the thesis.

Keywords: heart rate monitoring, sensor system, wearable wireless technology, BLE, real-time monitoring, monitoring system for professional team sports

TIIVISTELMÄ

Timo Onnia: SYKEVYÖSENSORIN OHJELMISTON KEHITYS JA EVALUOINTI JOUKKUEURHEILIJOIDEN MONITOROINTIIN
Diplomityö
Tampereen yliopisto
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
Toukokuu 2019

Ihmiskunta on aina ollut intohimoinen eri urheilulajeja kohtaan. Ensimmäiset urheilulajit periytyvät tuhansien vuosien takaa. Pitkästä urheilun historiasta huolimatta, viime vuosikymmenten teknologinen kehitys on muuttamassa urheilun luonnetta sellaisena kuin se on tunnettu. Parempi teknologia mahdollistaa nyt urheilusuoritusten mittaamisen ja analysoinnin tarkemmin ja syvällisemmin kuin koskaan ennen. Nämä työkalut ovat johtaneet kurinalaisiin treeniohjelmiin, auttaen huippu-urheilijoita ylittämään itsensä. Kuitenkin, erityisesti joukkueurheilussa on vaikea arvioida mitkä harjoitusohjelmat ja kuormitusmäärät ovat sopivia. Jokainen urheilija ja joukkue on erilainen, kuten ovat myös harjoitustilanteet ja pelit, missä kuormitus on suurta. Siksi paremmille ja monipuolisemmille joukkueurheilijoiden seurantatyökaluille on yhä enemmän tarvetta.

Tässä diplomityössä tavoitteena oli kehittää ja arvioida seuraavan sukupolven huippu-urheilijoiden joukkuelajeihin soveltuvaa puettavaa sykemonitorointiratkaisua syke- ja urheiluanalytiikkayritys Firstbeat Technologies Oy:lle. Työ koostuu olemassa olevan Suunto Movesense sensorijärjestelmän arvioinnista ja sen päälle kehitettävästä ohjelmistosta joukkueurheilukäyttöön. Kehitetty sensoriohjelmisto integroi olemassa olevan fysiologiseen seurantaan tarkoitetun algoritmikirjaston Movesense sensoriin, mahdollistaen sensorilla prosessoitavan langattoman urheilijoiden reaaliaikaseurannan. Kunkin urheilijan reaaliaikainen analytiikkatieto lähetetään BLE mainostuksen avulla vastaanottavalle laitteelle, joka on vastuussa tiedon visualisoinnista valmentajille. ECG:stä mitatut R-R-intervallit tallennetaan myös sensorin sisäiseen muistiin. Täten kaikki mittadata on saatavilla sekä treenin aikana reaaliajassa, että treenin jälkeen muistista ladattaessa. Sensorin hallinnointia ja datan latausta varten kehitettiin Movesensen alustan päälle kommunikaatiorajapinnat.

Osana diplomityötä myös Movesensen ECG:n R-piikin tunnistusalgoritmia arvioitiin ja analysoitiin, verraten sitä Firstbeatin Bodyguard 2 monitorointilaitteeseen kenttätesteissä. Työssä arvioidaan myös Movesense-sensorin ja lopullisen ohjelmiston soveltuvuutta määriteltyihin käyttötapauksiin.

Sensoriohjelmiston kehitystyö toteutettiin iteratiiviseen tapaan, arvioiden sensorin suorituskykyä pohjautuen palautteeseen jatkuvasti käynnissä olevista urheilujoukkueiden kenttätesteistä. Kenttätestien raporttien tuloksia on käsitelty ja analysoitu osana ohjelmistokehityksen prosessin ja sensorin suorituskyvyn arviointia. Muistin- ja virrankäyttö on raportoitu. Osana diplomityötä kehitettiin myös yksinkertainen R-R-intervallien signaalin generointiin soveltuva testaus- ja kehitystyökalu. Jatkuvan integraation ja testauksen kehittämisestä keskustellaan lyhyesti.

Tämän diplomityön tuloksena on tuotettu markkinoille valmis ammattiurheilijoiden joukkueille suunnattu kuluttajatuote, Firstbeat Sports sensori, joka kohtaa sille asetetut tarpeet ja vaatimukset. Raportoidut tulokset näyttävät sensorin toimivan luotettavasti haastavissa urheilujoukkueiden käyttötapauksissa. Työn lopussa käsitellään kehitystyön haasteita ja tulevaisuuden jatkokehitystä.

Avainsanat: sykemonitorointi, sensorijärjestelmä, puettava langaton teknologia, BLE, reaaliaikamonitorointi, ammattiurheilijoiden joukkuelajien seurantajärjestelmä

PREFACE

This Master's Thesis was conducted whilst working for Firstbeat Technologies Oy. During the year of making this new team sports monitoring product to happen, there were many things which were learnt and experienced.

I would like to give my thanks to all those who've been part of the project. Especially I'd like to thank Antti Portaankorva for running all the field tests and for being a great college, as well as my examiners Adjunct Professor Ilkka Korhonen and Assistant Professor Antti Vehkaoja for the work they did with the project.

The greatest of my appreciation and gratitude is and will always be for my wonderful wife for everything she has done. Without her this could not have been done.

Tampere, 18.5.2019

Timo Onnia

TABLE OF CONTENTS

ABBREVIATIONS AND NOTATIONS

AD Type	Advertising Data Type
AES	Advanced Encryption Standard
AFE	Analogue front end
ANS	Autonomic nervous system
AoA	Angle of Arrival
AoD	Angle of Departure
API	Application programming interface
ATT	Attribute Protocol
AV node	Atrioventricular node
BG2	Bodyguard 2
BLE	Bluetooth Low Energy
Bpm	Beats per minute
CPU	Central processing unit
CRC	Cyclic redundancy check
ECG	Electrocardiogram
EEPROM	Electrically erasable read-only memory
EPOC	Excess Post-Exercise Oxygen Consumption
ETE-library	A proprietary Embedded Training Effect library by Firstbeat
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GFSK	Gaussian Frequency Shift Keying
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HCI	Host Controller Interface
HR	Heart rate
HRV	Heart rate variability
HTTP	Hypertext Transfer Protocol
IBI	Inter-beat-interval
IC	Integrated chip
IMU	Inertial sensor unit
IoT	Internet of things
ISM	Industrial, Scientific and Medical
JSON	JavaScript Object Notation
L2CAP	Logical Link Control and Adaptation Protocol
LL	Link Layer
MBR	Master Boor Record
MCU	Microcontroller unit
MVA	Moving average
NNI	Normal-to-normal interval
OTA	Over-the-air
pNN50	Percentage of pairs of adjacent NNIs differing by more than 50 ms
PPG	Photoplethysmography
PSD	Power spectral density
QRT	Quick Recovery Test
RAM	Random Access Memory
REST	Representational State Transfer
RMSSD	root mean-square of successive differences
ROM	Read only memory
RPE	Rating of perceived exertion
RRI	R-peak to R-peak -interval
RSSI	Received Signal Strength Indicator

RTOS	Real-time operating system
RTT	Real time transfer
SA node	Sinoatrial node
SDNN	Standard deviation of all NNIs
SIG	(Bluetooth) Special Interest Group
SM	Security Manager
SNR	Signal-to-noise-ratio
SoC	System-on-Chip
TCP	Transmission Control Protocol
TL	Training load
TRIMP	Training IMPulse
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UUID	Universally unique identifier
VO2max	Maximum rate of oxygen consumption
YAML	YAML Ain't Markup Languag

1. INTRODUCTION

1.1 Motivation

Throughout the history of mankind, sports of all sorts have always been a big part of societies. Team sports are a highly competitive field with lots of research, fans, support and money behind them. All the recent developments and research in technology and analysis tools have also led to new innovations and novel ideas on how to increase the performance of individuals and teams. There has been development in many different fields of science which have led to development of new type of smart wearable sensors and tracking devices, to collect and analyse data, both in real-time and offline. Some of these enabling technologies include wearable textile electronics, localization technologies, measurement technologies, algorithm development, machine learning, and better understanding of the human physiology. Overall the trend of having much more electronics and technology being packed within smaller and smaller components, combining complex radio chips with traditional microcontroller units (MCUs) into complex System-on-Chips (SoCs) running at extremely low power, have led to being able to measure and process lots of data within a tiny device embedded into clothing, or any wearables that we already have.

This development of new and better technologies was the start point to this thesis as well, as there was a need for a more powerful and versatile wearable heart rate (HR) sensor product to be designed for team sports monitoring. The new product would need to introduce new advanced features, such as real-time on-chip processed algorithm results of a player’s exercise, accelerometer data analysis combined with advanced HR and heart rate variability (HRV) analysis, internal memory for datalogging, and easy programmable interface for future development. As part of this thesis, the requirements were taken, technical product specification were made, and a market-ready product Firstbeat Sports sensor was produced using a Movesense platform by Suunto. This project was done whilst working for Firstbeat Technologies Oy, a company which develops, produces, sells and licences advanced HR and HRV analytics products for customers and 3rd party companies. Overall the objectives of the thesis were to:

- Develop an application that would meet the requirements (e.g. power consumption and resource usage as well as different use cases) for team sports monitoring sensor, using Movesense

- Evaluate and analyse the suitability of Movesense sensor and developed system for team sports monitoring, using field test reports
- Evaluate Movesense system as a suitable platform for team sports monitoring
- Explore ways to perform offline-testing and debugging more effectively
- Write a brief introduction and summary about Movesense development for someone without prior experience with it, as part of this thesis

In this thesis sensor software development and evaluation was done to a commercial reprogrammable smart sensor Movesense. The main work of the thesis comprises of designing the specification of the application programming interfaces (APIs) between the sensor and a receiver device, software development, continuous evaluation and testing of the product. Work was done in collaboration with colleges responsible for receiver device software development, testing and product management.

Software development of the product was done as an iterative work, continuously developing and evaluating the performance of the product. Both manual and hardware-in-loop testing was done for the software. A simple R-peak signal generator was devised for the purpose of hardware-in-loop simulation. Some unit testing and integration testing in a PC simulated environment were also explored.

Final evaluations of the product include summarised and analysed reports from field tests, and other quality metrics reported.

1.2 Structure of the thesis

This thesis has been divided into six chapters, beginning with this *Introduction*. Following the introduction, a *Background* chapter about cardiovascular system and team sports monitoring will be provided. In this chapter, Firstbeat's current team sports monitoring products will be briefly discussed together with some other companies in the field.

Firstbeat Sports sensor development using Movesense technology will be discussed in Chapter 3, Firstbeat Sports sensor development. The chapter will begin by presenting Movesense related technologies, such as Bluetooth Low Energy (BLE), FreeRTOS, Representational State Transfer (REST) and Movesense sensor's main SoC Nrf52832. Following these introductions, the Movesense platform will be discussed. Further specifications and requirements for the new team sports monitoring HR sensor application will be presented in section 3.12. Some details about software development, such as state logic, its validation, and software testing will be discussed here as well.

Chapter 4, *System evaluation*, will present the results of the thesis. Here the results from field tests will be used to evaluate and analyse the suitability of the developed sensor application and Movesense platform for team sports monitoring. A report from controlled field test with a reference device for recording RRI-signal will also be provided and analysed. Power consumption and memory usage of the final application are reported.

Finally, Chapters 5 and 6 present discussion and conclusions of the thesis project. Here the overall project flow will be analysed and pondered, and some current state of the product is considered. Future improvements are also discussed. Conclusions summarise the thesis and its results.

2. BACKGROUND

2.1 Cardiovascular system

Cardiovascular system is one of the most central parts of human physiology. It is a life sustaining part of the body which takes care of the circulation of oxygen and nutrients to the cells, as well as carrying away metabolic waste (Springhouse, 2002, p. 84). The continuously pumping heart can never fail in order to fulfil its mission to provide this flow of blood to the vessels, supporting rest of the systems and organs of the body. Therefore, the contraction of the heart muscle itself is an autonomous activity, making sure that no matter if the nervous system would fail, the heart keeps pumping. The rate of the contraction though, can be and is adjusted by the autonomous nervous system (ANS). This need of adjustment comes e.g. from need to adjust the blood flow to match the needs of an external physical load of the body, to increase or decrease the transportation of oxygen and nutrients to muscles. In order to sustain a long-lasting physical exercise such as marathon running, control of the blood flow is critical to provide an effective oxygen transportation. In many sports, rapid changes in external load are also present, resulting in fast changes of cardiac output from 5 litres per minute to 25 litres per minute, which is mainly achieved by increasing the heart rate (HR). (Springhouse, 2002, p. 94)

HR is one of the most important and common single physiological metrics which can be easily measured to estimate a person's physiological response to external load when doing sports (Springhouse, 2002, p. 94). HR is generally expressed as bpm (beats per minute), indicating the number of times a heart pumps per minute. HR calculation is updated per each new pump, and these consecutive HR calculations can be called a HR -signal. This signal can be easily visualized, analysed and used to derive many important metrics out of it, making it a popular tool to assess one's physiological condition e.g. in many clinical or sports use cases.

Individual's HR signal's range is constrained within some range. As maximal HR of person is limited, and largely determined by one's gender and age, it is not a factor used to define athlete's fitness level. Rather with more professional athletes', their hearts are more effective, resulting more cardiac output with same HR as with non-professionals. Also, their heart is usually larger. A more effective pumping of blood with one contraction results into much lower HR in rest. Therefore, it is not uncommon to have HRs below 40 bpm (beats per minute) with professional athletes doing endurance sports. Resting HR can be also used as a meter to determine someone's level of fitness. (Springhouse, 2002, p. 94)

Beat-to-beat -interval, or heart rate variability (HRV) is another important measure used to analyse physiological responses. It is calculated as a time distance between two consecutive pumps of heart. As with HR signal, these consecutive values form an HRV signal which can also be visualized and analysed for more advanced understanding of heart's and body's functioning. It is especially useful when analysing how body responses in relation to different internal or external stimulus, such as stress, exercise, relaxation, sleep, social events or eating. These stimuli affect our nervous system to react in different ways, which will increase or decrease the beat-to-beat interval of heart. Humans' nervous system consists of two main parts: the central nervous system and peripheral nervous system. Central nervous system consists of brain and spinal cord. The PNS further consists of three parts: somatic, enteric and automatic nervous systems (ANS). Of these systems, ANS is the responsible for adjusting the HR of the body. (Springhouse, 2002, p. 86) It is also largely responsible for controlling and regulating other functions of internal organs, affecting e.g. digestion, pupillary response and respiration rate. ANS is yet subdivided in to two parts: sympathetic and parasympathetic nervous systems. Sympathetic nervous system is activated when there is need to stimulate the body's *fight-or-flight response*, which e.g. increases HR. In the case of exercise this system ensures the sufficient delivery of oxygen to the muscles, by upkeeping the HR. Parasympathetic nervous system is largely the opposite, being active mainly when the body is in a relaxed state. Therefore, it slows the HR down. Together sympathetic and parasympathetic nervous systems help maintaining the homeostasis in the body. Prolonged imbalances in life can however disturb the natural balance of the body. Disturbances in homeostasis can be caused by for example continued stress, bad healthy habits, lack/excess of exercise, or lack of sleep (Kim, H. G., Cheon, E. J., Bai, D. S., Lee, Y. H., & Koo, B. H., 2018).

Over the past two decades, HRV has been studied extensively as a signal to provide insight into body's internal balance within the ANS. Nowadays HRV is being used in many clinical and consumer applications as a basis signal for analysing and detecting humans' stress reactions and states of relaxation. Sleep analysis can also benefit from HRV, although it is only part of it. Several scientific and statistical methods have been developed and published to derive analyses from HRV signal. The most common methods of analysis can be divided into time and frequency domain methods. Some of the common time domain methods include root mean-square of successive differences (rMSSD) of adjacent NNIs (normal-to-normal intervals), standard deviation of all NNIs

(SDNN), and percentage of pairs of adjacent NNIs differing by more than 50 ms (pNN50). In frequency domain analysis, power spectral density (PSD) analysis is commonly used to analyse what frequency components are prevalent in the HRV signal, and how they change over time. Frequency domain analysis usually divides the interpretation of results to three separate frequency channels, of which each are associated with different physiological phenomenon. These channels are high frequency (HF; 0.15–0.4 Hz) or vagal component, low frequency (LF; 0.04–0.15 Hz) or sympathetic component, and very low frequency (VLF; <0.003–0.04 Hz). Further, each of these frequencies are associated with different body functions such as respiratory sinus arrhythmia (HF), sympathetic nervous system's regulatory mechanisms e.g. blood pressure (LF) and temperature regulations (VLF) (Malik, 1996) (Shaffer & Ginsberg, 2017). Recent studies have also proposed new approaches to understanding the significance of HRV in relating to other human states as well, such as the psychological state of the person. The field is therefore going onwards and is under continuous development. (Ernst, 2017) HRV:

As HR signal and HRV signal are closely related and complementary to each other, it is necessary to understand how they correlate. With higher HR the baseline of the HRV is lower, as the time between heart beats is shorter, and vice versa for lower HRs. However, this does not make them alternative measures, but rather HR signal is just an averaged visualisation of the HRV signal. Therefore, HRV signal carries more information within it. (HeartMath Institute, 2019)

Beat-to-beat signal can be measured in number of techniques such as ultrasonography, doppler, photoplethysmography (PPG) and electrocardiography. Each of these come from their own application domains and can give different kind of information about heart. As for measuring the plain HR, electrocardiography and PPG are the most commonly used, due to their easiness of access. In Figure 1 we can see signal measured with an electrocardiography, which is called an electrocardiogram (ECG). For comparison, PPG signal is also included.

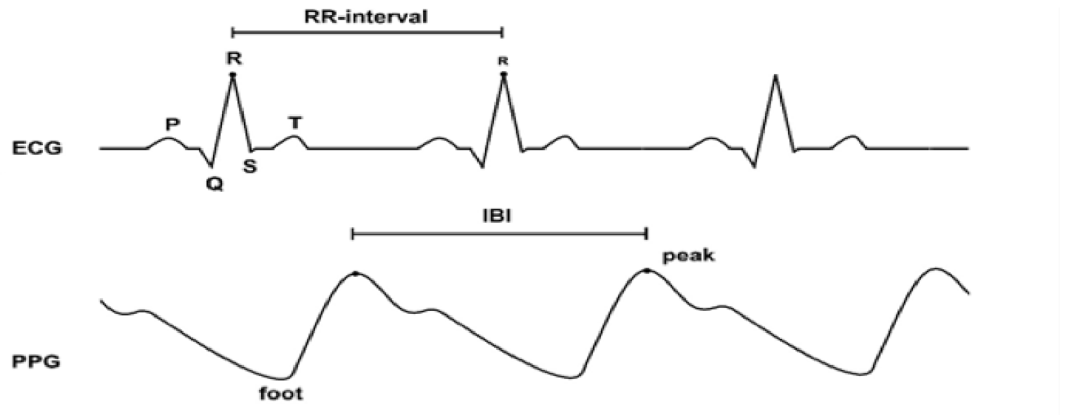


***Figure 1** Examples of how ECG and PPG signal look like. An inter-beat-interval (IBI) and an RR-interval are marked. (Fergus, 2017)*

Electrocardiography is used to measure the electrical activity of the heart, whereas photoplethysmography indirectly measures the heart's rhythm from blood pulse. In the last decade PPG, or optical heart-rate monitors have become extensively popular in not only consumer devices, but also medical grade devices, due to advanced sensor electronics and signal correction algorithms. One of PPGs benefits is easiness of access as it can be integrated e.g. into smart watches. It works by emitting short pulses of light into the skin and at the same time measuring the intensity of light reflected. The resultant signal is the reflected light intensity modulated by the blood flow under the skin. Accuracy of PPG can vary a lot, however, depending on factors such as brightness of the sunlight, arterial stiffness, skin colour and movement of skin in relation to the sensor. Electrocardiography does not have these restrictions, and as a more robust and accurate method, it is most often the choice of technique to measure accurate beat-to-beat signal. (Scheid, J. L. & O'Donnell, E., 2019)

To further understand where ECG originates from, we need to understand heart, and how it functions. Figure 2 shows an open-cut image of heart with its main components, focusing on heart's conductive system. Walls of the heart compose of cardiac muscle, called *myocardium*. It can be divided to left and right side. The left side receives oxygenated blood from the lungs via pulmonary veins and pumps it to the body's blood circulation via aorta. The right side receives the unoxygenated blood and pumps it to lungs for oxygenation via pulmonary artery. Each side consists of an atrium for receiving the blood, two valves for controlling the blood flow, and a ventricle where the blood pressure is generated to pump the blood onwards. Pumping of the blood is caused by the heart muscle contracting itself. On a cellular level, contraction of the muscle cells (*myocytes*) originates from the electrical activation of the cells. An electrical stimulus signal causes cells to exchange ions between surrounding tissue fluid, which disturbs the electrical equilibrium of the cells. This results into mechanical contraction of the myocytes, and propagation of this electrical stimulus to neighbouring cells in all directions making the muscle to contract. The electrical activation and propagation of the cells is called action potential. Action potential first begins with depolarization, where cell's electric potential changes from -70 mV to around 20 mV. After a plateau phase follows a repolarization, when the cell returns to its equilibrium state. Action impulse lasts for about 300 ms. (Jaakko Malmivuo, 1995, pp. 185-189)

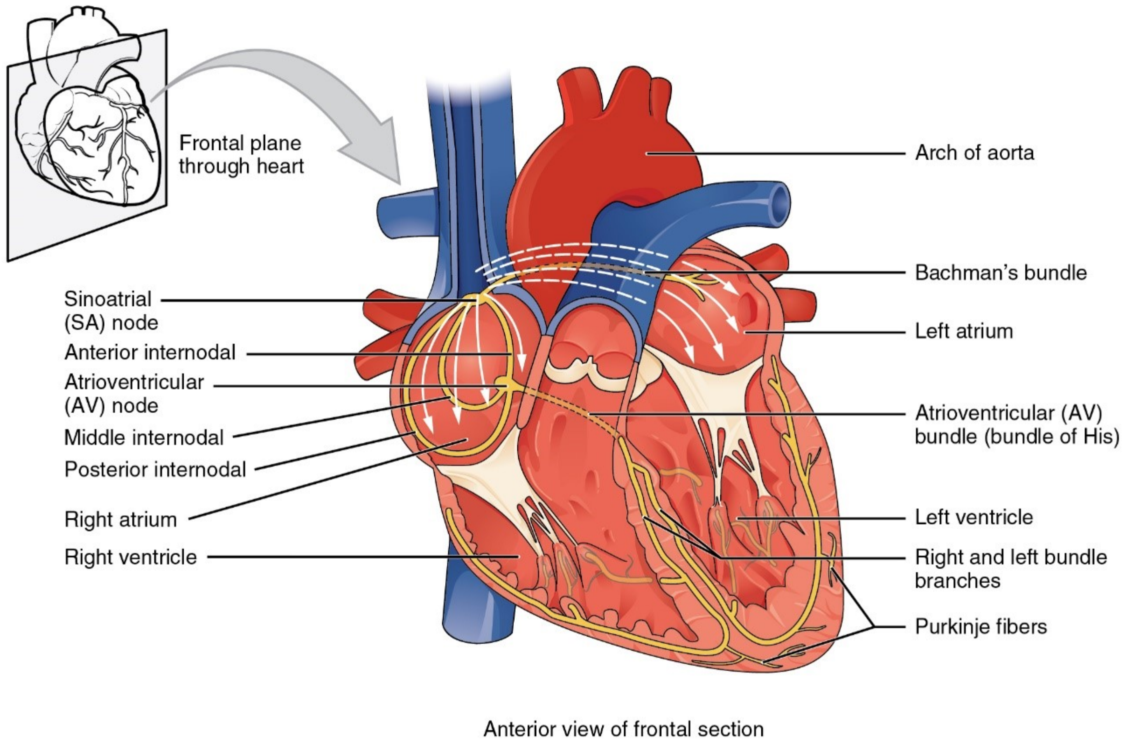


Figure 2 *Conduction system of heart (OpenStax College, 2013).*

Electric impulses which activate the heart muscle originate from Sinoatrial (SA) node – a part with specialized muscle cells, self-excitatory pacemaker cells. From SA node the electric impulse is conveyed first to both atriums, and then to Atrioventricular (AV) node. After a short delay the electric impulse then carries on into both ventricles via bundle of His and Purkinje fibres. The short delay allows time for the atria to contract and fill the ventricles with blood before it is time for the ventricles to contract. In the figure of the conduction system of heart we can see these conduction paths of the heart's electric impulses. (Jaakko Malmivuo, 1995, p. 188)

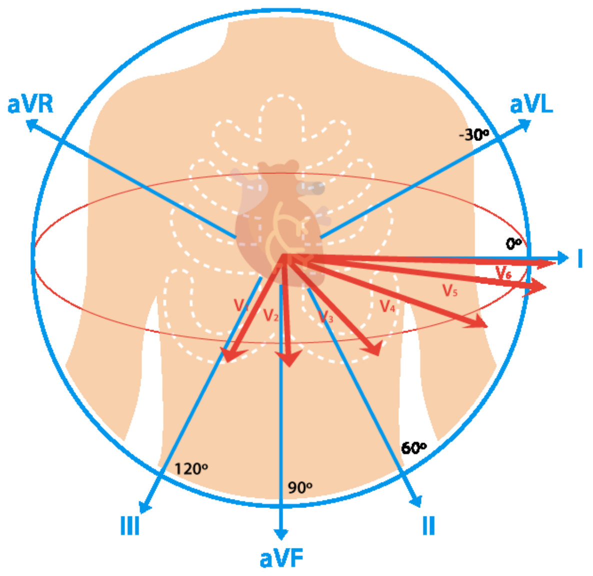


Figure 3 *12-lead ECG measurement system with limb leads, augmented leads and precordial leads (Cables and Sensors, 2019).*

From the clinical perspective, measuring the electrical activations and propagations of the heart cells can tell about possible pathological conditions, such as dead cells. Therefore, much research has been done to model the heart's electrical activity. These models usually benefit from the possibility to measure the changes of electrical potentials on the skin of the human torso noninvasively. Traditionally the heart is modelled as a volume source conductor consisting of multiple dipole sources. The sum of these dipole sources can be measured as projections of heart's electrical activity against used measurement locations (leads). The most commonly used measurement location arrangement is called 12-lead ECG, presented in Figure 3. It consists of three limb leads, three augmented limb leads and six precordial leads on the chest. The roots of 12-lead system originate from Willem Einthoven (1860-1927) about a hundred years ago. He established an Einthoven's triangle, or three limb leads. These are the only bipolar leads in the system, meaning that the leads are measured directly from one electrode location to another. Other leads are unipolar, having a reference point being averaged by other leads. Augmented leads use pairs of two limb leads as a reference for the third electrode. Precordial leads' reference is so called Wilson's terminal, which is an averaged potential of the three limb leads. All these twelve different leads or their projections can visualise the heart's electric activity from different directions, also providing different information from different parts of the heart. Figure 3 shows the directions which the 12-lead system tries to capture (the actual polarity of the six limb leads is not visible). (Jaakko Malmivuo, 1995, pp. 50, 387-389)

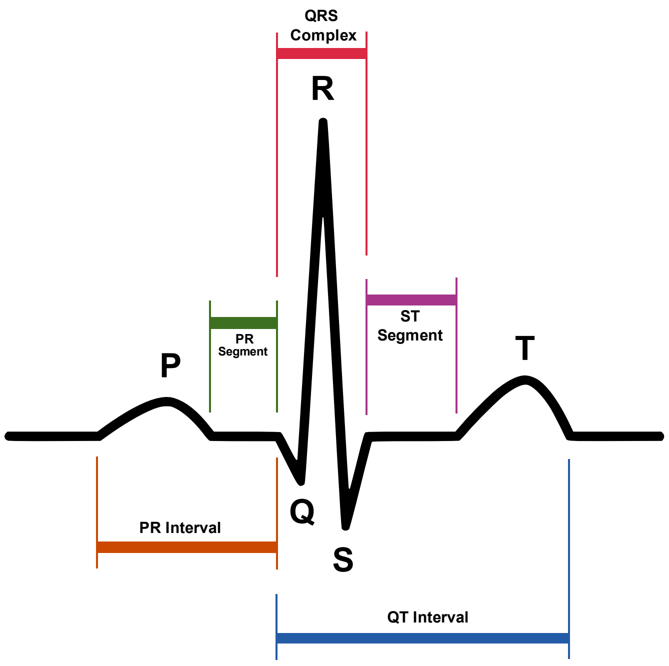


***Figure 4** Normal ECG signal with P, Q, R, S and T parts marked (Atkielski, 2007).*

In this thesis the application of electrocardiography is a HR or beat-to-beat monitoring system for team sports usage. Therefore, only the RRI signal is needed, and thus only one lead is enough to measure the ECG. Figure 4 shows an example of a typical ECG from one lead, consisting of five distinguish parts: P, Q, R, S and T. These all originate from different polarization phases of the heart, but for measuring the beat-to-beat interval, the so called QRS complex is the most useful. Several mathematical methods, such as Pan-Tompkins algorithm have been developed to automatically and accurately measure the high peak of the R wave. These consecutive measurements form HRV signal as was illustrated earlier in Figure 1. Further, as we are only interested about the R-peaks, the overall quality of the other waves is less important. Most HR sensors in the markets measure the signal with a HR strap around one’s torso. It is one of the most convenient ways to measure accurate HR. The following sections will lead more towards the application of the HR monitoring in team sports, and realization of the new monitoring system presented in this thesis.

2.2 Physiological analysis tools for team sports

As already stated, professional team sports are a highly competitive field of human society. It is part of most of our lives in one way or another. Even if we don’t participate or actively follow any sports, we are still subjected to sports news, new world records, or competition scores through others or different media. However, much less is often spoken about the work that led to the successes or the hours of training behind these achievements. This work consists of mentoring, exercise, recovery, sleep, and other matters both physical and mental. Other people such as coach, family relations and other team members are all involved in bringing out the best performance of people. All these different aspects of individuals in teams make up a complex pack of different things affecting the whole team performance. Not to even mention about other matters such as game strategies. Competitiveness of sports and technological advances are continuously driving the level of coaching to bring the best out of teams. This is also the purpose of the monitoring system devised in this thesis, focusing on HR monitoring during the exercises or competitions. In this section a few common keywords, topics and measures are presented, which further expound on the topic of different possibilities of training analytics, related to sports monitoring.

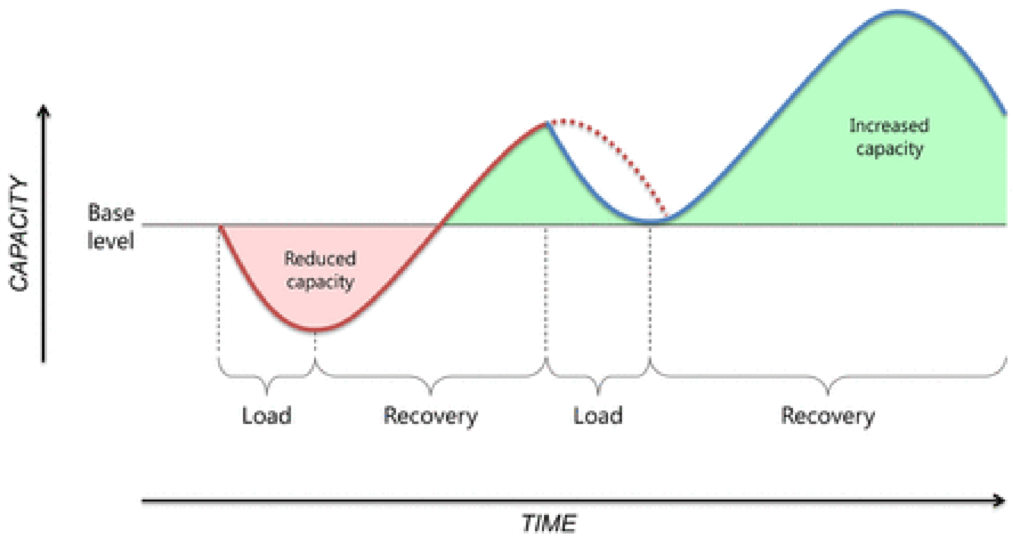


Figure 5 *Physiological adaptation through cycles of adding training load and recovery (Soligard T., Schwellnus M, Alonso J., et. al., 2016).*

2.2.1 Training theory, training load and recovery

In the competitive field of professional sports, athletes and their support staff are forever striving to reach higher and higher in performance – to close the small margins which keep them from rising to the next level. Many things may influence the individuals personal progress, of which their training routine is of the greatest importance. Many different training theories exists, but the paramount of them all is to increase the fitness and performance of a person by the process of biological adaptation. In practise this means increasing the athletes' training volume and intensity to their limits. This will be followed by the body adapting to the level of exercise, maximising performance improvement. The adaptation does not come overnight though, as the body needs time to recover after a strenuous activity. The effect of training, training load (TL) and recovery has been visualised in Figure 5. Poorly managed TL together with no time to recover may increase the risk of injury and prolonged fatigue and will lower the performance. Appropriate physiological and psychological homeostasis needs to be reached by finding the right balance. Consequentially, there is a need to be able to scientifically or quantitatively measure the amount of TL and recovery. (Soligard T., Schwellnus M, Alonso J., et. al., 2016)

At this current date, there are no golden standards or references as to clearly state or measure the precise TL of a person, due to complexity of the subject. When trying to estimate a TL, measurements regarding to it are often separated to two categories: internal and external TL. External TL defines only the external stimulus which is pressed on the athlete without considering athlete's internal characteristics. Internal TL then describes the individual's physiological and psychological response to that external load. External loads measured can be e.g. performance time, training frequency, power, speed, acceleration, number of repeats, or distance ran. In addition, e.g. life-events and daily travel could be an external load as well. Measurements related to internal TL can be for example HR, blood lactate concentration, oxygen uptake, rating of perceived exertion (RPE), sleep quality, biochemical assessments, TRIMP (Training IMPulse), HRV and different questionnaires (Soligard T., Schwellnus M, Alonso J., et. al., 2016). Different measures emphasize different aspect of TL. Therefore, none of these parameters alone can give a total view of a person's true internal TL, as it is a sum of many factors (Kaikkonen, P., Hynynen, E., Mann, T. et al., 2010) (Halson, S.L, 2014)

2.2.2 Some notable measurements or metrics

TRIMP (Training IMPulse) is a measure which aims to include both training intensity and duration into the calculation (Banister E, 1991). It is calculated from the training's duration, maximal, resting and average HR, from the time of training. TRIMP accumulation rate per minute or TRIMP/min is a windowed version of TRIMP which can be calculated in real-time during training. This can be helpful for coach in assessing the immediate responses to on-going training (Firstbeat Technologies Ltd., 2019). Further modifications of TRIMP, such as Edward's TRIMP or individualized TRIMP have been introduced. However, the original TRIMP is generally used and considered as a useful marker for estimating internal TL. (Halson, S.L, 2014)

Lactate Concentration of blood is a physiological blood test biomarker sensitive to training intensity and duration. It is especially used in evaluating endurance performance

(Halson, S.L, 2014). Lactate is a by-product from energy usage in the body. During intensive exercise, lactate concentration of blood rises. When the concentration reaches a *lactate threshold,* fatigue level of an athlete will rapidly rise. Therefore, the lactate threshold level is also used as a marker for endurance fitness. (Jules A. A. C. H., P. G., Stuurman F. E., Wouter A. S. de,Muinck Keizer, Yuri M. M., Cohen A. F., 2018)

HRV or heart rate variability has been studied in many publications to be related to athlete's internal load, stress or recovery. Firstbeat Technologies Ltd. provides HRV based recovery analysis as part of their analytics libraries (Firstbeat Technologies Ltd., 2015).

EPOC (Excess Post-Exercise Oxygen Consumption) is a measure which describes the level of disturbance of the body's homeostasis, as caused by an exercise. It is related to the level of recovery required. The so called "oxygen debt" is caused by elevated metabolic rate and replenishment of body's resources after the training. (Gaesser, G. & Brooks, G., 1984) Generally, EPOC has been measured analysing respiratory gases in a laboratory. However, scientific studies have shown that indirect predictions of EPOC can also be calculated using HR measurements. (Firstbeat Technologies Ltd., 2012)

HR Zones, by definition, are a set of HR ranges given as percentage ranges of an individual's maximal HR. These zones can be used as a tool e.g. to focus training on either anaerobic or aerobic workouts. HR is quite often divided into five zones plus HR reserve, though the exact percentage values do vary. (Scheid, J. L. & O'Donnell, E., 2019)

Resting HR or heart rate during resting is most often correlated with an individual's fitness level. Elite athletes can have an extremely low HR of even 40 bpm. (Springhouse, 2002, p. 94)

VO_2max or maximal oxygen consumption means the individual's maximal amount of oxygen, which he can consume typically during one minute of intensive physical activity. It is generally considered as a golden standard for one's level of aerobic fitness. (Scribbans, T. D., Vecsey, S., Hankinson, P. B., Foster, W. S., & Gurd, B. J., 2016) VO_2max is measured by measuring gas exchanges during an exercise in a laboratory. Yet, accurate estimations of it can be made during running with 95% accuracy, using HR and speed data. (Firstbeat Technologies, 2014)

2.2.3 Team sports monitoring

Wearable sports tracking and HR monitoring have been a major trend in the last decade. After optical HR sensors' rapid development, multiple different sports devices have come to the consumer markets. However, the team sports market is slightly different, as it is much less crowded than rest of the wearable markets. Professional teams are a high demanding customer, with growing needs to reach the higher level of performance. Also, professional sports teams rely often on scientifically researched studies and analytics. In the teams sports the goal is to measure the external and internal load, estimate the recovery times, as was discussed in previous chapter. In addition, last few years have seen an increasing trend in the use of localization services such as GPS or BLE to track the players on the field. (Roell, et al., 2018) This information can be used for strategic planning, but also to follow different metrics of players, such as speed, movement and sprints. Localization metrics and other external measures themselves cannot, however, tell too much about the internal TL of a player, and it is therefore a complementary to tracking teams' wellbeing. Therefore, HR monitoring is still essential to team monitoring. However, due to technological improvements and high competition, more is being required of the devices. Some companies have e.g. moved on from chest straps to sports underwear with electrodes embedded into the textile. Another thing is the need for coaches to both have internal memory on the sensors and have a possibility for real-time training data metrics from the athletes. Robustness of sensors is also an important matter, as tight schedules of teams don't allow for any addition hassle with devices.

Polar Electro, Catapult Sports and Firstbeat are some of the most notable players in the team sports markets. A brief introduction on each will now follow.

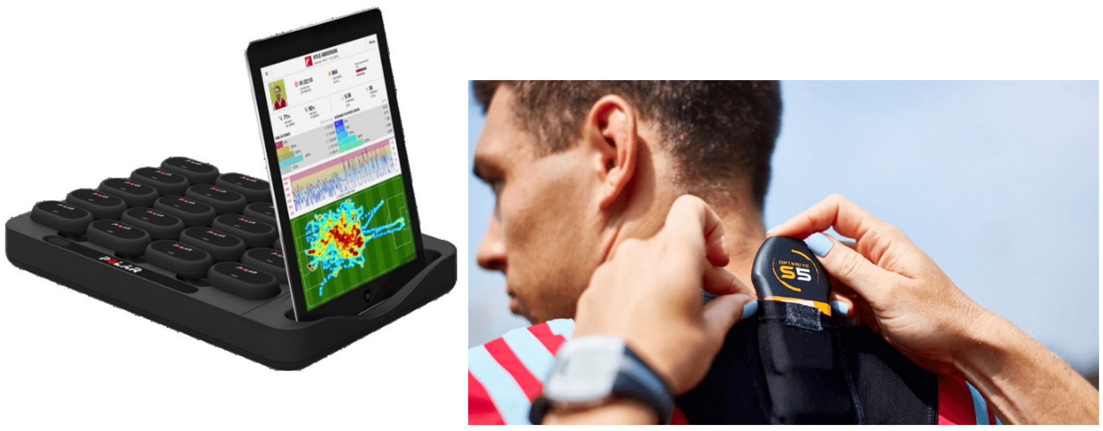


Figure 6 *GPS and HR tracking system Polar team pro (left) and Catapult Sports' GPS solution OptimEye S5 (right) (Polar Electro, 2019) (Catapult Sports, 2019).*

Polar Electro Oy is a Finnish manufacturer of wearable HR monitors and trackers, which was established in 1975. Apart from manufacturing sports HR watches, they offer a team sports tracking system as part of their portfolio. Their tracking system includes a sensor and a special shirt with a place for the sensor, in addition to coach's monitoring device iPad. The sensor is located at the upper back, below neck. The shirt also embeds electrodes for HR measuring. The system allows tracking positioning, speed, distance, accelerations, sprints, and running cadence, using GPS, HR and inertial sensor unit (IMU) sensor data. The system is capable of transmitting data over BLE. Figure 6 (left) presents their sensor charging dock with iPad. (Polar Electro, 2019)

Catapult Sports Ltd. is an Australian sports technology company, founded in 2006, which provides both indoor and outdoor localisation tracking solutions to team sports. Over 2500 teams use their solutions. Catapult's outdoor tracking solution uses GNSS (Global Navigation Satellite System) localisation tracking solutions to access both GPS and GLONASS satellites. Indoor tracking solution uses several locally installed sensors to receive localisation transmission data at a license-free ultra-wideband 5,2 GHz frequency. In addition to localisation, they also offer video monitoring and analysis software. As they also have IMU sensors, they provide similar metrics as Polar does about acceleration, jumps etc. They have also released information about their upcoming new device Catapult Vector. It has been advertised to have localisation capabilities for both indoor and outdoor tracking as well as a vest for HR monitoring with the same sensor. (Catapult Sports, 2019)

2.2.4 Firstbeat Sports

Firstbeat Technologies Oy. is a Finnish company in the field of professional sports and wellbeing analytics. Their main products include HR and HRV analytics found from their own, and licensed devices. Firstbeat originally started their business around professional sports already in 2002, and the company has been actively involved in research and development in this field ever since. Firstbeat covers a notable part of the team sports markets, with more than 1000 elite teams, 26 national teams, 25% of Champion League soccer teams and over 50% of NHL teams using their analytics and devices. Firstbeat's current team sports monitoring solutions include both memory belts for offline sports tracking as well as real-time transmitting HR belts. Additionally, the company's Bodyguard 2 HRV recording device is being offered for more extensive and recovery analytics, allowing for example tracking of sleep quality and stress reactions during daily activities. Bodyguard 2 and a picture of the team sports real-time monitoring system have been presented in Figure 7. (Firstbeat Technologies Oy, 2019)

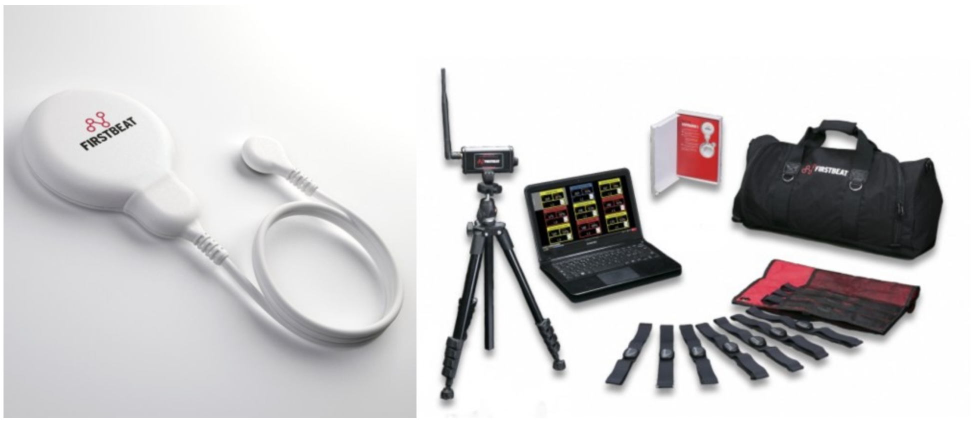


***Figure 7** An HRV measurement device for stress and recovery (left) and team sports real-time monitoring system (right) (Firstbeat Technologies Oy, 2019).*

Together with different device solutions, Firstbeat offers a software called Firstbeat Sports for managing the training data and analytics. Firstbeat Sports is a PC software running on a laptop. Mobile applications are also provided for individual athletes for accessing their own metrics.

In this thesis a new sensor solution has been developed for the growing markets, to meet the needs of the increasing customers. The need came from a lack of having both memory recording and real-time monitoring features within a same device. Also, it was desired to run the real-time analytics within the sensor itself. At the same time together with the sensor development, a new client device monitoring software has also been developed. The new software will be running on iPad, allowing more practical handling of the client device during training. The new sensor developed in this thesis, and the new upcoming monitoring software will work hand in hand to take team sports monitoring tools to a new level.

2.2.5 Embedded Training Effect (ETE) library

Embedded Training Effect library (ETE) is Firstbeat's proprietary physiological analysis library. Different versions of ETE have been licensed to several different companies in the wearables' markets. These companies include e.g. Samsung, Suunto, Garmin, Huawei and Huami. Most of the products where ETE is used are smart or sports watches, though other products exist as well. (Firstbeat Technologies Oy, 2019) In addition to licensing the ETE library, Firstbeat uses it within their own team sports monitoring solutions. Prior to this project ETE library has never been run on any of the company's team sports HR sensors themselves, but the analytics have been done on a client device.

ETE has several physiological features which it can analyse and calculate in real-time, using beat-to-beat HR data, and optionally other data sources such as accelerometer and cycling power data. Many of these features, such as EPOC and Quick Recovery Test (QRT) are based on proprietary algorithms, based on scientific studies and research. Further documentation and proof of their validity has been provided publicly in company's whitepapers, but they will not be discussed in this thesis. However, here is a list of only a few major features of ETE library (Firstbeat Technologies Oy, 2019):

- TRIMP and TRIMP per minute accumulation rate
- EPOC
- Anaerobic and aerobic training effect
- Energy expenditure
- Quick recovery test
- Maximal oxygen intake (VO_2max) fitness level
- Training Load and recovery time
- Lactate threshold
- All-day Stress & Recovery

3. FIRSTBEAT SPORTS SENSOR DEVELOPMENT

In this thesis a novel team sports monitoring HR sensor application *Firstbeat Sports* is developed using a commercial *Suunto Movesense technology*. Before discussing about Movesense sensor and platform, some background information is given on the topics of Bluetooth Low Energy (BLE), FreeRTOS and Representational State Transfer (REST). After these, the actual Movesense system and the Movesense sensor's MCU (microcontroller unit) NRF52832 will be presented. Finally, the development of the team sports application *Firstbeat Sports* will be discussed.

3.1 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) is a radio standard supported and used in many devices needing a wireless communication interface such as PCs, mobile devices, smart watches, sport tracking devices, IoT sensors, medical devices and many others. Since 2010, when BLE was first introduced by Bluetooth Special Interest Group (SIG), BLE sensors have become more and more ubiquitous being available in many devices. It has been adopted as a wireless protocol solution at much faster rate than many other wireless technologies. Part of the reason for this has been the rise and increase of mobile and tablet device markets, at the same time as the BLE was introduced. Also, at this time the big companies such as Apple and Samsung adopted the technology early-on to their products. Not to mention the fact that nowadays you can purchase a full-featured System-on-Chip (SoC) with both microcontroller unit (MCU) and radio for less than 2$ per chip in low volumes. This hasn't always been true for WiFi, GSM, Zigbee, etc. although now some manufacturers, such as Nordic Semiconductor also provide e.g. Zigbee and Thread protocol stacks together with BLE stacks for the same SoC. (Akiba, et al., 2014)

BLE standard is a subset of a wider wireless technology standard, which is just called Bluetooth. Bluetooth has been actively developed by SIG since late 90's and has a different focus than BLE. Originally BLE was designed by Nokia under a name Wibree. Its baseline of design was to be the lowest power consuming radio protocol, optimized for low cost, low bandwidth and low complexity. With this technology it is easily realistic to run sensors for extended time using just a coin battery, with months to years of power-on time. Original Bluetooth, often called Bluetooth Classic, fundamentally differs from BLE by being specified more on to a strict set of use cases. BLE instead was conceived to allow more open hands for the developers to fulfil their ideas, without having to know too much about the underlying technology. (Akiba, et al., 2014)

After its adoption to Bluetooth, BLE was also formerly marketed as Bluetooth Smart. Separation between Bluetooth Classic and BLE is needed because these protocols are not directly compatible with each other. Therefore, another branding name Bluetooth Smart Ready was introduced to signify that a Bluetooth device has capabilities to use both protocols, Bluetooth Classic and Bluetooth Smart, as it was called. These Bluetooth Smart Ready devices are also referred to as dual-mode devices, being able to act in two modes. Figure 8 presents the different Bluetooth versions and how they are configured. (Akiba, et al., 2014)

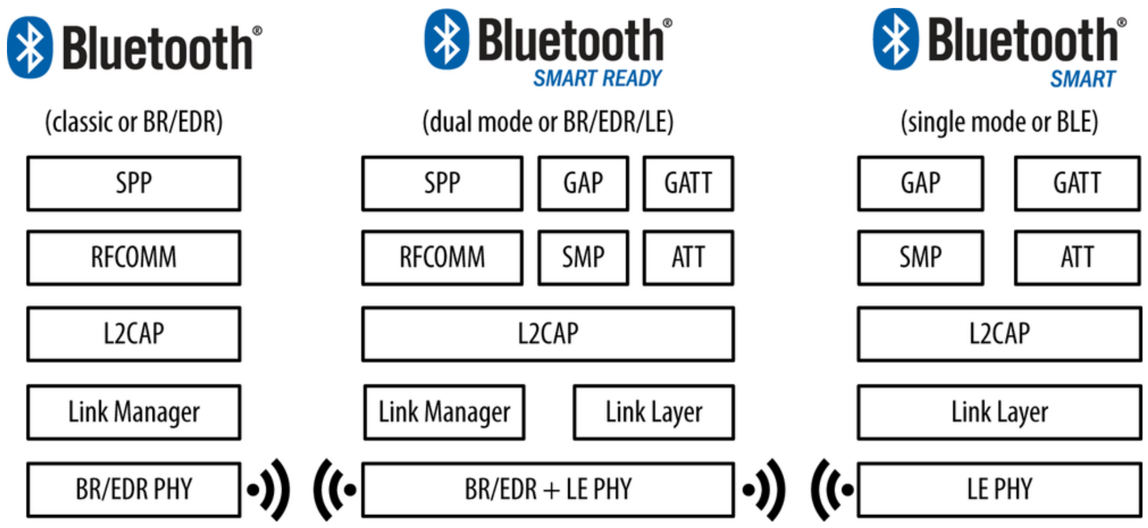


Figure 8 *Different Bluetooth versions and their configurations (Akiba, Davidson, Cufí, & Townsend, 2014).*

Bluetooth standard has incrementally upgraded to the state where it is today. These updates have not only added improvements and new features for new use cases, but also different options for data transfer speeds. Notable versions of data transfer speeds of Bluetooth Classic protocol include Bluetooth Basic Rate (BR) since version 1.0, Enhanced Data Rate (EDR) since version 2.0 and High Speed (HS) since version 3.0. The maximum data transmission speeds of these Bluetooth versions are consequentially 721kbit/s, 2,1Mbit/s and 24Mbit/s. It must be noted though, that the Bluetooth HS does not actually use the Bluetooth protocol itself for the transmission but negotiates the data transfer to be done over a collocated 802.11 link. Inclusion of BLE standard into the Bluetooth's standard was committed to version 4.0 in June 2010. Since then, new released versions of the Bluetooth standard have been 4.1, 4.2, 5 and 5.1. BLE 4.0's data transfer speed depends on the use case, and it can range between 125 kbit/s to 1Mbit/s (whereas the actual modulation rate of the radio is set by the specification at a constant

1MBps). Update of the standard to Bluetooth 5 added a possibility for the BLE to double the speed to 2Mbit/s. (Akiba, et al., 2014)

All these speed rates are however theoretical maximums, and when converted to real-life data transfer speeds, the result depends on multiple factors, such as environment, data traffic, limitations of the devices and protocol overheads. For example, in a normal use case with BLE, a typical best-case scenario can be estimated to be anything between 5-10kB per second. (Akiba, et al., 2014)

Although speed is an important aspect of a wireless protocol, it is important to remember that BLE is not designed for speed but for low power. These two often contradict, and therefore compromises must be made when designing applications for BLE. Maybe more important and interesting than speed, are some of the other new features added in the most recent upgrades of BLE. These upgrades include e.g. Bluetooth Mesh for creating large-scale device networks, Low Energy (LE) Long Range, LE Advertising Extensions enabling sending much more advertisement data, Higher Output Power up to 20dBm, and Angle of Arrival (AoA) and Angle of Departure (AoD) which are used for location services and tracking of devices. All these new upgrades bring some new use case possibilities for BLE, and manufacturers are constantly updating their protocol stacks to keep up-to-date with the latest available standards. (Akiba, et al., 2014)

Both Bluetooth Classic and Bluetooth Low Energy use the industrial, scientific and medical (ISM) radio bands, from 2.400 to 2.485 GHz. In BLE the frequencies are further divided into 40 2MHz-wide channels, whereas Bluetooth Classic uses 79 1MHz-wide channels. Presentation of these channels for BLE is included in Figure 9. For data transmission BLE uses Gaussian frequency shift modulation, which is also used in Bluetooth Basic Rate scheme. Data transfers are also done in short bursts instead of continuous data flow. To save energy, radio is simply switched off between bursts. The same frequencies of these ISM bands are also used by many other wireless standards, such as WiFi, Zigbee, ANT+, and Thread. As these different technologies have been designed for various use cases, no further comparison between them will be done. Also, as the Movesense sensor under investigation in the thesis, uses BLE 4.2, the scope of the following brief introduction will focus on this specific version of the Bluetooth standard. (Akiba, et al., 2014)

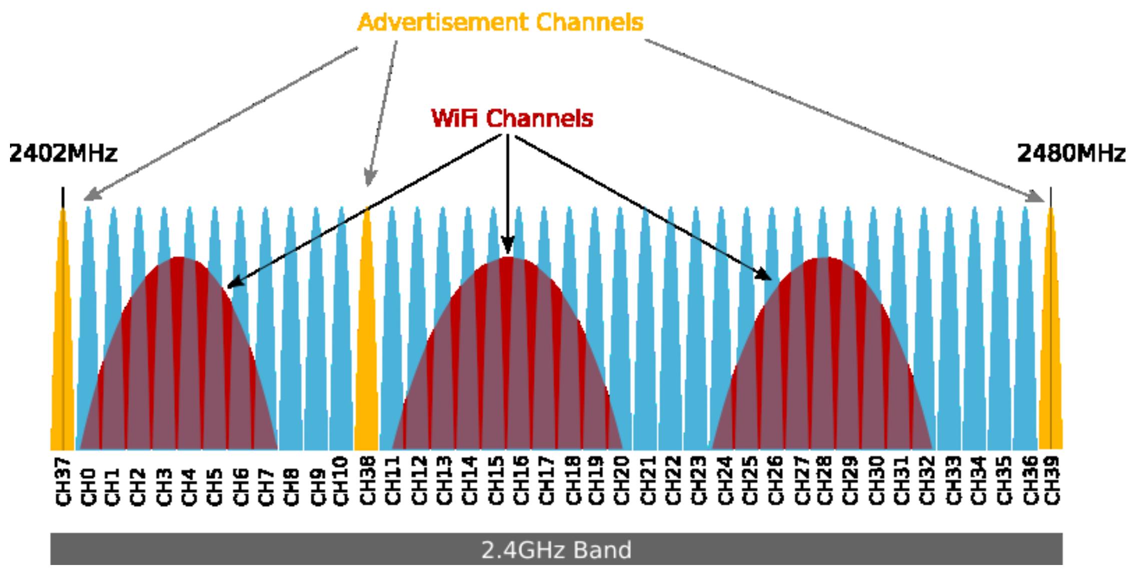


Figure 9 *BLE standard separates the ISM frequencies to 37 data transmission channels and 3 advertising channels (Argenox, 2019).*

BLE standard defines three distinctively separate parts, which constitute the Bluetooth protocol stack. These three main building blocks are Application, Host, and Controller. Each of these blocks have their individual roles, which allows them to be also manufactured as separate IC chips. Controller is the only critically real-time requiring block, including the radio and lower layers of the protocol stack. Therefore, the BLE standard specifies a Host Controller Interface (HCI) to allow interoperability between Hosts and Controllers produced by different companies. Separation of host and application blocks is also possible by designing own proprietary protocols. (Akiba, et al., 2014)

All the layers of Bluetooth Low Energy device have been presented in Figure 10.

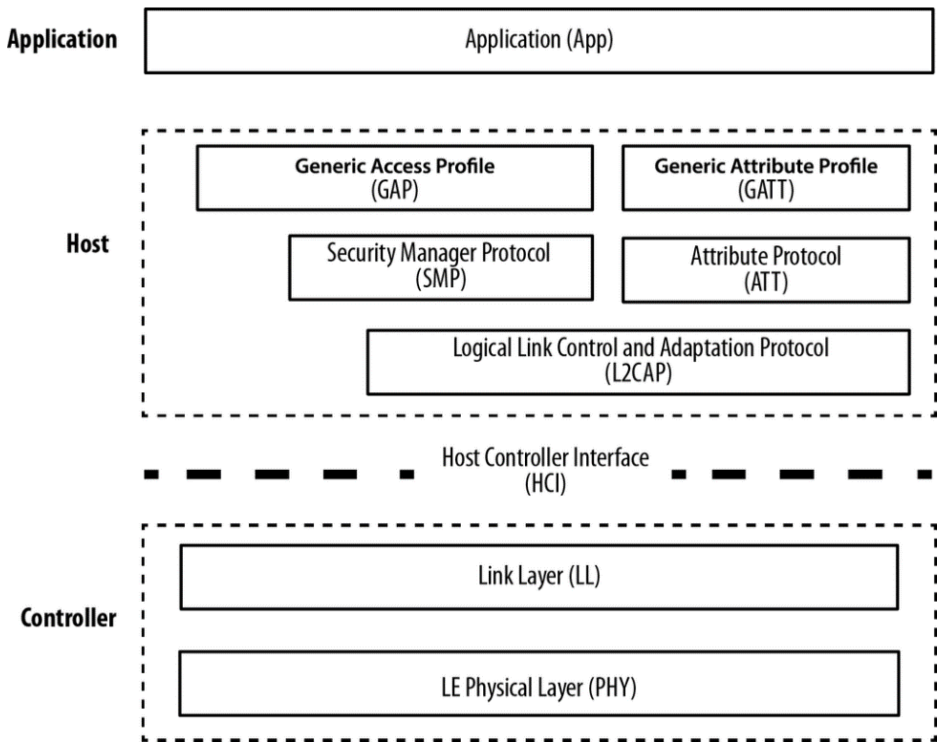

Application
Application (App)
Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager Protocol (SMP)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Controller
Link Layer (LL)
LE Physical Layer (PHY)

Figure 10 *BLE protocol stack with Application, Host and Controller layers (Akiba, et al., 2014).*

Now, a short overview of different parts of the stack is given, to explain and to understand different aspects of BLE which can be used and tweaked when designing one's own application working on top of BLE stack. (Akiba, et al., 2014)

Controller

The actual radio with the analogue communication circuitry is in the lowest layer of BLE stack. This is called LE Physical layer (PHY), and it does all the modulating and demodulating of analogue signals and interpreting the received signals back into digital symbols. As already mentioned, a constant 1Mbit/s modulation rate is used together with Gaussian Frequency Shift Keying (GFSK), using the free 2.4GHz ISM bands. During connection a frequency hopping spread spectrum technique is used to switch between used radio channels. Formula used for hopping is

$$\boldsymbol{nextChannel} = (\boldsymbol{currentChannel} + \boldsymbol{hop})\, \boldsymbol{mod}\, \mathbf{37}. \qquad (1)$$

The central device is responsible for selecting the value of hop when first establishing a connection with a peripheral device. Using this technique minimizes the amount of collisions which can interfere with the transmission, with other wireless protocols using the same ISM bands.

As can be seen from Figure 10, Link Layer (LL) is the part responsible for interfacing with both Host block, via HCI and Physical layer. This is the layer which does all the hard work taking care of the BLE protocol stack's timing. This layer usually combines custom hardware and software to meet its requirements. Some of LL's responsibilities are providing e.g. preamble, access address, air protocol framing, CRC generation and verification, data whitening, random number generation and AES encryption. Link layer is also responsible for assigning the roles related to connection: advertiser, scanner, master and slave. Bluetooth device address defined by the LL can be either a Public device address, which is a factory-programmed unique address, or a Random device address, which can be dynamically generated at run-time. It is also possible to assign both to the device.

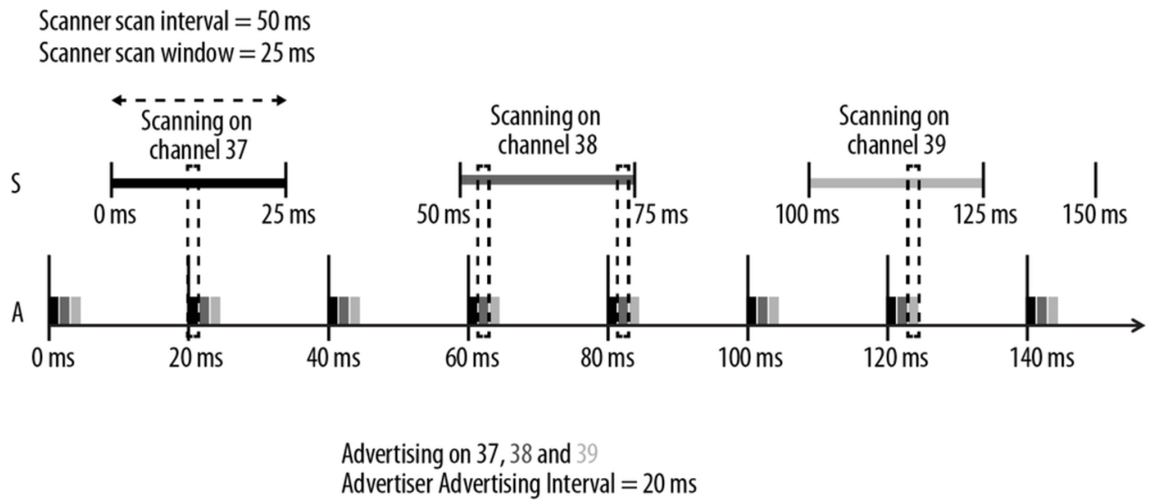


Figure 11 *Scanning interval of central device and advertisement interval of peripheral device can affect a time to find the advertising BLE device (Akiba, et al., 2014).*

As was presented in Figure 9, there are three advertising channels in BLE. These channels are provided as a way for advertising devices to inform other scanning devices of their existence. Through replying to these advertisements, it is possible to establish connections between devices, if allowed or desired. From an application designer's point of view, advertising and discovering peripheral devices is one of the most visible parts performed by the LL, and these can also be tuned by adjusting three parameters of the BLE network: advertising interval, scan interval and scan window. In BLE the advertising is done at set intervals, on the three advertising channels at almost same time. To minimize a chance of consecutive collisions, a random delay of up to 10ms is added on top of the set advertising interval. Figure 11 shows how the three parameters can affect discovering of a device. As the scanner does not scan more than one channel at a time, discovering of the device can take a long time if scan window is small and both advertising and scan intervals are long. This is especially true when there is a lot of traffic and interference with many collisions occurring. Increasing both the scan window and decreasing the intervals would however consequently also increase power consumption, both in the advertising device and the scanner. Therefore, compromises need to be done by the user to estimate the optimal power consumption in relation to how often the advertising packets needs to be received.

For establishing different advertisement types in BLE, there are three properties defined. These properties and advertising packet types defined by them have been presented in Table 1. Connectability defines whether a Central device can establish a connection with the peripheral or not. Scannability means, that after scanner device has received an advertisement packet from the broadcaster, it can "scan" the advertiser by sending it a request packet asking for more information that the advertiser has to offer. Then the advertiser sends another packet with additional data. This does not provide any means for the scanner to send any data to the advertiser though, as it can only send a scan request. Finally, there is a possibility to direct the advertising packet to a specific device, with no data payload allowed in the advertising packet. This is used for offering the scanner device a possibility to connect, without sending any other information.

Table 1 *Advertising packet types (Akiba, et al., 2014).*

Advertising Packet Type	Connectable	Scannable	Directed	GAP Name
ADV_IND	Yes	Yes	No	Connectable Undirected Advertising
ADV_DIRECT_IND	Yes	No	Yes	Connectable Directed Advertising
ADV_NONCONN_IND	No	No	No	Non-connectable Undirected Advertising
ADV_SCAN_IND	No	Yes	No	Scannable Undirected Advertising

Connection itself is established by the master, who first scans for advertising packets from slave, and then sends a connection request. In BLE, the connection between devices is defined as a connection event, which happen at predefined times at specified channels (determined by hop), which have been agreed at the time of establishing the connection. A connection event consists of a sequence of data exchanges between slave and master. Between connection events, devices' radios go to idle mode to save energy. Connection events can be changed to anything between 7,5ms and 4s. This is also called connection interval. The data packets have a payload of 27 bytes, of which about 20 bytes are usable depending on the selected payload protocol. Linker layer also makes sure that the packets have been received, by resending all the packets until they have been acknowledged. Acknowledgement requires that the 24-bit CRC matches that payload, and that packets were received. (Bluetooth SIG, 2013). This ensures that the BLE's data transmission is reliable.

Host

As seen in Figure 10, the components of Host block are Logical Link Control and Adaptation Protocol (L2CAP), Security Manager (SM), Generic Access Profile (GAP), Attribute Protocol (ATT), and Generic Attribute Profile (GATT). Here follows a short summary of each component with some key-points listed.

Logical Link Control and Adaptation Protocol (L2CAP)

L2CAP is the layer between HCI/Controller and upper layers (GAP, GATT, application), and takes care of the data transfers between them (Texas Instruments, 2016). Like TCP, it provides a way for multiple protocols to use a same single physical link. L2CAP multiplexes data between higher layer protocols and encapsulates and breaks them into smaller 27-byte BLE data packets, and then at the receiver's side joins the packets back together for the higher layers.

Security Manager (SM)

BLE supports devices to securely connect with each other by either doing a one-off pairing, or a more permanent bonding. Both ways offer a secure link for connection but bonding between devices will be remembered also after disconnection.

Attribute Protocol (ATT)

ATT is a simple client/server protocol which allows simple attributes to be shared from a server to a client. Basically, this means that the devices offer some data in the form of attributes, which compose of three parts: a 16-bit handle, and UUID which defines a type of the attribute, and a value of certain length. The ATT protocol offers functions for reading and writing values, having permissions etc. (Epxx, 2019)

Generic Attribute Profile (GATT)

GATT is a profile in BLE which provides means of hierarchically structure sets of ATT attributes, and use them to provide more meaningful services to the client. GATT encapsulates the data into services, which consist of one or more characteristics. Bluetooth Classic has multiple profiles for transferring data between devices. These devices and their profile can be e.g. handsfrees, speakers and game controllers. BLE only implements two profiles: GATT profile and GAP. Of these two, GATT is the actual data transmission profile. GATT profile offers many predefined services, already in BLE standard, such as *HR service* for offering a client HR data. *HR Measurement* and *Body Sensor*

Location are two of the characteristics which it has. It is also possible to define own services. (SIG, 2019). If a device, e.g. HR sensor wants to let other devices to know that it provides a HR service without needing to first connect to the sensor, it needs to add GATT Attribute data into its advertising packet's payload. Otherwise other devices might not allow connecting to that sensor for receiving HR data.

Generic Access Profile (GAP)

GAP is a profile which is used at the higher level to configure how the control layer should be used, e.g. regarding to device's discoverability, connection and security establishment. GAP defines the framework for BLE devices, that they must follow to successfully operate with each other. It is in GAP where all the roles (central, peripheral, observer, broadcaster) and many other aforementioned matters are defined at the high level. Every BLE device should have a mandatory GAP service to be accessed via GATT. This service provides for example Device name characteristic.

GAP also specifies the advertising data format of the advertisement payload. In BLE the payload available for user is 31 bytes. GAP defines that the data constructed within payload needs to be in the following format: length (1 byte), AD Type (Advertising Data Type, 1 byte), and the actual data (variable length). Bluetooth SIG has specified some of the most common AD types, but there's also a Manufacturer Specific Data AD type available for own free form data. A full structure of an advertisement BLE packet is presented in Figure 12. One of the common AD types is Flags which informs scanners about the discovery or type of the advertisement packet, as presented in Table 1.

It must be noted that the space in advertisement payload is very limited, and e.g. after including the overhead of Flags and Manufacturer specific data, there would be only 26 bytes left for the application data (1B length, 1B Flags AD type, 1B variable + 1B length, 1B Manufacturer Specific data AD type, 26 B for the data). If additional GATT services, such as HR service needs to be advertised, this further takes away from the application space.

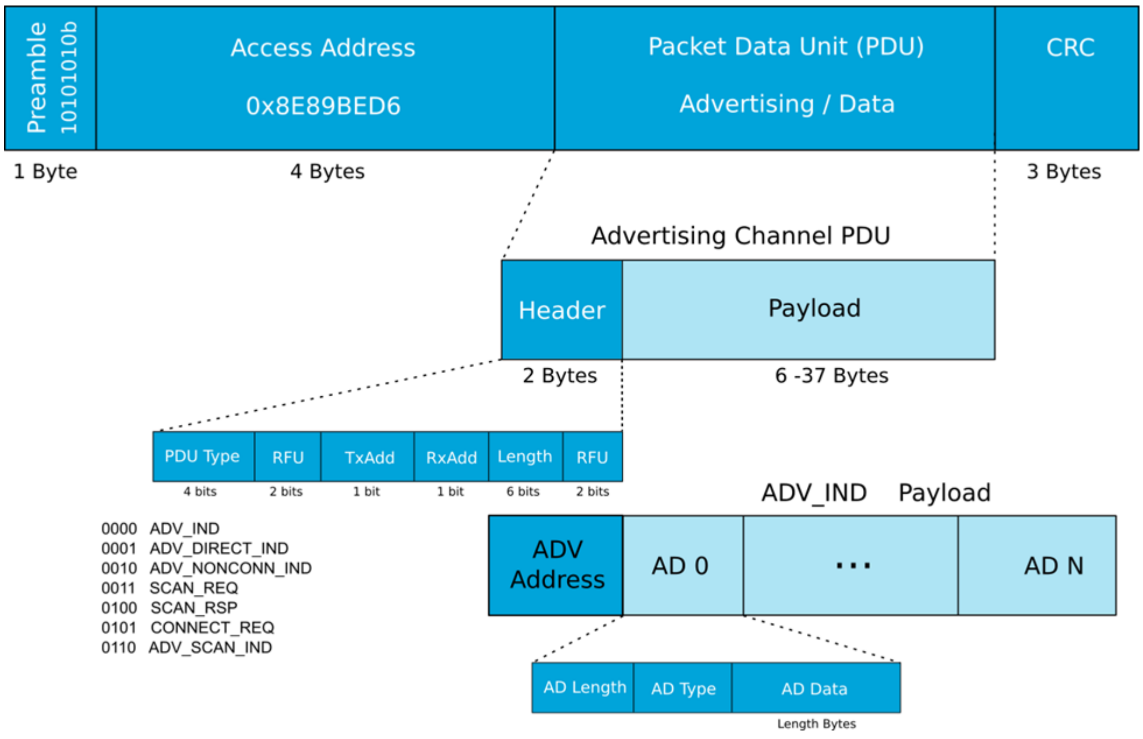


Figure *12 Structure of a BLE advertising packet (Ardelean, 2015).*

Application

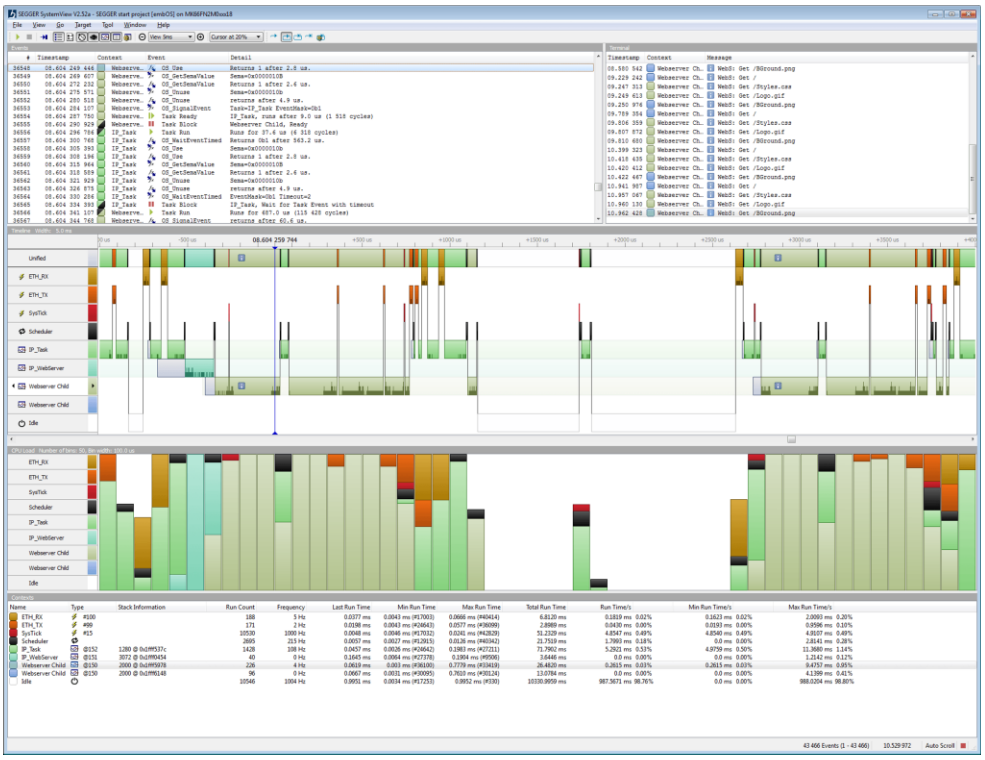
Application block is the highest layer in the Bluetooth device. It contains everything related to the actual application logic, such as user interfaces, state logic or whatever the user wants to do with the device. Application architecture as such has not been specified at all by any standard, but is presented together with host and controller, to give an idea of where the application of the user resides in the whole big picture.

3.2 FreeRTOS

FreeRTOS kernel is a real-time operating system (RTOS) for embedded devices, which has been licenced under MIT licence as part of *Amazon Web Services (AWS) open source project*. Therefore, the sources are available for free of charge to be used in any commercial applications, without need to expose user's proprietary code of their application. FreeRTOS is a market leading RTOS with more than 100K downloads per year, it is supported by over 35 microcontroller architectures, and it is actively being developed by AWS. Originally FreeRTOS started in 2003 by Richard Barry, who later transferred the stewardship of the development to AWS in 2017. The primary design goals of FreeRTOS are ease of use, small footprint, and robustness. (Microchip, 2019) (FreeRTOS, 2019) (Lacamera, 2018)

Some of the FreeRTOS's key features and their short descriptions (Gay, 2018):

- **Threads or tasks for multi-tasking and scheduling**
    - Round-robin scheduling is used to assign time slices for processes in equal portions in circular order
    - Thread priorities are supported
    - Mutexes or locks for providing thread-safe operations by limiting access to resources which are being used by other threads
    - Semaphores for managing used resources
    - FreeRTOS uses queues for communication between interrupts, threads or tasks. They can be used as thread safe FIFO (First in First Out) buffers, to send data to a queue of another thread, to be handled in that thread when the queue has passed. (FreeRTOS, 2019)
- **Pre-emptive scheduling option**
    - Pre-emption means that a thread or task can be interrupted by the system to handle e.g. time-critical interrupts from an MCU at the time when they occur. (Wikipedia, 2019)
- **Efficient Software timers**
- **Tick-less mode for low power applications** (Lacamera, 2018)
- **Flexible memory management**
    - Applications can be completely statically allocated, or alternatively FreeRTOS provides five different schemes of memory allocation for dynamic allocation.
- **Small footprint.** FreeRTOS has 6K to 12K read only memory (ROM) footprint
- **Low overhead**
- **Trace support for debugging in real-time**
    - With tracing it is possible to solve complex problems and understand how the system works
    - Tracing can help find answers to questions such as: how often interrupts occur, what is the program flow, how long do tasks run and how long do they need to wait.
    - Tracing allows more thorough profiling of the system to be done, to evaluate code timing and performance.
    - Figure 13 A commercial analysis tool SystemView by Segger, which supports FreeRTOS trace .



```
  #    Timestamp       Context      Event               Detail
36548  08.604 249 446  Webserve...  OS_Use              Returns 1 after 2.8 us.
36549  08.604 269 607  Webserve...  OS_GetSemaValue     Sema=0x0000010B
36550  08.604 272 232  Webserve...  OS_GetSemaValue     Returns 1 after 2.6 us.
36551  08.604 275 571  Webserve...  OS_Unuse            Sema=0x0000010b
36552  08.604 280 518  Webserve...  OS_Unuse            returns after 4.9 us.
36553  08.604 284 107  Webserve...  OS_SignalEvent      Task=IP_Task EventMask=0b1
36554  08.604 287 750  Webserve...  Task Ready          IP_Task, runs after 9.0 us (1 518 cycles)
36555  08.604 290 929  Webserve...  Task Block          Webserver Child, Ready
36556  08.604 296 786  IP_Task      Task Run            Runs for 37.6 us (6 318 cycles)
36557  08.604 300 768  IP_Task      OS_WaitEventTimed   Returns 0b1 after 563.2 us.
36558  08.604 305 393  IP_Task      OS_Use              Sema=0x0000010b
36559  08.604 308 196  IP_Task      OS_Use              Returns 1 after 2.8 us.
36560  08.604 315 964  IP_Task      OS_GetSemaValue     Sema=0x0000010B
36561  08.604 318 589  IP_Task      OS_GetSemaValue     Returns 1 after 2.6 us.
36562  08.604 321 929  IP_Task      OS_Unuse            Sema=0x0000010b
36563  08.604 326 875  IP_Task      OS_Unuse            returns after 4.9 us.
36564  08.604 330 286  IP_Task      OS_WaitEventTimed   EventMask=0b1 Timeout=2
36565  08.604 334 393  IP_Task      Task Block          IP_Task, Wait for Task Event with timeout
36566  08.604 341 107  Webserve...  Task Run            Runs for 687.0 us (115 428 cycles)
36567  08.604 344 768  Webserve...  OS_SignalEvent      returns after 60.6 us.
```

```
Timestamp   Context          Message
08.580 542  Webserver Ch...  WebS: Get /BGround.png
09.229 242  Webserver Ch...  WebS: Get /
09.247 313  Webserver Ch...  WebS: Get /Styles.css
09.249 613  Webserver Ch...  WebS: Get /Logo.gif
09.250 976  Webserver Ch...  WebS: Get /BGround.png
09.789 354  Webserver Ch...  WebS: Get /
09.806 359  Webserver Ch...  WebS: Get /Styles.css
09.807 872  Webserver Ch...  WebS: Get /Logo.gif
09.810 680  Webserver Ch...  WebS: Get /BGround.png
10.399 323  Webserver Ch...  WebS: Get /
10.418 435  Webserver Ch...  WebS: Get /Styles.css
10.420 412  Webserver Ch...  WebS: Get /Logo.gif
10.422 467  Webserver Ch...  WebS: Get /BGround.png
10.941 987  Webserver Ch...  WebS: Get /
10.957 067  Webserver Ch...  WebS: Get /Styles.css
10.960 130  Webserver Ch...  WebS: Get /Logo.gif
10.962 428  Webserver Ch...  WebS: Get /BGround.png
```

Name	Type	Stack Information	Run Count	Frequency	Last Run Time	Min Run Time	Max Run Time	Total Run Time	Run Time/s	Min Run Time/s	Max Run Time/s
ETH_RX	#100		188	5 Hz	0.0377 ms	0.0043 ms (#17003)	0.0666 ms (#40414)	6.8120 ms	0.1819 ms 0.02%	0.1623 ms 0.02%	2.0093 ms 0.20%
ETH_TX	#99		171	2 Hz	0.0198 ms	0.0043 ms (#24643)	0.0577 ms (#36099)	2.8989 ms	0.0430 ms 0.00%	0.0193 ms 0.00%	0.9596 ms 0.10%
SysTick	#15		10530	1000 Hz	0.0048 ms	0.0046 ms (#17032)	0.0241 ms (#42829)	51.2329 ms	4.8547 ms 0.49%	4.8540 ms 0.49%	4.9107 ms 0.49%
Scheduler			2695	215 Hz	0.0057 ms	0.0027 ms (#12915)	0.0126 ms (#40342)	21.7519 ms	1.7993 ms 0.18%	0.0 ms 0.00%	2.8141 ms 0.28%
IP_Task	@152	1280 @ 0x1fff537c	1428	108 Hz	0.0457 ms	0.0026 ms (#24642)	0.1983 ms (#27211)	71.7902 ms	5.2921 ms 0.53%	4.9759 ms 0.50%	11.3680 ms 1.14%
IP_WebServer	@151	3072 @ 0x1fff0454	40	0 Hz	0.1645 ms	0.0064 ms (#27378)	0.1904 ms (#9506)	3.6446 ms	0.0 ms 0.00%	0.0 ms 0.00%	1.2142 ms 0.12%
Webserver Child	@150	2000 @ 0x1fff5978	226	4 Hz	0.0619 ms	0.003 ms (#36100)	0.7779 ms (#33419)	26.4820 ms	0.2615 ms 0.03%	0.2615 ms 0.03%	9.4757 ms 0.95%
Webserver Child	@150	2000 @ 0x1fff6148	96	0 Hz	0.0667 ms	0.0031 ms (#30095)	0.7610 ms (#30124)	13.0784 ms	0.0 ms 0.00%	0.0 ms 0.00%	4.1399 ms 0.41%
Idle			10546	1004 Hz	0.9951 ms	0.0034 ms (#17253)	0.9952 ms (#330)	10330.9959 ms	987.5671 ms 98.76%	0.0 ms 0.00%	988.0204 ms 98.80%

43 466 Events (1 - 43 466) 10.529 972

***Figure 13** A commercial analysis tool SystemView by Segger, which supports FreeRTOS trace (SEGGER Microcontroller Systems LLC, 2019).*

Some of the more advanced features which FreeRTOS lacks, are e.g. device drivers, advanced memory management, user accounts and networking. These are usually found in traditional operating systems (OS) such as Linux, Microsoft Windows of OSX. Adding more advanced features would however quickly stray away from the original focus of being a small and simple. Also being a real-time OS means that the operating system has strict processing time requirements that it needs to fulfil, being consistent in the amount of time that it takes to process a task. This limits real-time OS's use-cases as to what it sensible to be done with it, as it is primarily designed for embedded systems with hard time constraints, rather than PCs or even single-board computers. (Wikipedia, 2019)

3.3 Representational State Transfer (REST)

Representational State Transfer (REST) is an architectural style for helping create and organize distributed systems. It was first introduced by Roy Fielding in the year 2000 (Fielding, 2000). The style does not imply any specific rules or guidelines, but rather an ideology of how distributed systems should be organized. REST itself is protocol-independent, defined to be used within a URI scheme. REST -type style has been adopted mainly to the area of creating Web Services, and therefore these RESTful Web Services (RWS) are commonly using a set of HTTP methods and responses for their operations.

RESTful system can however be implemented without using any previously defined protocols. (Doglio, 2018).

REST -style of system aims to improve distributed systems by e.g. performance improvement, having simple and efficient communication style, adding scalability, portability, reliability and visibility.

REST -style sets some constraints for the system, as it most conveys the following styles:

- Client-Server -based architecture
- Stateless system: each request sent by client must have all the information required. Server cannot use any previously saved data to interpret the request received.
- Cacheable: Server does not need to run database requests for every client request, but cached data can be used in the answer.
- Uniform interface: All clients and servers should use uniform interfaces for communication
- Layered system: RESTful services' components in the server should be separated to layers, which communicate only layers below or above.
- Code-on-demand: This optional constraint allows client to receive executable code from the server, such as JavaScript scripts etc.

In RESTful system anything that can be conceptualized, can be thought as a *resource*, whether it be an image or a textual document. These *resources* can then have one or more representations of the same resource, such as different file formats. Resource identifiers used in RESTful systems are usually *unique resources identifiers*, or URIs, such as */mem/datalogger/config*. RESTful system also defines that there can be actions which can be performed on these resources. As mentioned earlier, HTTP protocol is commonly used as a standing point to define the possible actions. Table 2 presents some of the common HTTP actions. It is usual that only some of the commands are applicable per specific resource, instead of all.

***Table 2** Some of the most common HTTP actions with explanations (Doglio, 2018).*

HTTP Command	Action
GET	Access a resource without altering it (read-only).
POST	Create a new something (such as resource) in the server.
PUT	Updates something in the server.
DELETE	Removes a resource from the server.

As well as commonly using HTTP commands for communication between client and server, HTTP status codes are also used to return status values for the received HTTP commands. In HTTP these status codes are numbers which have pre-defined meanings, trying to cover most of the use cases in the internet. Table 3 presents five groups of HTTP status codes and some common example codes. (Doglio, 2018).

Table 3 *Five groups of HTTP status codes and some common example codes (Doglio, 2018).*

HTTP Status Code	Explanation
1xx	**Informational status**
2xx	**Request was completed successfully. Provides desired content for the client.**
200	OK. Request completed well and content returned to client.
201	Created. Content was created e.g. after POST or PUT.
204	No Content. This can be used e.g. after DELETE, when request was OK, but there is no content to be returned.
3xx	**Resource in the server has been moved.**
301	Moved Permanently. If a resource in the server has moved, this returns the new URI for the resource.
4xx	**The request of the client did something wrong**
400	Bad request. This is a common response if the request was malformed or was lacking something.
403	Forbidden. Request tried to do something forbidden.
404	Not found. Resource of the request was not found from the server.
5xx	**Internal error within the server caused a crash of the server**
500	Internal server error. This generic code is received by the client if server crashes due to some internal bug or unexpected behaviour.

As RESTful systems also most often need to send some data between client and server, an open-standard file format called JavaScript Object Notation (JSON) is commonly used. JSON encapsulates the data using a human-readable text format, where data objects consist of attribute-value pairs and array data types. The JSON format originates from JavaScript, but as such it is a language-independent data format. (Doglio, 2018).

An example of JSON -formatted HTTP GET response:

```
{
  "response": 200,
  "responsestring": "HTTP_CODE_OK",
  "operation": "get",
  "uri": "/net/184030000281/Meas/Acc/Info",
  "content": {
    "SampleRates": [
      13,
      26,
      52,
      104,
      208,
      416,
      833,
      1666
    ],
    "Ranges": [
      2,
      4,
      8,
      16
    ]
  },
  "querytimems": 37,
  "querytimens": 37696937
}
```

YAML (YAML Ain't Markup Language) is a human-readable data serialization language or data storage format, which is a superset of JSON. YAML has a minimal syntax, and it uses indentation to indicate nesting, [] for lists and {} for maps. Both JSON and YAML were initially developed in the early 2000's. YAML is the data format, which has been adopted to a RESTful API specification called Swagger specification, or later OpenAPI Specification (since 2016).

However, YAML is just a file format used by the OpenAPI specification. The actual OpenAPI specification further standardizes how to actually write a desired machine-readable interface file for a RESTful API. These interface files are made to describe, produce, consume and visualize RESTful web services.

An example of a Swagger 2.0 compliant YAML formatted file, defining a RESTful API for GET request on the resource path */Ui/Ind/Visual*:

```
Ui_resource.yaml

swagger: '2.0'

info:
  version: NA
  title: Indication - Movesense-API
  description: |
    API for creating different types of user indications.
  x-api-type: public
  x-api-required: true

paths:
  /Ui/Ind/Visual:
    get:
      description: |
        Get the state of the visual indication
      responses:
        200:
          description: Current state of the visual indication
          schema:
            $ref: '#/definitions/VisualIndState'

definitions:
  VisualIndState:
    required:
      - state
    properties:
      state:
        $ref: '#/definitions/VisualIndType'
  VisualIndType:
    type: integer
    format: uint16
    minimum: 0
    maximum: 2
    x-enumNames:
      - NO_VISUAL_INDICATIONS
      - CONTINUOUS_VISUAL_INDICATION
      - SHORT_VISUAL_INDICATION
```

In practice these yaml files need to be translated to another programming language, to correspond a specific environment requirement. OpenAPI specification or YAML as such do not provide means for any programmability, and therefore these yaml files are used to generate the actual source codes for the target language such as Java, PHP, Node.js, or Go. Currently the official OpenAPI specification version 3.0 does not have a supported implementation e.g. for C or C++. This is due to fact that usually web services and RESTful APIs are created with more high-level languages. These languages have been designed primarily for the same internet environments where RESTful APIs are primarily being used. (OpenAPI Initiative, 2019)

3.4 Introduction to Suunto Movesense platform

Movesense sensor is a commercial product offered by Suunto. Movesense is being described as an open development environment for motion sensing solutions. Its idea is to offer developers and 3rd party companies a possibility to prototype, develop and commercialize their own product ideas and concepts with the Movesense platform, and then sell it as their own product. Movesense can be either re-branded as customer's own product (sold as a white-label product), or with the Movesense-branding itself. With

Movesense platform, users can bring their own products to markets without the need to design and manufacture their own hardware. Developing an own Bluetooth radio device can be a relatively costly operation for smaller companies, as Bluetooth SIG qualification fees start from of thousands of euros. In addition, development, testing, manufacturing and RF qualification for CE/FCC are all costly operations, especially if volumes are low (Bluetooth SIG, 2019). This makes Movesense a viable option for companies who want to focus on the actual application side of the product, also without caring too much about the underlying low-level architecture of the system. Figure 14 presents two Movesense sensors and a snap-button HR belt. (Suunto Movesense, 2019)

Figure 14 *Movesense sensor and a HR chest strap (Suunto Movesense, 2019).*

As a platform, Movesense development environment offers both hardware and software for developers. On hardware side the actual sensor device is being offered together with its programming jig, including SEGGER's J-Link Base debugging probe. On software side, both mobile libraries and interfacing libraries for the sensor device itself, are offered. These libraries accompanied with their documentations are openly provided in company's Bitbucket Git-repository (Suunto Movesense, 2019).

Movesense sensor intends to be a programmable multi-purpose device with Bluetooth connectivity, external memory and different sensors. Sensors included in the device are accelerometer, magnetometer, gyroscope, thermometer and an optional one-channel HR sensor. More detailed specifications are presented in Table 4.

Table 4 *Technical specifications and features of Movesense sensor (Suunto Movesense, 2019).*

Component	Description
Sensor casing	Diameter: 36,6mm, thickness: 10,6mm. Water resistant to 30m. Weight 10g with battery.
MCU: NRF52832 (Nordic Semiconductor)	A SoC comprising 32 –bit ARM® Cortex®-M4 with 64kB on-chip RAM and 512kB on-chip FLASH, Integrated Bluetooth Low Energy 4.0 radio, 2,4GHz
Additional memory for data logging	3Mbit EEPROM (comprising of separate 1Mbit + 2Mbit chips)
Accelerometer and gyroscope: LSM6DSL (STMicroelectronics)	- Accelerometer, range: ±2/±4/±8/±16g full scale, sampling frequency: 12.5/26/52/104/208Hz - Gyroscope, range: ±125/±245/±500/±1000/±2000 dps full scale, sampling frequency: 12.5/26/52/104/208Hz
Magnetometer: LIS3MDL (STMicroelectronics)	Magnetometer, range: ±4/±8/±12/±16 gauss full scale
Temperature sensor: TMP112 (Texas Instruments)	Temperature, accuracy: ±0.5°C (max) from 0°C to +65°C and ±1.0°C (max) from –40°C to +125°C
Biopotential Analog Front-End (AFE): MAX30003 (Maxim Integrated)	Ultra-Low Power, Single-Channel Integrated Biopotential (ECG, R to R Detection) AFE
I/O	- Led (red) on the front - 1 –wire interface master (accessible through Movesense connector) - Debug UART, debug I/F, power & ground (inside the battery slot)
Energy Source	CR 2025 3V lithium coin cell battery

Software development tools offered by Movesense platform include the following:

- **MovesenseCoreLib** library for the sensor device. Library abstracts away the low-level handling of the embedded system, offering an asynchronous REST API platform and providing access to Bluetooth and device's sensors. The API specification is based on Swagger 2.0 syntax.

- **MDS-library** for mobile devices to interface with the sensor device over Bluetooth connection. Library is available for both iOS and Android devices.
- **PC Simulator** for simulating some of the device's functionalities on a Windows environment. Simulator can be used for debugging and developing user's own applications without the requirement to access to a physical sensor device.
- **Wbcmd tool** (whiteboard command tool) for computer, for communicating with the sensor device over USB-port, when developing and debugging the device on a programming jig. Wbcmd tool can also communicate locally with a PC Simulated sensor device, running a simulation locally on the computer from an executable file.

3.5 NRF52832 MCU

The Movesense sensor used in this project has a BLE SoC manufactured by Nordic Semiconductor. Nordic Semiconductor is a Norwegian company which is part of Bluetooth SIG. The company was first established in 1983. They have been developing solutions for BLE for several years and are widely known for their wireless solutions. One of their famous products Is Nrf52832, which is also found inside Movesense. It is part of their Bluetooth 5 capable NRF52 series. Nrf52832 is a general-purpose multiprotocol SoC, which is built on ARM Cortex-M4 CPU running at 64 MHz. The SoC has 64kB on-chip RAM and 512kB on-chip FLASH. Nrf52832 also has a very low power consumption, making it ideal for wearable devices with small batteries. (Wikipedia, 2019) (Nordic Semiconductor, 2019) (Akiba, et al., 2014)

To accompany the products that Nordic offers, they provide something called SoftDevices, which are precompiled and linked binary images implementing desired radio protocol stacks into Nordic's devices. The underlying problem which SoftDevices solve, is to abstract the access to the very complex processor's resources regarding to the use of radio interfaces included in the SoC. Providing SoftDevices as a binary image, also allows Nordic Semiconductor to further continue the development of different radio protocols and systems. SoftDevice is also separated from user application and can be changed independently without user needing to link to any external libraries. (Akiba, et al., 2014)

Nordic has a variety of different SoftDevices available for multiple different protocols, such as BLE, Zigbee, Thread, ANT and 802.15.4. These are all different low power short-range wireless protocols that are being used for different use cases. Nordic Semiconductor's SoftDevices are also able to co-exist within same application, to allow multiple communication protocols been used at the seemingly same time. Nevertheless, as BLE being the most popular of these, being found from most mobile consumer devices, it is also the choice being used in Movesense sensor, with no other protocols being used. (Akiba, et al., 2014)

All the benefits of a SoftDevice solution considered, the downsides of it are also important to understand. SoftDevice requires system resources from the SoC, which will not be available for the user's own application. Depending on the SoftDevice, it can take e.g. 80KB of flash and 8KB of SRAM memory. Also, it naturally adds latency and architectural limitations, as user's own application will be running on top of SoftDevice's system, using software interrupts on ARM core to communicate with the SoftDevice. Since a SoftDevice acts as a black box, it is impossible to debug its inner workings if something goes wrong. (Akiba, et al., 2014)

The SoftDevice which has been included within Movesense sensor, is version S132 SoftDevice. It implements a Bluetooth low energy central and peripheral protocol stack solution, supporting up to twenty connections with and additional observer and a broadcaster role all running concurrently. S132 SoftDevice is Bluetooth 4.2 compliant, and it provides a Master Boor Record (MBR) for over-the-air (OTA) device firmware updates. With the SoftDevice, application and bootloader can be updated separately. The SoftDevice runs a thread-safe supervisor-call based API, with asynchronous, event-driven behaviour. For the security and robustness, there is also a memory isolation between application and the protocol stack.

Figure 15 shows the architecture of the nRF52 software. The lowest-level HW is accessed by the ARM CMSIS interface and building on top of that there is a master boot record, profile and application code, application specific peripheral drivers, and firmware module identified as a SoftDevice. SoftDevice itself further consists of separate parts implementing the stack itself together with its Application Programming Interface (API) and managing the use of hardware and SoftDevices. The API gives access for the actual application development, being used by the SoftDevice to trigger interrupts as events to the application. The application then needs to respond to these events and make sure that e.g. events for BLE control procedure are serviced, to prevent the procedures from timing out resulting in link disconnection. The event-driven architecture therefore ensures that the time-critical Bluetooth radio stack is always being served first in the priorities. The lower priorities, such as the user's application, are then handled through on their own time. This should not, however limit the user application too much. (Nordic Semiconductor, 2017)

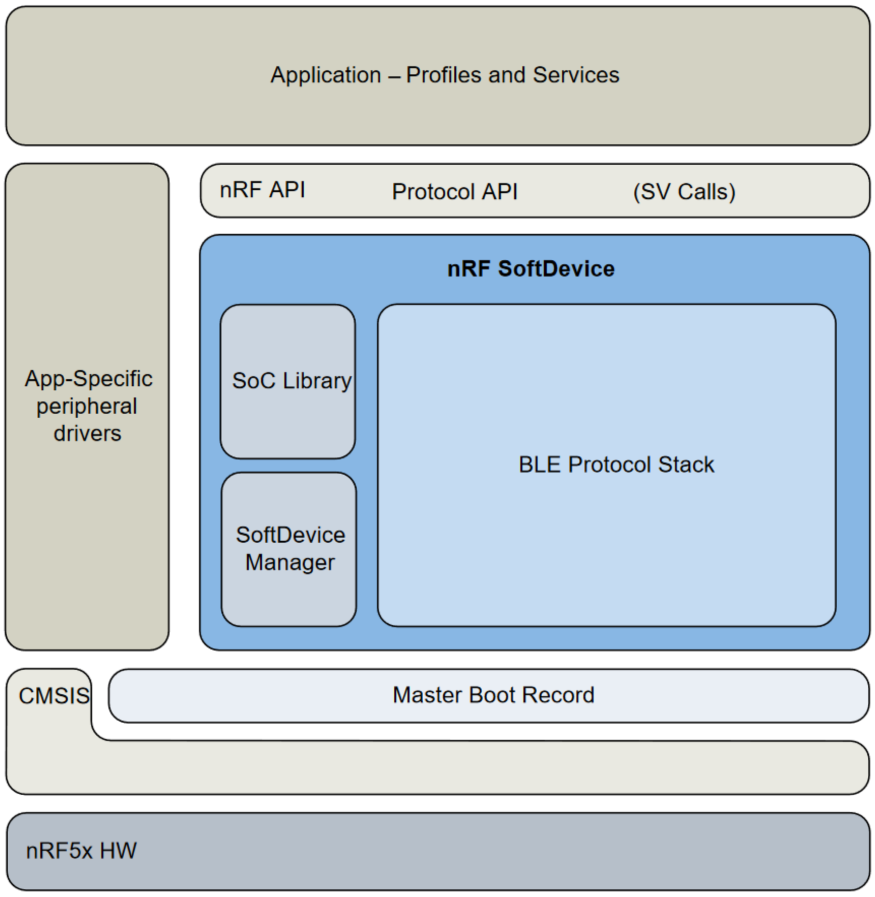


Figure 15 *nRF52 series software architecture, when using SoftDevice (Nordic Semiconductor, 2017).*

From the point of view of developing a Movesense-application, the underlying architectures have not been made very visible for the developers. Movesense software abstracts away most of its architecture. For example, setting up of SoftDevice, Nordic's SDK and FreeRTOS are mainly done within compiled binaries without too much documentation. However, understanding the underlying systems can be beneficial especially when trying to understand why something goes wrong. The SoftDevice API functions, which user application uses, return error codes which are used to verify the correctness of functioning of the SoftDevice and application. Unrecoverable failures (faults) detected by the SoftDevice are reported to the application and can be used for debugging errors. Two types of faults can be reported: SoftDevice assertions and attempts by the application to perform unallowed memory accesses. These so-called *hard faults* can also be accessed via MovesenseCoreLib together with the MovesenseCoreLib’s own assertion notifications.

OTA-updates are an especially critical element of consumer devices, where the end-user cannot possibly have any other way to update the device, other than with their smartphone or equivalent. Considering the use case of Team Sports Monitoring system under development in this thesis, having OTA-updates allows the company not only to fix possibly found issues with minimal intrusion, but also to add new features to the sensors. These new features can be anything from real-time algorithms to features supporting better user experience.

Power management is managed by SoftDevice's *POWER API*. It is designed for optimizing the power usage. When there are no application events from the user, the CPU will go to idle state to save power. User application will only be woken up by application events. SoftDevice interrupts will be handled by the SoftDevice directly. These features allow the SoC to be run for a very long time with being able to optimize the application's power management, e.g. by using tick-less mode when using FreeRTOS. (Nordic Semiconductor, 2017, p. 16)

By default, SoftDevice isolates its memory and peripherals from the application, both in program memory and in RAM. This ensures that the application won't accidentally write over any of SoftDevice's memory, or otherwise interfere the SoftDevice, making the system's performance more predictable and robust. This is called sandboxing and runtime protection. An attempt to write to SoftDevice's protected memory regions will result in an unrecoverable error. (Nordic Semiconductor, 2017, pp. 16-17)

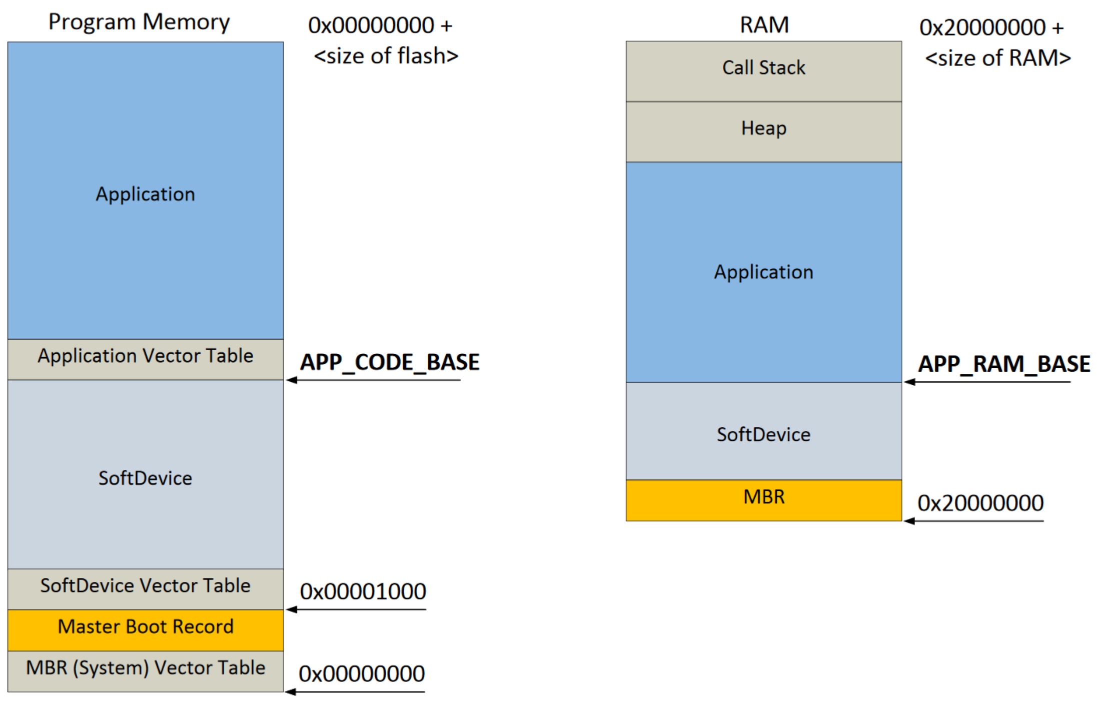


Figure 16 *Memory mapping of the SoftDevice S132 (Nordic Semiconductor, 2017, p. 58).*

Memory mapping of the resources of the SoftDevice are presented in Figure 16. The figure shows a general picture of where different parts reside in the program memory and RAM. MBR (Master Boot Record) is responsible for providing and interface which is used for in-system updates of the SoftDevice and bootloader firmware. When used, the optional bootloader is placed within the application region of the program memory. Note that a more detailed program memory map of the MovesenseCoreLib will be presented in Chapter 3.9, as this is rather a general view of the SoftDevice’s memory mapping.

3.6 Developing for Movesense platform

As it has been said already, developing your own software for Movesense platform has been aimed to make as easy as possible. This should allow user to focus on developing their own application without need to specify or even understand the embedded software architecture running below the surface. Therefore, developer relies heavily on the resources which have been provided in MovesenseCoreLib, and there is not much lower level control over the hardware. Some of the things managed in the background include:

1. Interfacing the hardware components, such as EEPROM memory, IMU-sensors and analogue front end (AFE) / HR sensor
2. Bluetooth protocols, data serialization and transferring, connectivity, bonding
3. Setting up and managing of a SoftDevice, which runs the Bluetooth stack
4. Setting up of system's real-time operating system (RTOS) and managing all the threads, priorities and other system calls
5. Setting up of an event based proprietary Whiteboard platform to offer the REST APIs for developers

Although it has been made easy for a developer to ignore what happens in the background of MovesenseCoreLib, some system components will still be briefly presented in the following sections. This will be done at the level of necessity to understand the basics, but also no more information can be given about the system other than what Suunto has provided with its documentations. As most of the Suunto Movesense library components provided with MovesenseCoreLib, are pre-compiled object files, with only headers provided, it is impossible to understand how some parts of the system work, or why they exist. Also, it must be noted, that the version of MovesenseCoreLib which is under inspection in this thesis, is the newest version available at this current time, version 1.8.1. As the outcome of the project, which this thesis presents, was mainly the software application developed onto the Movesense sensor, the following sections will focus on the matters relating to the sensor development. Therefore, the mobile MDS-library will not be discussed within this thesis.

3.7 Architecture of MovesenseCoreLib

The architecture of complete Movesense application running in the sensor, comprises of several different parts. The main blocks of this architecture are presented in Figure 17. These blocks include the underlying hardware and SoftDevice, FreeRTOS, Suunto's Whiteboard, Movesense-specific libraries and the user's own application code. White-board is Suunto's proprietary communication library. Whiteboard sets up the environ-ment for defining and managing of REST APIs. These APIs are the core of everything in the Movesense. Through these APIs it is possible to handle information between the Whiteboard system itself, and Movesense sensor specific library modules available. Whilst detailed handling of hardware, SoftDevice, FreeRTOS and Whiteboard are mainly hidden behind the scenes, there exists some APIs which provide means of communica-tion between these parts of the system. 1-Wire chip (DS24L65) and HR sensor (MAX30003) can for example be accessed at a register level. The level of access varies between resources, and therefore one must find for their own application what is needed and what is available. At the top of the architecture lays the user's own application, which will be built on top of the Whiteboard's libraries, together with Movesense specific re-source libraries.

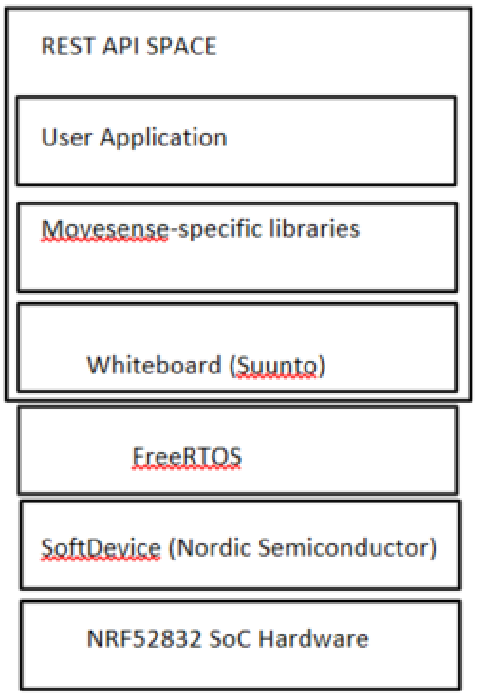


Figure 17 *System architecture of Movesense sensor software.*

3.8 Build process and flashing of an application

Movesense application building is managed by a cross-platform build process management system Cmake. It has compiler independent configuration files, allowing targeting different native compilers if wanted or needed. The default build system for Movesense is Ninja. It has been designed to be fast in running builds, but also to have its Ninja build files to be generated by a higher-level build system, such as Cmake. When creating an own application for Movesense, developer must have a *CMakeLists.txt* file within their application folder, just as all the sample applications have. This file can be easily modified to e.g. link external binary libraries to the project. The *CMakeLists.txt* doesn't otherwise have much inside, as most of the build has been managed in other cmake-files found within MovesenseCoreLib. The main flow of build is described in *MovesenseFromStaticLib.cmake* file, including and defining the rest of the build related files. This cmake file e.g. searches for the pre-built binaries such as wbo-files (whiteboard objects). (Wikipedia, 2019) (CMake, 2019) (Ninja, 2019)

Other important build related tools are Movesense's SBEM-generator and general YAML-to-C++ -generator. The latter is included within the pre-built binaries, and thus cannot be viewed. It is however responsible for generating all the C++ code (*app-resources/resources.h*), based on user's YAML-resource APIs (*wbresources/exampleFile.yaml*). The SBEM-code-generator is a more specific python tool (*sbemCodeGen.py*) used to convert subscriptable resources into compact serialized data formats for more efficient and convenient data handling and saving. The generated results (*sbem_definitions.cpp*) can be inspected to ensure that subscriptions are working as intended. These sbem-definitions are used by *mem/datalogger* and *mem/logbook* services to save and retrieve logged data to and from Movesense's EEPROM memory. Also, the MDS-library uses the generated data descriptors to decode the saved entries.

As mentioned, building a Movesense project is simply done with two tools. First Cmake is run to create the Ninja configuration file *build.ninja*, after which ninja tool itself is run. Additional utility commands, such as "*dfupkg*" or "*flash*" have been also provided to make a zip-file for OTA-update and for flashing a sensor in programming jig. Movesense's documentation presents a typical flow of building and flashing an application (Suunto Movesense, 2019):

```
  cmake    -G    Ninja    -DMOVESENSE_CORE_LIBRARY=../MovesenseCoreLib/
-DCMAKE_TOOLCHAIN_FILE=../MovesenseCoreLib/toolchain/gcc-nrf52.cmake
-DCMAKE_BUILD_TYPE=Release ../samples/hello_world_app
```

```
ninja
ninja dfupkg
ninja flash
```

3.9 Memory mapping and resource usage

When designing an application for a small wearable embedded device, extra care needs to be taken to be aware of the constraints and capabilities of the system. To ensure that we know and understand the limitations of the system, at least following aspects should be considered: MCU's processing power, RAM and flash memory, external memories (e.g. EEPROM) and real-time timing requirements of different components of the system. In addition to hardware limitations we need to consider the constraints and overheads which come from chosen software technologies or libraries. In case of Movesense, much of the choices have been made for the user, which can speed up the development of a new product or a proof-of-concept. MovesenseCoreLib builds on top of Nordic's SoftDevice and SDK, FreeRTOS and Suunto's proprietary libraries. As a result, there is of course some overhead of RAM and flash memory present in the system already. The memory mapping of compiled application, together with other available memory resources is listed below (MovesenseCoreLib 1.8.1):

- Memory mapping of FLASH memory
    - 0x00000 - 0x1f000: MBR (Master Boot Record) + SoftDevice
    - 0x1f000 - 0x74000: Application
    - 0x74000: Manufacturing / Product data
    - 0x75000 - 7E000: Bootloader
    - 0x7E000: System settings
    - 0x7F000: Bootloader settings

All the listed memory addresses can be found after building an application, by inspecting a resulted *target.map* file. However, this summary was received from Suunto. Understanding how the memory is mapped can be a great help when debugging the resulted hex file of an application. Here it must be noted that the application space (348 160 bytes) listed above includes also the rest of the MovesenseCoreLib. To see how much space there is left for a user application, we can build a sample application called plain_app, which has no implementation of any specific application. We can also get the available heap and stack of an application by using REST APIs. These APIs will be listed later.

Summary of the available memory resources for plain_app is listed below:

- FLASH: available application memory (plain_app)
  - Release build 98 992 bytes
  - Debug build 46 488 bytes
- RAM: available volatile memory (plain_app)
  - RTOS's Heap: 11 120 bytes
  - RTOS's application-thread's stack: 4 096 bytes (1 024 words)
- EEPROM: available non-volatile memory
  - 347k bytes

If there is a need to increase or decrease the size of the stack, it naturally also changes the size of the heap. Heap is reserved from the memory which has been left after all the system's stacks have been reserved. MovesenseCoreLib provides various different REST APIs which have been optimised for the Movesense sensor. Most of these APIs have example sample applications provided within the repository. When beginning to develop a new application, it can be useful to first review a similar sample application for reference. Building a sample application can also give a view of how large one's own developed application size would be. By building all the sample applications, it was found that the *custom_gattsvc_app* takes the most memory of them all. Knowing the available application memory space (348 160 bytes) it was calculated that building on top of *custom_gattsvc_app* leaves us about 38 kB left for our own application in debug build, and about 90 kB in release build. Estimating the result size of different combinations of sample apps is not, however, as simple as summing up the app sizes, as some of them share the same libraries and optional modules.

As mentioned in Chapter 3.2, FreeRTOS has some profiling options available in it. However, these options have not been enabled for the user in Movesense. Therefore, further profiling and estimation of runtime overheads for different REST API calls has not been done. The same chapter also briefly mentions that there are different memory schemes available for configuring threads or contexts in FreeRTOS. In Movesense all the stacks of contexts are reserved at the very beginning of setting up the Whiteboard. During the lifetime of the software, those context stacks are never freed. This is the simplest and thread safest of the memory schemes of FreeRTOS, making the OS very stable.

Altogether Movesense has several threads or contexts running in the background. Most of these threads are reserved for Whiteboard, and two of the threads are available for user applications. These are *application* and *meas* threads. The *meas* thread is meant for only real-time operations and processing. For example, */meas/ecg* and */mem/datalogger* are run in *meas* thread. *Application* thread is therefore left for everything else, and especially for long lasting heavy operations. The configurations of these two threads are available in *app_root.yaml*, though the *application* context's stack size can also be overdriven with a macro *MOVESENSE_APPLICATION_STACKSIZE(1024)* in *App.cpp*, where the parameter's datatype is a four byte long word.

3.10 Whiteboard and REST API in MovesenseCoreLib

Whiteboard is Suunto's proprietary communication and resource management library. It is the same library which is run in Suunto's other products, such as smart watches. Each individual product needs to have its own port or modification of the whiteboard, to meet the specific requirements of the product and hardware. Therefore, Movesense sensor also has its own modification of the Whiteboard, which comes with the MovesenseCoreLib. As a proprietary library, little is documented about the whiteboard system itself, but it does offer some useful public REST APIs. Purpose of this section is not to give a full documentation about all the APIs and how to use them, but to give general level information which might be useful to the reader, when at first coming across the MovesenseCoreLib. The REST APIs defined by MovesenseCoreLib follow the same idea of tradition HTTP methods, as was discussed in Chapter 3.3. Further documentation can be found online on the Movesense's Bitbucket.

Some of the most notable REST APIs (application programming interfaces) provided by whiteboard:

- */Whiteboard/Info/Subscriptions* - Returns information about subscribers of a specific resource.
- */Net* - information about connected whiteboard clients. Allows the device to know when someone has a connection to it e.g. from a mobile device.
- */Whiteboard/Metrics/Threads* - Information about the threads running on the system, their states, priorities etc. Especially *freeStack* property is useful, as it gives the miminum amount of stack space that has been unused in bytes.
- */Whiteboard/Metrics/Events* - Returns information about Whiteboard execution contexts' event queues and their usage. Provides helpful metrics, such as *highWaterMark*, and *verySlow,* to tell e.g. if contexts have been blocked by some slow operations.

Whiteboard provides a system for defining own modules, which can function as Clients and/or Resources. This means that according to the roles of these modules, they can provide resources and/or access other modules' resources. For example, a Client can only ask information from other modules, but cannot be asked anything. Resources of modules are called REST APIs, or just APIs. Movesense APIs have two interfaces which are used to access the resources: REST APIs following the Swagger 2.0 format (saved as yaml -file format), and C++ files generated from the yaml-files. MovesenseCoreLib also comes with its own Movesense-sensor -specific APIs.

REST interface allows definition of different resource paths, such as */movesense/Info*. These resources can provide different resources, which in MovesenseCoreLib include the following:

- GET
- PUT
- POST
- DELETE
- SUBSCRIPTION

Subscriptions in yaml-files are defined as POST to path */someResource/Subscription*, but it was included here as in the C++ code and wbcmd.exe -tool it is implemented as its own separate subscription -command. In yaml-file the user can declare HTTP-responses, such as 200 (OK), or 400 (BAD REQUEST), which the resource will execute when user wants to. These resources can also take in parameters, and they can return values, such as integers, vectors, strings, or some other user defined values. The format of the parameters and return values passed is JSON format, according to the Swagger 2.0 specification. These paths and JSON parameters are used when communicating externally with other whiteboard-compatible tools, such as mobile MDS-library, or PC-tool wbcmd.exe.

Here is an example of an API subscription defined in applications yaml-file:

```
  paths:
    /ExampleResource/Subscription:
      post:
        description: |
          Subscribe to RR values.
        responses:
          200:
            description: Operation completed successfully
          x-std-errors:
            description:   See    common    error    codes    http://devel-
oper.suunto.com/api/std-errors#subscribe
          x-notification:
            description: New value
            schema:
              $ref: '#/definitions/LastRRValue'
      delete:
        description: |
          USubscribe from RR values.
        responses:
          200:
            description: Operation completed successfully
          x-std-errors:
            description:  See  common  error  codes  http://devel-
oper.suunto.com/api/std-errors#unsubscribe
  definitions:
    LastRRValue:
      required:
        - lastRR
      properties:
        lastRR:
          description: RR
          type: integer
          format: uint16
          x-unit: ms
          minimum: 240    # equals in HR 250 bpm
          maximum: 60000  # equals in HR 1 bpm
```

After adding this resource to a list of providers in the Movesense provider application, a corresponding C++ *resources.h* file will be generated at the time of building the project. The generator file itself is included within the pre-compiled binary and is therefore not possible to be inspected. Regarding to subscriptions though, there is also another generator used, of which's Python script file is available within the MovesenseCoreLib. Inspecting this script's outputs might sometimes give some additional insight into debugging. Generated serializations are called *sbem-definitions*. They are used by the whiteboard when someone subscribes to that specific resource, or when */mem/datalogger* API is called to save subscriptable resource's notifications into EEPROM. Subscribing to the defined resource can be done e.g. with wbcmd-tool on PC, with the following command:

```
wbcmd --port COM3 --path /ExampleResource/Subscription --op subscribe
```

If fully implemented, this would result in a JSON answer in wbcmd-tool, like this (here instead an example of GET request):

```
 wbcmd --port COM5 --path /fb/mem/configuration
WbCmd Movesense:
{
  "response": 200,
  "responsestring": "HTTP_CODE_OK",
  "operation": "get",
  "uri": "/net/184030000281/fb/mem/configuration",
  "content": {
    "movementLevel": 3
  },
  "querytimems": 48,
  "querytimens": 48895614
}
```

From within the C++ code, a module can subscribe to a resource, such as `/ExampleResource/Subscription` e.g. with the following method:

```
   whiteboard::Result   subscription_result   =   asyncSubscribe(WB_RES::
LOCAL::EXAMPLERESOURCE(), AsyncRequestOptions::Empty);
```

This asyncSubscribe() -method will call onSubscribe() method of the module implementing the resource. Response to asyncSubscribe() will be given with a method returnResult(). Eventually returning from onSubscribe() causes a callback into listening modules onSubscribeResult(). This is where we know that the original call has finally finished. Subscriptions themselves are to allow a provider to continuously send values to a client which has subscribed to that resource. The actual notifications of subscription are sent to subscribed clients with method updateResource(), and clients then receive these notifications into onNotify() method.

When calling resources with async-call, it needs to be also noted that by default those calls will be synchronic (if the resource is in the same context as the caller), if not otherwise defined as asynchronic with AsyncRequestOptions.

3.11 Movesense challenges with modularity

Whiteboard system in Movesense relies heavily on modularity, where most of the APIs offered by MovesenseCoreLib are optional modules which can be included or excluded from the project. This is true also when developing a user application - it is implemented as an inherited class and included into the whiteboard through macros in a file *App.cpp*. The system abstracts away the actual instantiation of the modules. If user needs to have multiple modules or classes, this might lead to challenges if information would need to be shared between these modules or class instances. Whiteboard's own system's APIs can be used to exchange data between classes internally, but it also adds an extra layer of need to define each datatype or struct as its own yaml-resource path. Therefore, other possible design patterns might be needed to manage communications between multiple classes, if defining separate REST APIs for transferrable data is not feasible. Possible design patterns could include e.g. singleton-classes or injecting some code within the *MOVESENSE_PROVIDERS_BEGIN()* section, to get the instance pointers of desired classes.

3.12 Application development for team sports

As it has been discussed before, the project needs for a new team sports sensor came from the business of Firstbeat Technologies Oy, to meet the new requirements of customers. There was a need for a new sensor system, where analytics could be run within the sensor and broadcasted in real-time to a coach's client device. The new system should also have internal memory for offline training and BLE to get rid of a separate transceiver unit.

Prior to starting this thesis work, Suunto Movesense sensor was chosen as the main candidate for the new product. The choice had been made based on prior evaluation of suitability of BLE for team sports monitoring systems (Portaankorva, 2018) and because the sensor seemed to meet all the requirements and targets set by the customers and business. As part of this thesis, further evaluation of the Movesense sensor was performed. As the evaluation and development of the system evolved, the Movesense sensor was finally chosen as the choice of technology for the next generation team sports sensor.

In the following sections the most important use cases for a new team sports monitoring system are provided. These requirements came from the business and product owner and were further discussed and refined in the development team. Based on the use cases, more technical requirements such as REST APIs, BLE adv packet formats, state logic and data savings were specified. A brief description of the final system has been provided. Debugging and development processes have also been briefly discussed in their own section.

3.12.1 Specification and implementation

Use cases are often told in a form of a story describing a real-life situation and conditions where a device will be used. The following story presents one of the most typical use cases:

"Football team has a practice at 10 am. Coach arrives to the field 15 minutes earlier to prepare for the training session, including preparation of the monitoring system. As athletes arrive, they'll pick up their own sensor belt and put it on. Most of the players moisten the belt and adjust it appropriately. As the belts are put on, coach sees the broadcasted data on the client application, such as iPad. During the training coach keeps the iPad with him to follow the training metrics of individuals and whole team. Part of the time he leaves the iPad to a side, to take part of the practice. After the exercise athletes leave their belts to a bag next to the field. Coach initiated uploading of data from sensors to the cloud via the client application and goes to collect training equipment from the field. After upload is done, coach and each athlete receive their individual report of the training. Coach can now start analysing the whole team's data." – *Typical use case*

As this was only an idealistic typical use case, various exceptions need to be considered, e.g. the following:

"Some players arrive 15 minutes late, pickup their belt, put it on and participate in the training. The belt should then appear in the client application." – *Late coming*

"Some players need to leaver 30 minutes earlier. They want to also record some training at home, so they need to take their belts with them. Before leaving, coach uploads their training data to the client device. At the next time when coach loads the data from the sensors, the home trainings are uploaded at the same time with team trainings." – *Early leaving and training at home*

"During training, an individual player is given a permission from coach to connect the HR sensor to his smart watch. This prevents the sensor from broadcasting the data to the client device but allows the athlete to focus on specific parts of the training himself by checking his watch." – *Smart watch connectivity*

"Coach is not attending the beginning of the training, but athletes begin the training session by themselves. They put their belts on and start training as they normally would. As coach arrives to the side of the field with his iPad, he sees the real-time training metrics of the individual athletes and team as if he'd been there the whole time." – *Unsupervised training or coach attending late*

"Training went too long, and coach has no time to upload the training data from the belts. Coach will load the data from the sensors later in the evening." – *Uploading done later*

While this was not by any means a comprehensive listing of all the use-cases possible and specified, it works as an introduction to further present and justify the technical requirements which were set for the final team sports sensor product. The following requirements and specification were set already knowing that Movesense sensor would be utilized.

A brief listing of the logical and technical requirements for the product:

- Idle mode and measurement state logic
    - When sensor is not used, it should stay in an idle mode to save battery

  - Sensor needs to automatically detect when it is on the skin. When skin contact or impedance change in electrode leads is detected, sensor should power on.
  - When sensor is powered on, it should validate ECG signal to start recording RRI signal and run training analytics.
  - Recorded RRI signal and appropriate metadata should be saved into the internal memory of the sensor.
  - Sensor needs to detect when it is taken off the skin after training and stop recording.
  - Sensor should not stop recording during training, even if it is momentarily misplaced.
  - Sensor should not start recording if not on skin, e.g. in bag.
  - After recording has stopped, sensor should allow enough time for the client to connect and load the data before returning to idle state.
- Broadcasting logic
  - When sensor is not in an idle mode or after skin contact is detected, it should always broadcast BLE advertising packets with some specified information.
  - During measurement the advertisement packets should include all the necessary and optional training metrics such as training duration, HR, EPOC, HR zones and TRIMP, etc.
  - Each advertisement packet should include all the required information for client to decode it, without prior knowledge of the previous packets received or the state of the training.
  - When sensor is turned on, but no skin contact is detected, optional information about the device such as the number of measurements recorded should be broadcasted. This shouldn’t however be a requirement for any use cases.
  - Sensor should provide standard Bluetooth HR service connectivity for smart watches.

- Connectivity logic
  - Client application should be able to pair or connect with any sensor at any time, whilst they are broadcasting advertisement packets.
  - Possible on-going recording shouldn't be interfered by a connection with a client device.
  - Appropriate REST APIs should provide additional information, allow controlling measurement states, loading of data and configuring the sensor.
  - If no new recordings are to be made after disconnection, sensor should go to idle state soon after.
- Power consumption
  - Battery of the sensor should endure for 4 months minimum, with 10 hours of training per week.
- Upgrading firmware should be possible with OTA-updates.

Though the typical use cases of the product remained the same since the beginning of the project, quite a few adjustments and details were added later, and not all the final requirements are included here. However, Figure 18 presents the underlying state machine, by which the finalized sensor software works. The typical use case follows the following flow: HR strap is put on, sensor detects the leads and goes to a *startup*-mode, signal is validated, and sensor starts recording RRI during the training. After sensor stops receiving RR-intervals, it will go to a *waiting* mode to wait until the recorded data is downloaded. Finally, sensor returns to *idle* state. Other use cases were also considered, and the state machine was fine tuned to be suitable for them as well. Additionally, a *wet belt detection* had to be added as a check when going to idle mode. This was due to tests showing that Maxim30003 AFE chip responsible for *leads-on detection*, recognises an extremely wet or sweaty belt to be in a *leads-on state* (i.e. attached to body), even when it is not. This is due to reduced impedance between electrodes in the textile heart rate belt. As a workaround for detected wet belt condition, a low frequency movement detection is used instead to return from *idle* state.

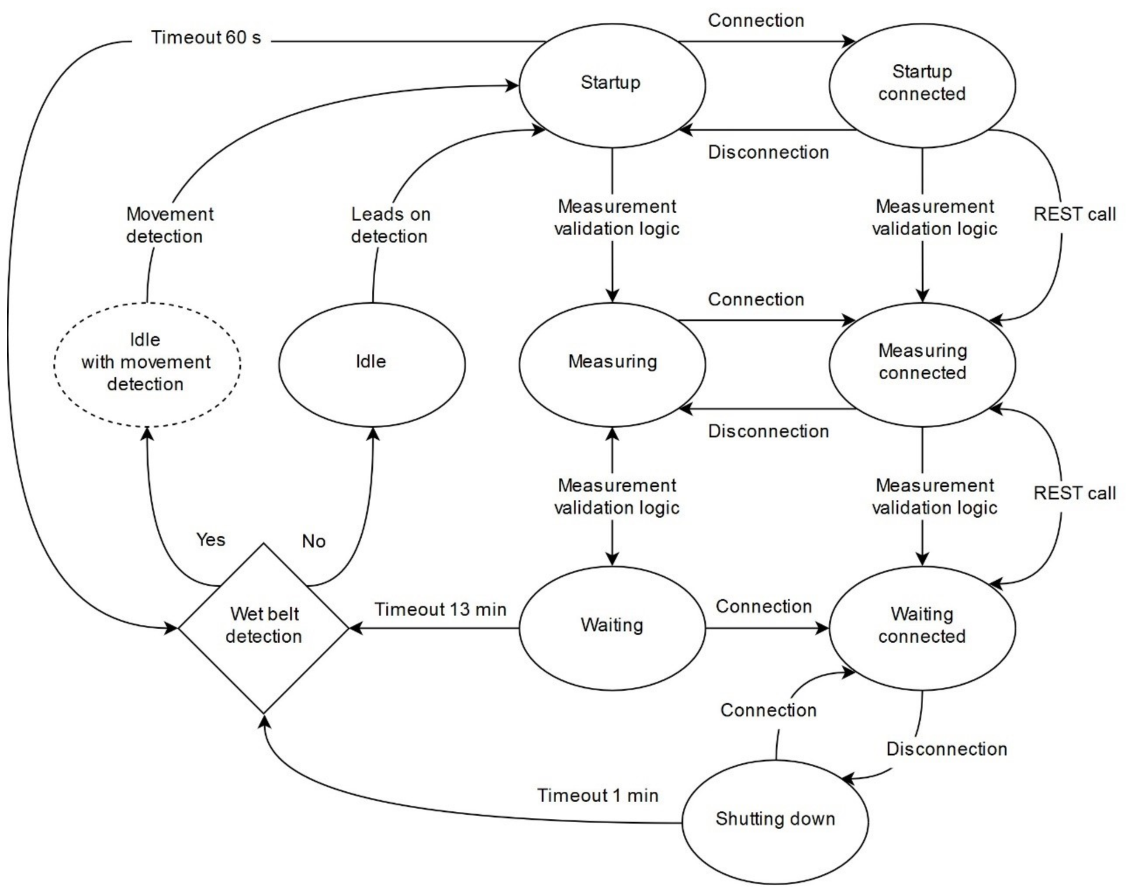


***Figure 18** State machine of developed sensor application.*

Results of evaluation included later in chapter 4, present another HW issue which had to be resolved during the development – rare HW resets. These were concluded to occur from battery connector wearing and loosening, or possibly from the use of jig, which might press the sensor too hard when used, mechanically deforming the sensor. Whatever the source of issue, a resolution had to be made. Therefore, a system recovery was designed to be performed whenever a reset would occur during measuring state. This allows continuation of measurement and ETE analytics as they were before the reset. This is done by frequently updating required recovery data to EEPROM, and then recovering that data back from EEPROM in case of a rare HW reset. These steps are, however, not presented in Figure 18.

Overall, the state machine was designed to be simple, customizable and to meet the needs of the use cases listed. Most importantly the state changes committed by measurement validation logic needed to be robust in detecting when to record. In unclear cases the validation should also be biased to rather recording too easily, than to leave out any training data. This was one of the greatest challenges and its implementation will be discussed further in the next section.

3.12.2 Measurement validation logic

Measurement mode validation plays a crucial role in any wearable HR measurement system, where we simply cannot record all the time due to memory and battery limitations.

Suunto Movesense sensor offers three possible ways to measure RR-intervals:

1. an API for pre-processed HR and RR-interval (*/meas/hr*)
2. A licensable proprietary ECGRR-module
3. A REST API */meas/ecg* for measuring the ECG yourself, to run your own R-peak detection algorithm.

The easiest and cheapest way is to use the readily available */meas/hr* API, which is enough for most use cases. In this case the RR-intervals come from the Maxim30003 AFE chip itself, with a detection resolution of 8 ms. As this is not accurate enough for recovery tests (such as QRT by Firstbeat), it could not be used in this product. Instead the ECGRR-module was used and modified in collaboration with Suunto.

ECGRR-module provides a modified version of the generally well known PanTompkins algorithm for R-peak detection. Some modifications have been made to adjust the algorithm for sports use cases, as the original algorithm makes some assumptions of the HR due to its clinical origin. In addition to providing RRI-values, the module can also be tuned to provide once-per-second SNR (signal-to-noise-ratio) values. The SNR estimates calculated by the module are derived from the signal by using some assumptions about the noise, standard deviation and signal amplitude, using one second time window. Final sensor application uses SNR and MVA (moving average) of RRI in its measurement validation logic, to detect when the sensor should start and stop measuring.

Multiple versions of start and stop logic were trialled and tested during the development of the sensor application. Table 5 briefly presents the main tested versions of validation algorithms.

Table 5 *Pros and cons of different RRI measurement validation algorithms.*

Algorithm	Pros	Cons / issues
MVA of RRI from Maxim AFE (24.9.2018)	Robust and simple. If there are no RRI values, there is no point in recording anything.	Maxim AFE has low resolution. Maxim AFE does not provide any signal quality metrics, such as SNR.
MVA of RRI from ECGRR-module (19.11.2018)	Sufficient level of accuracy.	Tuning of the ECGRR-module took time. Initialisation of module was slow, and the module was too sensitive (detect RRI when there's no signal e.g. on table)
SNR from ECGRR-module (4.3.2019)	SNR is a good measure of general quality of the signal. Clear distinction in most cases of belt-off can be seen.	SNR estimate can drop low if there is much movement artefact in the signal - even if the ECGRR would still detect correct R-peaks.
SNR + MVA of RRI from ECGRR-module (10.4.2019)	This combines the pros of both SNR and RRI MVA -> we can evaluate the snr quality, but also know to stop if there is no RRI coming	No issues have been found.

3.12.3 ETE-library integration

The possibility of integrating the physiological analysis library ETE directly within the Movesense sensor, was one of the most important reasons for choosing the Movesense platform. Integration of the ETE binary was a straightforward process due to easiness of CMake’s capabilities in adding a new compiled library into the build. The developed application with ETE inside allows running advanced analytics directly within the sensor, as athletes are training in the field. Lost BLE packets are not an issue, as each BLE packet has all the information needed for the client to decode the data, to provide real-time analytics of each athlete’s training.

The included ETE analytics in the final application are:

- EPOC
- TRIMP
- TRIMP/min (TRIMP accumulation rate per minute)
- Aerobic and anaerobic training effect
- Lowest, average and highest corrected HR of training
- Times in HR zones
- Cumulative energy expenditure

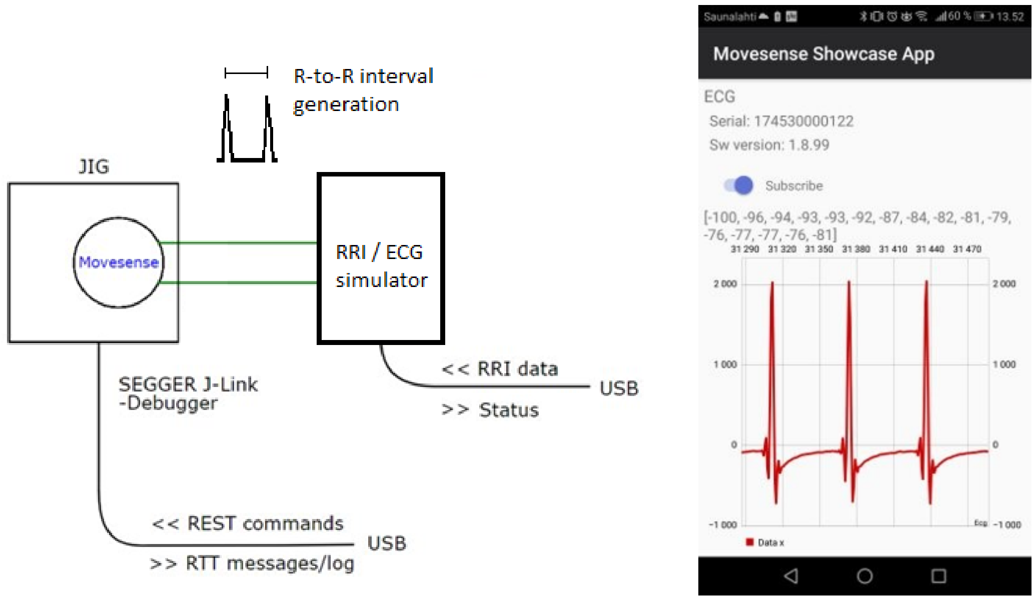
The developed sensor with the new TRIMP/min, is first of its kind as no other competitor in the markets provide TRIMP/min for real-time analytics during a training. With the feature, coaches can more easily recognise and track the fast changes in athlete's internal loads. (Firstbeat Technologies Ltd., 2019)

3.12.4 Development and offline testing

Testing and debugging tools and capabilities are an essential part of any software development process to ensure a correct functioning of an embedded device. In the case of Movesense sensor, there are several different options: flashing and debugging jig, PC simulator for offline testing of an application, using REST APIs over BLE (Android and iOS), and a separate REST API *System/Debug/Log* for logging of debug-messages into the non-volatile EEPROM memory. During the development of this new team sports application, all these options were explored and used. Due to the nature of the application under development, an additional testing tool, an ECG-signal simulator, was also developed. In this chapter, a brief coverage of these will be provided.

Movesense jig comes with Segger's J-Link base model debugger probe, which can be used for both flashing and debugging. Segger's proprietary RTT-debugging feature has been included within MovesenseCoreLib (SEGGER Microcontroller GmbH, 2019). This allows an invasive, but fast output of debug prints via JTAG. In the background, each time when debug prints are printed via RTT, *memcpy()* function is called to copy the message to RTT's buffer. This buffer is then non-invasively emptied via JTAG to PC with e.g. RTT viewer. MovesenseCoreLib's debugging print methods such as *DEBUGLOG()* macro or *DebugLogger::error()* have been provided to enable the use of RTT for debugging. Traditional halting of CPU and stepping through application is also possible, but this will often cause the application to eventually assert, as this will break SoftDevice's real-time requirements. (Nordic Semiconductor, 2015)

In addition to using debugging tools provided with the Movesense system, hardware-in-loop (HIL) simulation platform was designed to test and validate the final application. The designed platform includes a Movesense sensor connected to PC via jig, and an RRI-signal-generator for generating R-peak signals directly to the sensor. This enables repeating and testing of various situations, such as belt on and off conditions, as well as running of long real-time training sessions. Connecting the same generated signal source to multiple Movesense sensors simultaneously, also allows simulating a same training with e.g. a whole team. A block diagram of the developed generator is presented in Figure 19, together with a resulted ECG-looking signal. The idea of the generator was not to generate real ECG, but only the R-peaks to make the sensor record RRI values.



[-100, -96, -94, -93, -93, -92, -87, -84, -82, -81, -79, -76, -77, -77, -76, -81]

***Figure 19** Block diagram of RRI-signal-generator (left) and resulted signal as visualised with Android Movesense Showcase App (real-time ECG streaming) (right).*

In the future development it has been planned to automate building and testing of the sensor software by implementing a continuous integration (CI) pipeline. The current Suunto's bitbucket uses version control software Git. In Git, when doing a commit into a development branch an automatic building and testing could be triggered in another lab-PC. There automated scripts would be run to test and validate different use-cases together with the HIL simulation. Test reports could be automatically generated. This CI pipeline will enable more efficient and accurate testing. Therefore, the RRI-signal-generator is the first step towards this more advanced testing and development environment.

4. SYSTEM EVALUATION

Software development is most often in deep connection with testing of the software. In modern agile development methods, the project evolves in an iterative manner, small continuous progress and improvements being done all the time, corresponding to the results of testing and changing product requirements. (Larman & Basili, 2003) Throughout the development of the sensor software, multiple iterations have been gone through. Testing has included e.g. manual testing on jig, manual testing with iPad, unit testing on PC simulator, testing with hardware-in-loop simulation on jig, and several field tests. Testing has been done by several people in the development team.

One of the main focuses of the product development has been the robustness of the measurement validation algorithm, to never miss recording a training, whilst at the same time trying not to consume too much extra power or memory. Another important focus has been to define and provide all the needed REST APIs for controlling the sensor and measurements, as to cover all the use cases.

This chapter here comprises the most important and significant results of the test reports. The following evaluation reports will be presented in a chronological order, to give insight into the development process and challenges that arose. Therefore, field test results will be presented first as they have been an on-going aid to the development process. They are also used to evaluate the fulfilling of use cases and set requirements. Following that, one controlled field test will be presented, which was used to evaluate Movesense's R-peak detection. Finally, power consumption and memory usage of the latest version of the developed application will be presented.

4.1 Supervised team field test results

Throughout the development project of the new sensor product, continuous and frequent field testing has been performed to aid the development process, test the sensor against real use cases and evaluate the product overall. The results of these on-going tests have been fruitful and very much useful, as they have revealed issues which might not have been found otherwise in plain software testing. A physical sensor is more than just a software inside, and therefore a thorough testing needs to be performed to test also the mechanical, physiological, electrical and usability issues in addition to software issues.

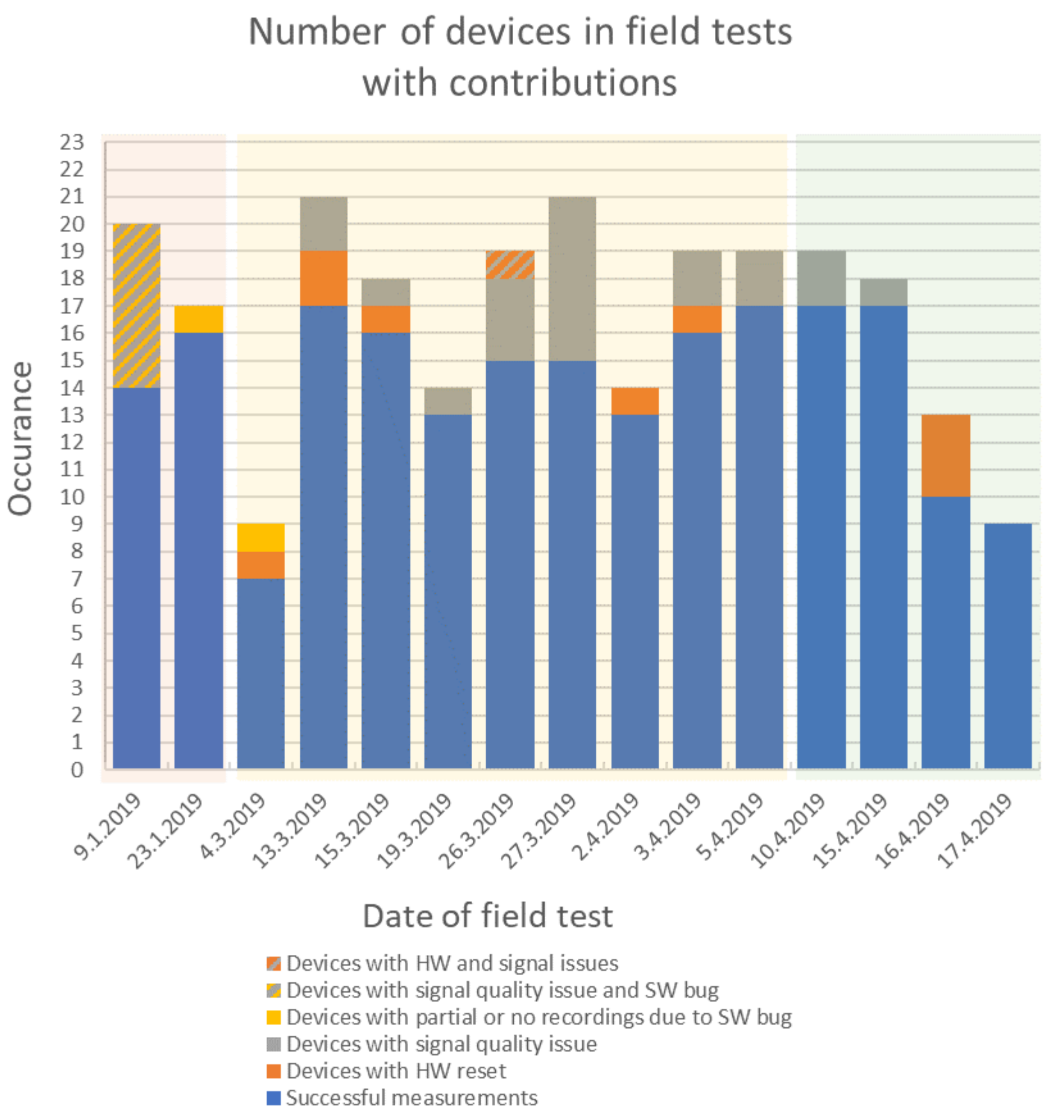


Figure 20 *Bar graph presenting total number of devices in each test, with number of issues indicated. Background colour separates different versions of RRI validator.*

All the field tests reported in this chapter were run with amateur or semi-professional sport teams as part of their normal training, with ranging number of participants at each field test. Figure 20 summarises the results of field tests, which were done during 2019. The figure presents successful measurements together with those which had issues. The issues have been divided into three main categories: HW resets, signal quality issues and SW bugs. HW reset means that a sensor reset itself due to mechanical connection issue in sensor battery’s enclosing. SW bugs of the application originated either from MovesenseCoreLib, or from an application bug. Signal quality issues originate from a sensor receiving poor signal, which caused measurement validation logic of the application to e.g. stop recording early, or start late. However, as discussed in chapter

3.6.2, many of these issues were tackled by improving the measurement validation logic. Based on the field test reports, the measurement validation logic was developed to be robust and less sensitive to occasionally occuring bad quality signal, as much as possible. Success rates of the measurement validation logic, without taking into account other issues, have been presented in Figure 21. These results present how many percentages of devices measured the whole training without stopping or restarting in the middle of the training. As can be seen, the final version has improved a lot since earlier versions, and can be esteemed to be working well. In these field tests with the final measurement validation logic, mean success rate of 95% was achieved.

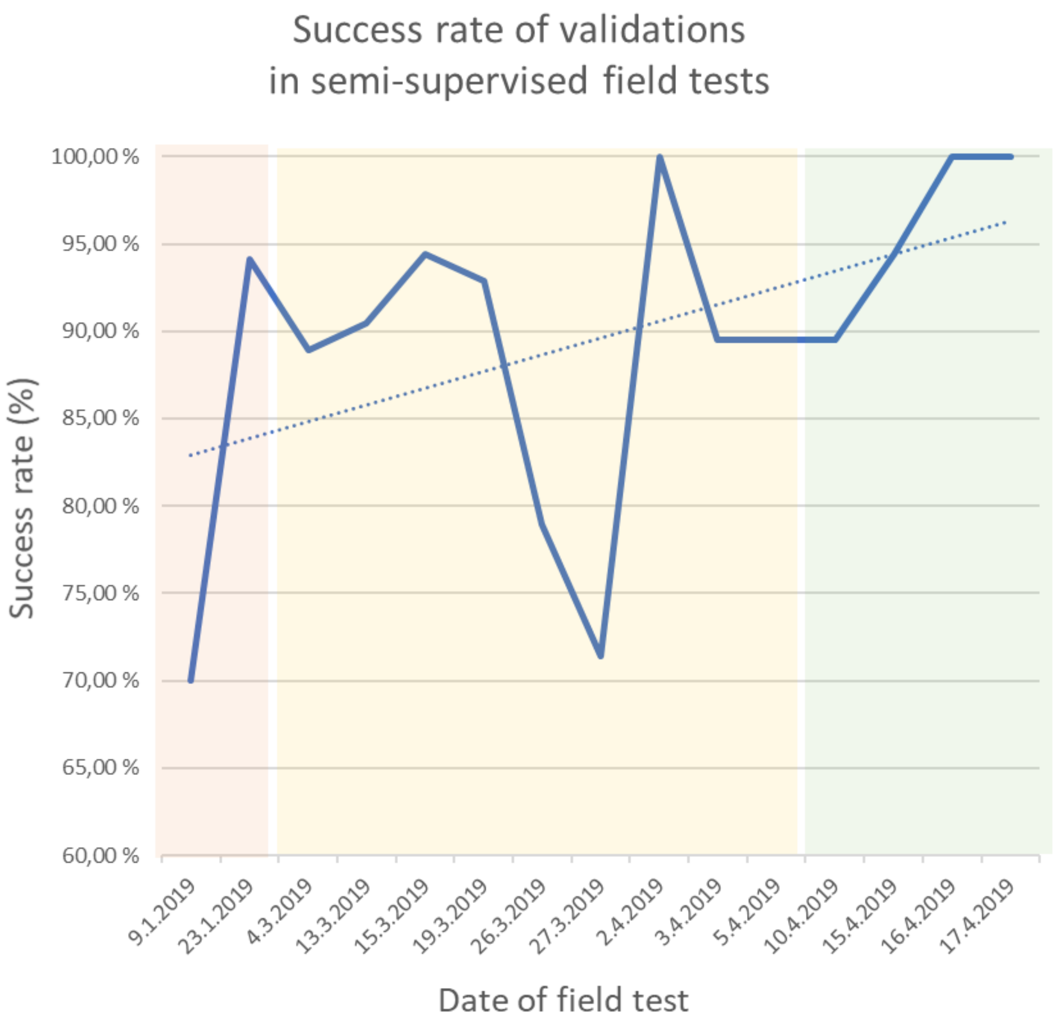


Figure 21 *Success rate of measurement validation logic in field tests. Background colour indicates different versions of RRI validator.*

To understand why 100% success rate is not always achieved, we need to understand the real-life use cases where the measurements are done. In these field tests, athletes' level of experience using HR belts varied. Also some of the athletes wet their belts as instructed, and some did not, also affecting the results. Other issues include e.g. belt worn upside down, too loose belt, or wrong placement of belt. Therefore, the results presented here are not only affected by the movesense sensor and its application, but also by these unkown variables. If we only consider the cases without user errors or issues with signal quality, we can conclude the success rate of the final measurement validation logic to be 100%. This is because the source of the falsely ended measurements is generated from the lack of RRI provided by the ECGRR-module, and by definition if there are no RRI values to be recorded, there is no need to keep recording.

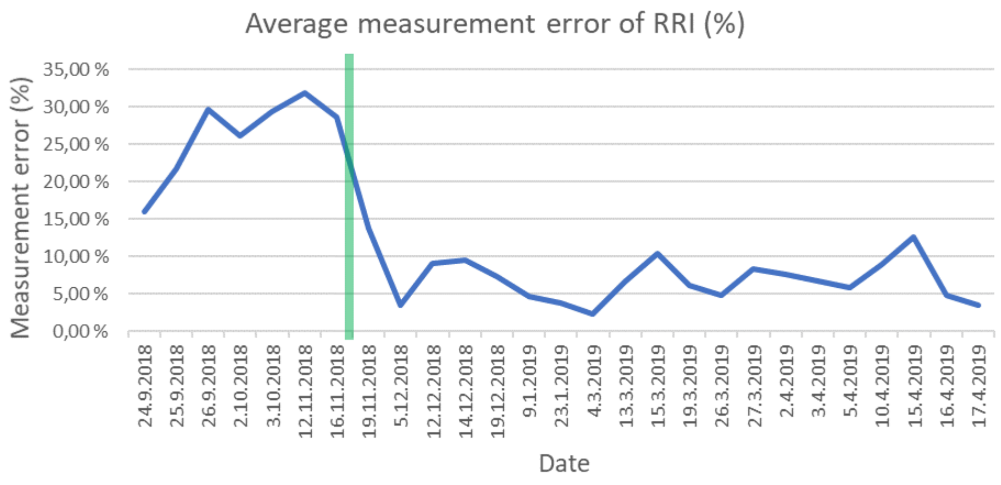
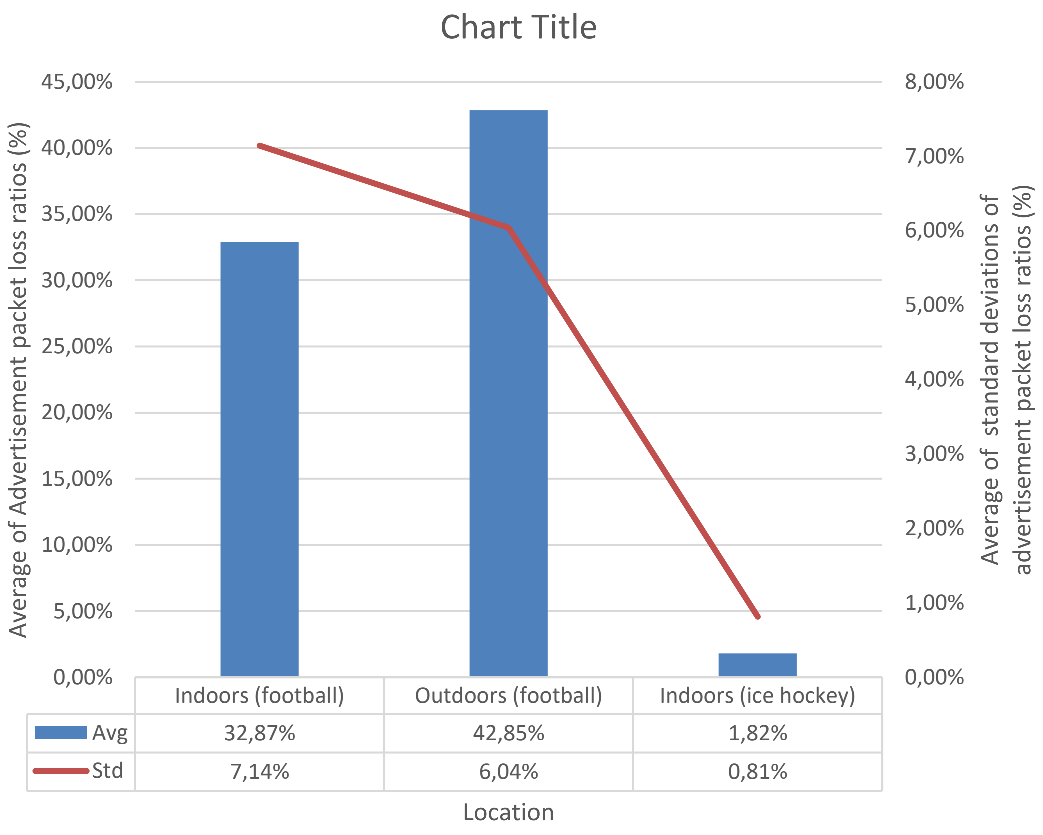


Figure 22 *Average measurement errors of RRI by field test dates. Changing from using of Maxim AFE RRI, to ECGRR-module's RRI is indicated with a green line.*

As discussed in chapter 3, the native RRI signal from Maxim AFE chip within Movesense sensor were not accurate enough for the product which was developed. For this reason a proprietary ECGRR-module was used and tuned for this application. The supreriority of the ECGRR-module's accuracy regarding to RRI quality can be seen in Figure 22. It shows a significant drop in error rate after changing to use the module. The error rate has been calculated by the properiatery ETE algorithm library.



	Indoors (football)	Outdoors (football)	Indoors (ice hockey)
🟦 Avg	32,87%	42,85%	1,82%
🟥 Std	7,14%	6,04%	0,81%

***Figure 23** Impact of different locations to advertisement packet loss ratio.*

As the teams sports monitoring system can be used both for real-time monitoring of teams and for offline training (data being loaded from the sensor later), it is important to also benchmark the performance of “how real-time the data is?” For this purpose, different use case locations were compared. Figure 23 presents averages of different advertisement packet loss ratios by location. The loss ratio was defined as how many non-duplicate packets were lost (the content of packets change once per second). As the field tests were not run within controlled environment, the radio environments could vary as well, even if the place would be the same. Also the number of devices changed between field tests, making the field tests not directly comparable. Therefore, the averages presented in the figure are derived from the means of each field tests by location.

Number of field tests per location are 13 (indoors, football), 12 (outdoors, football) and 2 (indoors, ice hockey. From the results we can see that there is a massive difference in packet loss ratios in ice hockey and football. This can be explained by the game field size in these two sports. The longer distances in football result in weaker RSSI of the signal, and higher rate of packet losses. Comparing the two indoors locations (ice hockey versus football), we can see that merely having reflective walls and roof around the field do not have that much of an effect as the size of the field. Of course it needs to be noted that the number of field tests between ice hockey and football are not directly comparable. However, as the lowest packet loss ratios of football were 20,14% (inside) and 23,02 % (outside), and the highest loss ratio of ice hockey was only 3,09%, we can safely assume that the differences of packet loss ratios are quite significant between these two sports.

Overall, considering the results from both football locations, we can derive the conclusion that playing indoors in comparison to playing outdoors, can actually improve the packet loss ratios – in this case by around 10%.

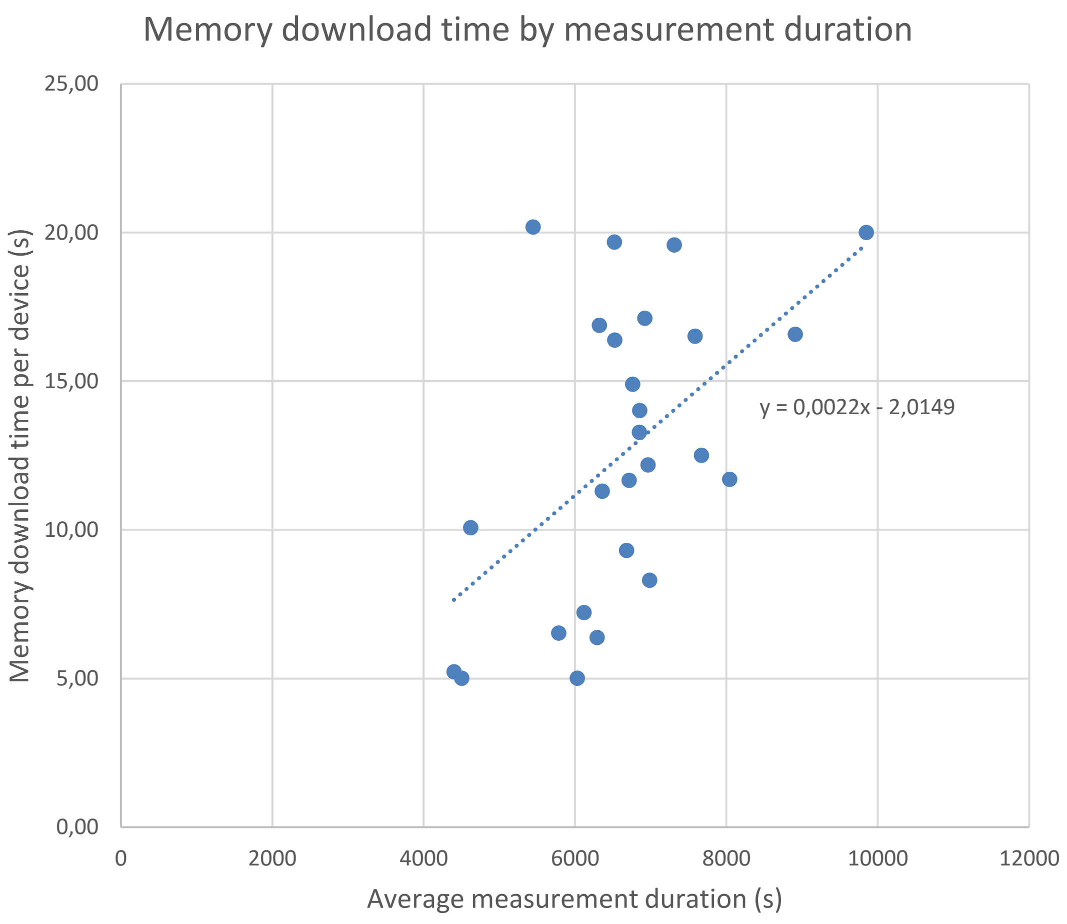


Figure 24 *Download times of training data per device in all field tests, by duration of logged measurement data.*

In typical use cases, the coach uses the client device to download all the measurements from the sensors. The time that it takes to connect to each sensor and perform the downloading depends on different matters: how much data is being downloaded, how many devices the client can connect to in parallel, radio traffic and other interfering signals. For estimating probable download time of a measurement with certain length, the download times were taken from the field test reports, and are presented in Figure 24. The variance between different field tests can be easily seen, as with the same measurement duration the download time has been varying by about 15 seconds at worst. However there is a clear trend, which predicts average download times of

$$\boldsymbol{y = 0{,}002x - 2{,}0149,} \qquad (2)$$

where *y* is the expected download time in seconds per device and x is the average duration of a training. For example, an average total download time of a two-hour training for 9 devices would be 111 seconds. Nine devices were used here as an example, as it was the maximum number of downloads done at once in the field tests, due to a limitation of a field test iPad software.

4.2 Controlled field test results

The supervised field tests were originally initiated more as a general use-case testing setup or as an asset to help the software development, rather than a controlled performance bench marker. From the results of the previous chapter we saw that these field tests could not provide reliable enough performance evaluation in all cases, as there were too many unknown variables that played a role. These unknowns included e.g. mis-attached belts, not following directions in wetting the belt, and having too loose belts. Also, the same belts had been used for quite a while, which might have added some noise to the signal in latter measurements. Therefore, one more controlled company internal test was performed, to better evaluate the performance of the measurement validation logic.

Quality analysis of the RRI signal was also done as part of the controlled test. This was done by having the participants wear a reference RRI recorder at the same time with Movesense sensor. The reference sensor was a wearable high-quality ECG/RRI recorder Bodyguard 2 (BG2), which was presented in chapter 2.3.1. It has been also used in scientific studies prior, due to its easiness of use and high accuracy (Parak & Korhonen, n.d.). However, BG2 has not been designed with the focus of sports use cases only, but rather for recording overall everyday activities of an individual. It has two leads which are attached to the body with medical grade wet gel electrodes.

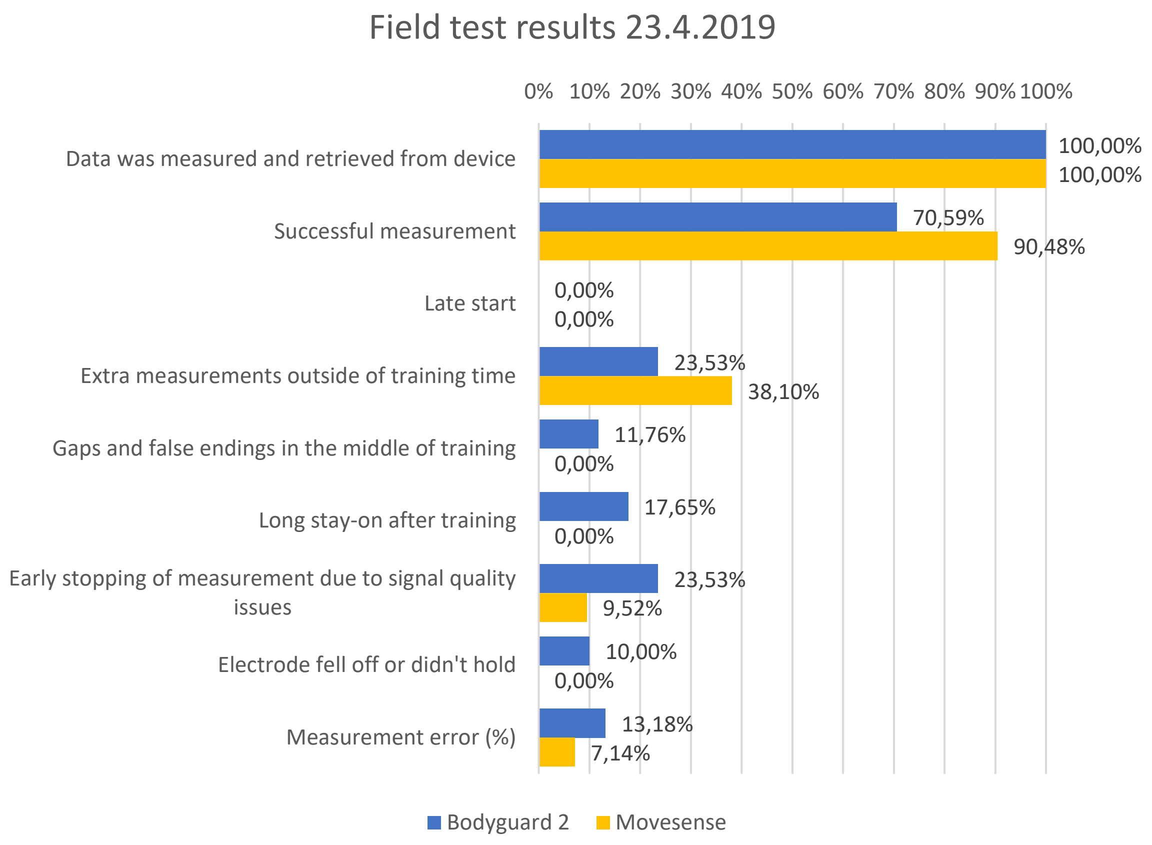


***Figure 25** Occurrences of issues present at the controlled field test with BG2 and Movesense sensor.*

Success rate, issues found during training, and measurement errors for both Movesense and BG2 in controlled field test are presented in Figure 25. Altogether there were 22 players, with 21 of them participating in the study. Unfortunately, 3 of the reference devices were unfunctional, recording no data. Also, one player only wore Movesense sensor. Therefore, 17 reference devices, and 21 Movesense sensors have been considered in these comparisons. Measurement error (%) was constrained to one hour, if a measurement stayed on after the training ended, or to the duration of the measurement if sensor fell off or stopped recording earlier. The field test begun by distributing the devices and guiding the players to wear them properly. All the HR straps' electrodes were sprinkled with water before the game. The training lasted for approximately an hour and had a 3-minute quick recovery test (QRT) before and after the game, were players laid still on the ground. During the whole test, advertised BLE packets were monitored. One Movesense device had a hardware reset. During the training one player changed BG2's electrodes due to loosened attachment due to sweating and another player took their sensor off for the same reason. Some players also took the sensors off before the QRT, due to bad holding of wet gel electrodes.

After the training and QRT, all the devices were collected into a bag, without taking Movesense sensors off from the belts. This is against directions of Movesense's manufacturer Suunto, as it can lead to false measurements. However, this was tested to see how many of the devices would have this issue. Both Movesense and BG2 had these issues, with Movesense having 38% and BG2 having 23% of the devices turning on after the exercise. It must be noted though, that as BG2 has been designed for 24-hour measuring, it is very desired that it should rather be always-on than miss any recordings. As discussed earlier, false positive measurements are not that much of an issue for Movesense sensor either. They only result in short recordings, which don't take much memory. They can also be automatically separated from the real measurements in the client device. A slight impact on the additional power consumption could be expected though, if these false positive measurements were to happen regularly.

Overall success of the measurements as presented in Figure 25 has been calculated from the devices, which didn't have any major gaps in the middle of the training, or which did not end the measurement too early. The one device with HW reset was counted as non-successful. We can see that Movesense sensor was more reliable in this test than BG2. Movesense also had better measurement error rate (7,1%) than the reference device (13,1%). Excluding the Movesense device with HW reset, there was only one Movesense which did not record the whole game without gaps. With this device and player possible reasons for the recording restarting during the game were inspected immediately as the recording stopped. It was concluded that the stopping of the recording was caused by sensor not detecting any R-peaks due to poor ECG signal. Worsening of the SNR was most likely a result of heavy sweating of the player combined with wearing a very tight synthetic looking shirt. Most likely this formed a slightly conductive path between the belt's electrodes, resulting in short-circuiting the electrodes at some level. This was suspected, as in some cases the belt has been reported not to have an absolute insulation between the belt's textile and the electrodes. This leads to some current flowing through a wet belt. Therefore, the recorded ECG is weaker as the signal degrades. This seems to be a rare occasion and a combination of some unknown factors which would need some further investigation. Also, this might be an explanation to some of the restarting and stopping of recordings in other field tests as well.

One objective of the controlled field test was to measure and analyse the accuracy of Movesense sensor's R-peak-detection. This was performed as part of a Quick recovery test (QRT), which is one of the recovery features of ETE-library, by Firstbeat. QRT is performed by laying down and staying still for approximately three minutes. In this controlled field test, QRT was done before and after the game. These recordings from both reference device BG2 and Movesense sensor were then synced and compared with different statistical error metrics. These results are presented in Table 6.

***Table 6** Mean values of error metrics for all valid QRT measurements.*

Measurement (approx. 3 min)	**ME (ms)**	**STD (ms)**	**MAE (ms)**	**MAPE (%)**	**MAD (ms)**	**Num. of measurements**
QRT before game	0,008	0,596	0,413	0,056	0,302	11
QRT after game	-0,005	0,833	0,491	0,076	0,301	7
Total	0,003	0,688	0,443	0,063	0,302	18

Purpose of this analysis was to evaluate the accuracy of Movesense's R-peak-detection, rather than evaluation of the whole system's total measurement error. Therefore, all the measurements with clearly visible artefacts or gaps were removed from this study. Figure 26 shows an example of a typical error signal between BG2's and Movesense's recordings, as well as a disqualified measurement with visible artefacts. Altogether 18 QRT measurements were analysed as part of this study.

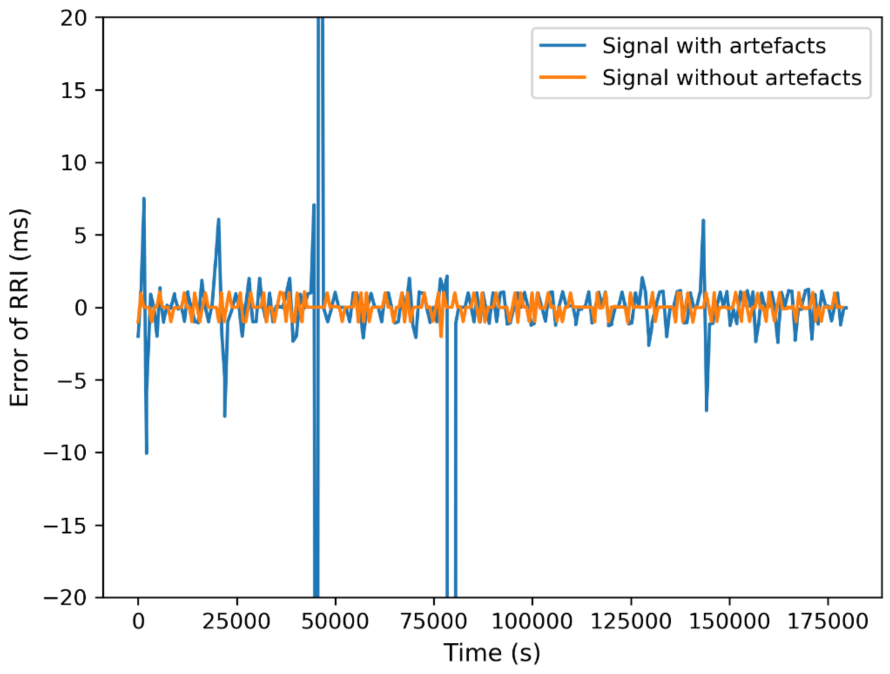


***Figure 26** A typical error signal from QRT measurements with Movesense (orange) and a disqualified measurement with artefacts (blue).*

The total results in Table 6 indicate very low error rates for the Movesense R-peak-detection. Mean error (ME) and mean absolute error (MAE) are well below 1 ms and median absolute percentage error (MAPE) is only 0,063 %. Also, the median absolute deviation (MAD) is only 0,302 ms and the standard deviation (STD) 0,688 ms. These results indicate that when there are no major artefacts present, R-peak detection of Movesense is well comparable with R-peak detection of reference device BG2. It needs be reminded that in the developed application the R-peak detection comes from the Suunto's licensed ECGRR-module instead of the native */meas/hr* service.

A prior hypothesis before doing the QRT measurements was such, that the signal quality of the Movesense electrodes would improve during training, as the electrodes would warm and get sweaty. However, as it was discussed earlier in this chapter, high sweating and wetting of belt might instead degrade the signal quality. To examine this subject further, Table 7 includes only those sensors which were successful and without any visible artefact in both QRT measurements before and after the game. Therefore, the sensors and participants in the table are the same before and after. This minimizes the affection of environmental matters and individual differences and should give better estimate of whether the signal quality was better prior to game or after.

Table 7 *Mean values of error metrics for same devices between before and after -tests.*

Measurement (approx. 3 min)	**ME (ms)**	**STD (ms)**	**MAE (ms)**	**MAPE (%)**	**MAD (ms)**	**Num. of measurements**
QRT before game	-0,008	0,564	0,412	0,052	0,331	5
QRT after game	-0,007	0,604	0,385	0,059	0,238	5

In the table there is very little difference between the error metrics when the same subset of sensor measurements is being investigated. Also, as the number of measurements in the subset is so small, that the significance of these differences is debatable, and more measurements would be needed to conclude whether prior or after training would give better signal.

Considering all results of the test which have been discussed, indication is that the Movesense HR belt performs better than the reference device in most areas of the controlled field test. The reasons for this are many but can be summed up into one understanding: BG2 is not optimized for high intensity sports, being not the best reference for this use case. Intensive movement causes extensive stress and artefact to its hanging lead, when not properly taped onto the skin. In this test, only some of the players taped the device on their skin. Sweating can further loosen the electrodes' attachment, eventually also worsening the signal quality and possibly making the electrodes fall off from the skin. It is important to understand though, that the actual accuracy of the RRI detection is much higher and more reliable outside of sports use case, and that is why it was chosen as a reference device. This is due to BG2 having much higher sampling rate compared to Movesense. If Movesense was worn outside of sports activity, the signal would eventually degrade as the electrodes would dry without sweat. However, as was proven, in sports case where intensive movement and sweating is involved, HR belt is more reliable than bare electrode attachment. In addition, the R-peak-detection of Movesense sensor was proven to be highly accurate when compared to BG2, with less than 1 ms mean absolute error in a close-to-optimal signal.

4.3 Endnotes of evaluation of measurement validation logic

Overall, at this stage of development and with the data recorded, it can be said that the measurement validation logic is as good as it can be for this application. If there are no valid RR-intervals detected from the ECG signal, there is little reason to keep recording a training, and therefore future improvements should rather focus on other perspectives of use cases, such as instructing the athletes to use HR belts correctly or improving of the ECG signal by improving the HR belts themselves.

4.4 Power consumption

When setting requirements for a new HR sensor, battery life expectancy was one of the most important aspects. Movesense has a replaceable primary battery, which means that for a large team with 30+ players it will take some minutes to switch all the batteries to new ones. Therefore, the requirement of battery life expectancy for a sensor was specified to be at least four months. However, the design of the software was done functionality first, as the battery optimizations have been some of the last changes done to the software. Optimizing the power consumption is similar to optimizing the measurement validation: we cannot miss to power on when the training is on, but we should also avoid turning or being on when there is no need to. Therefore, all these things go hand in hand and had to be designed together.

For providing accurate estimates of the battery life expectancy of the sensor, Otii power meter (Qoitech, 2019) was used to measure the current consumptions of different software states of the software. The measurement results are provided in Table 8. These are the most significant states from the point of view of power consumption. In the calculations each training was considered to be a 2-hour session with an average HR of 150 bpm.

***Table 8** Measured averaged current consumptions of different states of the sensor.*

State	Idle	Startup	Measuring	Waiting	Downloading event
Measured current consumption (mA)	0,019	0,272	0,31	0,25	1,92

With the measured current consumptions, power estimations and battery life expectancy estimates were made. These can be found in Table 9. The calculations were done for 10 and 20 hours of training per week, having five sessions per week. With most batteries, the actual capacity of a battery can be quite far from the informed capacity. This is generally a resultant of the battery's voltage dropping due to increase in the inner resistance of a battery. Correction factor of 0,7 was used to simulate battery wearing. The results show that even with the correction factor, the designed sensor can well reach over 4 months of battery life expectancy in the most typical use case of 10h of training per week. With better quality batteries, using better correction factor, even longer times can be expected. Also, it is notable, that although power consumption during a download event is as high as 1,92mA, its contribution to the calculated life expectancies was quite insignificant, as the event duration is so short compared to the other states. For example, it was calculated that for the use case of 10h/week with the correction factor 0,7, the battery life estimate is only 0,3 weeks less when considering downloading events.

Table 9 *Estimated power consumptions and battery life expectancies for use cases with 10h and 20h training per week.*

Training	10h/week	20h/week
Average power consumption per week (mAh)	6,6	9,3
Estimated battery life (weeks): "Perfect" battery (165mAh)	25,1	17,8
Estimated battery life (weeks): Correction factor 0,7 (115,5mAh)	17,6	12,5

At the time of finalizing this thesis, not all the minor power optimizations had been finished, leaving some room for further optimization. Also, in MovesenseCoreLib 1.8.1 there is a bug which increases the power consumption to be slightly higher than it should be. Therefore, the next MovesenseCoreLib update can be expected to further decrease the power consumption.

4.5 Memory usage

As it was already discussed in chapter 3.9, resource usage in embedded devices can be very limited. Therefore, optimizations need to be often done to accomplish desired results. Even during development, there might be a need to e.g. reduce the amount of debugging and testing prints, such as RTT-messages. This happened also during this project with debug release. In the development of this project, all the unnecessary bytes and libraries (such as C++ standard libraries) were stripped off to minimize the used resources. To answer some of the questions of sensor's performance, and for documentation and evaluation purposes, some details of the resource usage of the final application are presented in Table 10.

***Table 10** Resource usage of the final application.*

Application memory used of the 46 kB space (Debug)	88%
Application memory used of the 99 kB space (Release)	53%
Worst case free stack left	1,9 kB
Heap left after application instance	7,3 kB
EEPROM space available for logbook's datalogging	392 kB
How many RRI values fit into the logbook's memory?	162 000
How many hours of training can you record with an average training HR of 150 bpm?	18h

As the table presents, we see that the resources used by the application have been appropriate. After implementing the current application, there remains still some space for further development and new features. Especially release build has still much space left for further application code. However, limited resources in debug build might prevent proper testing of the software, if release build grows too large. The result of available hours of training to be saved into the EEPROM depends on both the implementation of MovesenseCoreLib’s logbook resource, as well as on what is being saved. Here the 18 hours reached is well sufficient for most teams’ weekly training data. However, it must be noted that if many training recordings are left to the sensor and downloaded at once, it will consequentially increase the download times as well. Therefore, it is recommended that the training data should be downloaded each time after training, if possible.

5. DISCUSSION

This thesis presents a work that was done to specify, develop and evaluate a novel team sports monitoring HR sensor application Firstbeat Sports, which is using Suunto Movesense technology as a sensor solution. The developed system is capable of measuring and analysing an athlete's ECG to derive advanced features from an exercise, such as TRIMP, EPOC and training effect. This data can be saved to internal memory in addition to broadcasting the data over BLE. This provides means for a coach to monitor their team and individual athlete's performance in real-time, or offline after the training.

First objectives of the thesis were to develop and evaluate an application for team sports monitoring using Movesense, as well as to evaluate the Movesense platform's suitability for the same use case. The development was driven by set requirements from the business. Use cases and technical specifications were written. Continuous evaluation of the system (both Movesense sensor and the application) under development was run, to ensure reaching of desired features of the new product. As part of this thesis, supervised field test results were parsed and analysed to evaluate the performance of the developed sensor application. A controlled field test was also performed to validate the accuracy of R-peak detection of ECGRR-module. In addition, power consumptions and memory usages of the application were measured and optimized.

Another objective was to explore ways to perform debugging and offline testing more efficiently. During the development, an RRI-signal-generator was designed and developed for these purposes. This made it possible to chain multiple sensors to the same RRI-signal-generator, to simulate real field tests in a repeatable manner. In the future this will allow developing a more fully automated continuous integration (CI) -pipeline with hardware-in-loop simulated testing.

The original deadline of a product release was aimed to be during the first quarter of 2019. The project involved much collaboration both externally with Suunto and internally within Firstbeat. During the project there were some challenges which hindered the work from reaching original time estimates of product release. These challenges included e.g. late deliveries (ETE, MovesenseCoreLib, ECGRR-module, iPad client application), project management issues (not definitive deadlines, lack of resources in code reviewing, changing requirements), MovesenseCoreLib SW issues and Movesense sensor HW issues (several bugs in SW and HW, lack of documentation). Also, lack of documentation and inexperience with a similar system from the past, added with the lack of code reviewing or second opinion, made the application development challenging and slower than what it could have been. To finally reach the target of releasing a market-ready product, some changes in the project management were done and some code reviews were performed. However, most importantly collaboration between Suunto was deepened, to find and fix issues both in MovesenseCoreLib and in developed application. Close connection and co-working on the project with Suunto, including a week spent at their premises, were needed to get things progressing and done. Considering all these challenges, a lot has been learned on collaboration and project managing of a project like this, and overall it has been a successful journey. The final product Firstbeat Sports sensor was released during the second quarter of 2019, 13th of May at Firstbeat HRV Summit in Helsinki.

In the future, there are many possible directions where the development of the sensor application could be taken. It is possible to not only improve and optimise the current application code (i.e. power consumption or resource usage optimisation), but new features can also be added. These features could e.g. benefit from accelerometer data. Also, adding some already existing features from ETE-library, such as QRT, might be desired. In addition to changes into the application code, updates on MovesenseCoreLib can also be expected.

6. CONCLUSIONS

In this thesis, a novel team sports monitoring HR sensor application was developed and evaluated. The results of the evaluation show that both Movesense and the developed Firstbeat Sports sensor application are well suited for team sports monitoring with Firstbeat Sports solution. The final measurement validation logic was concluded to reach 100% success rate in measuring all the team trainings, whenever Suunto Movesense ECGRR-module would detect sufficiently good ECG signal. However, in the presence of user error or signal quality issues, a 95% success rate was received. To minimise these issues, HR belts should have better insulations and athletes should be well instructed on how to properly use HR belts. Also, the controlled field test results show that the mean measurement error between ECGRR-module RRI and Bodyguard 2 was only 0,003 ms with 0,688 ms standard deviation. Therefore, the Movesense sensor is well comparable with Bodyguard 2, proving Movesense an accurate RRI measurement sensor for team sports monitoring. The results of the system evaluation show that all the objectives, which were set to the thesis and the project, were reached.

To provide better usability, power consumption was measured and optimised. The results predict the battery of the sensor to last well over 4 months with 10 hours of training being measured each week. Also, the final application was measured to allow approximately at least 18 hours of training to be logged into the internal memory of the sensor, which is well sufficient for most of the use cases of team sports.

Integration of the physiological algorithm library was well accomplished, the new product providing advanced real-time analytics of ETE library to run within the sensor. Also, the developed Firstbeat Sports sensor is the first in the markets to provide real-time TRIMP/min analytics of training for momentary internal load tracking.

Final product was publicly released on 13th of May 2019.

REFERENCES

Akiba, Davidson, R., Cufí, C. & Townsend, K., 2014. Getting Started with Bluetooth Low Energy. In: s.l.:O'Reilly Media, Inc..

Anon., 2019. *White Papers and Publications.* [Online]
Available at: https://www.firstbeat.com/en/science-and-physiology/white-papers-and-publications/
[Accessed 13 5 2019].

Ardelean, D., 2015. *Mobile development.* [Online]
Available at: http://sviluppomobile.blogspot.com/2015/09/using-ibeacons-with-windows-10-part-i.html
[Accessed 1 5 2019].

Argenox, 2019. *Argenox.* [Online]
Available at: https://www.argenox.com/library/bluetooth-low-energy/ble-advertising-primer/
[Accessed 1 5 2019].

Atkielski, A., 2007. *Wikimedia.* [Online]
Available at: https://commons.wikimedia.org/w/index.php?curid=1560893

Banister E, 1991. Modelling elite athletic performance. *MacDougall JD, Wenger HA, Green HJ (eds) Physiological testing of the high-performance athlete, 2nd edn. Human Kinetics Publishers Ltd, Champaign,* p. 403–424.

Bluetooth SIG, 2013. *BLUETOOTH SPECIFICATION Version 4.1.* [Online]
Available at: https://www.bluetooth.com/specifications/archived-specifications/
[Accessed 13 5 2019].

Bluetooth SIG, 2019. [Online]
Available at: https://www.bluetooth.com/develop-with-bluetooth/qualification-listing/qualification-listing-fees/
[Accessed 14 5 2019].

Cables and Sensors, 2019. *12-Lead ECG Placement Guide with Illustrations.* [Online]
Available at: https://www.cablesandsensors.com/pages/12-lead-ecg-placement-guide-with-illustrations

Catapult Sports, 2019. *Catapult Sports.* [Online]
Available at: https://www.catapultsports.com/

CMake, 2019. *CMake.* [Online]
Available at: https://cmake.org/
[Accessed 4 2019].

Doglio, F., 2018. REST 101. In: *REST API Development with Node.js : Manage and Understand the Full Capabilities of Successful REST Development.* s.l.:Apress.

Epxx, 2019. *Artigos.* [Online]
Available at: https://epxx.co/artigos/bluetooth_gatt.html
[Accessed 15 3 2019].

Ernst, G., 2017. Heart-Rate Variability-More than Heart Beats?. *Front Public Health,* 11 Sep.Volume 5.

Fergus, A., 2017. *How to easily measure your heart rate variability.* [Online]
Available at: https://www.alexfergus.com/blog/how-to-easily-measure-your-heart-rate-variability
[Accessed 10 5 2019].

Fielding, R. T., 2000. Architectural Styles and the Design of Network-based Software Architectures. *Doctoral dissertation, University of California.*

Firstbeat Technologies Ltd., 2012. *Whitepaper: Indirect EPOC Prediction Method Based on Heart Rate Measurement.* [Online]
Available at:
https://assets.firstbeat.com/firstbeat/uploads/2015/11/white_paper_epoc.pdf
[Accessed 14 5 2019].

Firstbeat Technologies Ltd., 2015. *Whitepaper: Recovery Analysis for Athletic Training Based on Heart Rate Variability.* [Online]
Available at: https://assets.firstbeat.com/firstbeat/uploads/2015/11/Recovery-white-paper_15.6.20153.pdf

Firstbeat Technologies Ltd., 2019. *Firstbeat.* [Online]
Available at: https://www.firstbeat.com/en/blog/real-time-trimp-min/
[Accessed 14 5 2019].

Firstbeat Technologies Ltd., 2019. *Introducing Firstbeat Sports Sensor and Live app: Bringing focus and mobility to coaching.* [Online]
Available at: https://www.firstbeat.com/en/news/introducing-firstbeat-sports-sensor-and-live-app/
[Accessed 13 5 2019].

Firstbeat Technologies Oy, 2019. [Online]
Available at: https://www.firstbeat.com/en/
[Accessed 13 5 2019].

Firstbeat Technologies Oy, 2019. *Professional sports.* [Online]
Available at: https://www.firstbeat.com/en/professional-sports/team-solutions/
[Accessed 13 5 2019].

Firstbeat Technologies, 2014. *Whitepaper: Automated Fitness Level (VO2max) Estimation with Heart Rate and Speed Data.* [Online]
Available at:
https://assets.firstbeat.com/firstbeat/uploads/2017/06/white_paper_VO2max_30.6.2017.pdf
[Accessed 14 5 2019].

FreeRTOS, 2019. *FreeRTOS.* [Online]
Available at: https://www.freertos.org/RTOS.html
[Accessed 4 2019].

FreeRTOS, 2019. *FreeRTOS.* [Online]
Available at: https://www.freertos.org/Embedded-RTOS-Queues.html
[Accessed 4 2019].

Gaesser, G. & Brooks, G., 1984. Metabolic bases of excess post-exercise oxygen consumption: a review. *Medicine and Science in Sports and Exercise,* Volume 16, pp. 29-43.

Gay, W., 2018. Beginning STM32: Developing with FreeRTOS, libopencm3 and GCC, Chapter 5. In: s.l.:Apress.

Halson, S.L, 2014. Monitoring Training Load to Understand Fatigue in Athletes. *Sports Med,* 44(Suppl 2)(139).

HeartMath Institute, 2019. *Heart Rate Variability.* [Online]
Available at: https://www.heartmath.org/research/science-of-the-heart/heart-rate-variability/

Jaakko Malmivuo, R. P., 1995. *Bioelectromagnetism.* New York: Oxford university Press.

Jules A. A. C. H., P. G., Stuurman F. E., Wouter A. S. de,Muinck Keizer, Yuri M. M., Cohen A. F., 2018. Repeatability and predictive value of lactate threshold concepts in endurance sports. *PLoS One,* 13(11).

Kaikkonen, P., Hynynen, E., Mann, T. et al., 2010. Can HRV be used to evaluate training load in constant load exercises?. 108(435).

Kim, H. G., Cheon, E. J., Bai, D. S., Lee, Y. H., & Koo, B. H., 2018. Stress and Heart Rate Variability: A Meta-Analysis and Review of the Literature. *Psychiatry investigation,* 15(3), p. 235–245.

Lacamera, D., 2018. Embedded Systems Architecture, Real-Time application platforms. In: s.l.:Packt Publishing.

Larman, C. & Basili, V. R., 2003. Iterative and incremental developments: A brief history. *Computer,* 36(6), pp. 47 - 56.

Malik, M., 1996. Heart rate variability: Standards of measurement, physiological interpretation, and clinical use. *Circulation,* Volume 93. 1043-1065..

Microchip, 2019. [Online]
Available at:
https://www.microchip.com/devtoolthirdparty/CompanyListing.aspx?compid=aa7433f4-941c-4d94-9e6c-e00470f8e3de
[Accessed 14 5 2019].

Ninja, 2019. *Ninja.* [Online]
Available at: https://ninja-build.org/
[Accessed 4 2019].

Nordic Semiconductor, 2015. *Nordic Semiconductor.* [Online]
Available at: https://devzone.nordicsemi.com/f/nordic-q-a/9622/application-debug-with-softdevice
[Accessed 1 5 2019].

Nordic Semiconductor, 2017. *SoftDevice Specification S132 SoftDevice v4.1.* [Online]
Available at: https://www.nordicsemi.com/Software-and-Tools/Software/S132
[Accessed 10 3 2019].

Nordic Semiconductor, 2019. *nrf52832 System on Chip.* [Online]
Available at: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52832
[Accessed 3 2019].

OpenAPI Initiative, 2019. *Implementations.* [Online]
Available at: https://github.com/OAI/OpenAPI-Specification/blob/master/IMPLEMENTATIONS.md#implementations
[Accessed 4 2019].

OpenStax College, 2013. *Wikipedia.* [Online]
Available at:
https://commons.wikimedia.org/wiki/File:2018_Conduction_System_of_Heart.jpg

Parak, J. & Korhonen, I., n.d. Accuracy of Firstbeat Bodyguard 2 beat-to-beat heart rate monitor.

Polar Electro, 2019. *Polar Team Pro.* [Online]
Available at: https://www.polar.com/en/b2b_products/team-pro

Portaankorva, A., 2018. *Master's thesis: EVALUATION OF BLUETOOTH LOW ENERGY TECHNOLOGY.* Tampere: Tampere University of Technology.

Qoitech, 2019. [Online]
Available at: https://www.qoitech.com/
[Accessed 11 5 2019].

Roell, M. et al., 2018. Player Monitoring in Indoor Team Sports: Concurrent Validity of Inertial Measurement Units to Quantify Average and Peak Acceleration Values. *Frontiers in Physiology,* Volume 9, p. 141.

Scheid, J. L. & O’Donnell, E., 2019. Revisiting heart rate target zones through the lens of wearable technology. *ACSM's Health & Fitness Journal,* 23(3), p. 21–26.

Scribbans, T. D., Vecsey, S., Hankinson, P. B., Foster, W. S., & Gurd, B. J., 2016. The Effect of Training Intensity on VO2max in Young Healthy Adults: A Meta-Regression and Meta-Analysis. *International journal of exercise science,* 9(2), p. 230–247.

SEGGER Microcontroller GmbH, 2019. *SEGGER.* [Online]
Available at: https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/
[Accessed 5 2019].

SEGGER Microcontroller Systems LLC, 2019. *SEGGER.* [Online]
Available at: https://www.segger.com/products/development-tools/systemview/
[Accessed 4 2019].

Shaffer, F. & Ginsberg, J. P., 2017. An Overview of Heart Rate Variability Metrics and Norms. *Front Public Health.*

SIG, B., 2019. *Bluetooth.* [Online]
Available at: https://www.bluetooth.com/specifications/gatt/services/
[Accessed 28 3 2019].

Soligard T., Schwellnus M, Alonso J., et. al., 2016. How much is too much? (Part 1) International Olympic Committee consensus statement on load in sport and risk of injury. *Br J Sports Med,* Issue 50, pp. 1030-1041.

Springhouse, 2002. *Lippincott Professional Guides: Anatomy & Physiology.* Philadelphia: Wolters Kluwer Health.

Suunto Movesense, 2019. *Movesense.* [Online]
Available at: https://www.movesense.com/
[Accessed 10 5 2019].

Suunto Movesense, 2019. *Movesense Bitbucket.* [Online]
Available at: https://bitbucket.org/account/user/suunto/projects/MOVC

Texas Instruments, 2016. *BLE-Stack User's Guide for Bluetooth 4.2.* [Online]
Available at:
http://dev.ti.com/tirex/content/simplelink_cc2640r2_sdk_1_40_00_45/docs/blestack/ble_user_guide/html/ble-stack-3.x/l2cap.html
[Accessed 25 3 2019].

Wikipedia, 2019. *CMake.* [Online]
Available at: https://en.wikipedia.org/wiki/CMake
[Accessed 4 2019].

Wikipedia, 2019. *Nordic Semiconductor.* [Online]
Available at: https://en.wikipedia.org/wiki/Nordic_Semiconductor
[Accessed 4 2019].

Wikipedia, 2019. *Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Real-time_operating_system
[Accessed 25 4 2019].

Wikipedia, 2019. *Wikipedia.* [Online]
Available at: https://en.wikipedia.org/wiki/Preemption_(computing)
[Accessed 4 2019].