Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Simon Gåseby Gjerde

Shoes for the future - Developing smart shoes for continous measurement of ballistocardiography

Master's thesis in Mechanical Engineering
Supervisor: Martin Steinert
June 2022

NTNU
Kunnskap for en bedre verden

Simon Gåseby Gjerde

Shoes for the future - Developing smart shoes for continous measurement of ballistocardiography

Master's thesis in Mechanical Engineering
Supervisor: Martin Steinert
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Abstract

This master thesis describes the development process of a pair of smart shoes for measuring ballistocardiography. The thesis was a continuation of a proof-of-concept prototype from a previous project thesis. The thesis aimed to develop a functioning smart shoe for measuring ballistocardiography and validate the results through testing. In addition, any serendipitous findings in the process were to be explored.

The shoes were developed using an iterative approach based on wayfaring with prototyping as one of the main tools. Multiple prototypes were created to validate hypotheses and act as a tool for generating new design questions. Both divergent and convergent prototypes were created to ensure the best solution was created. In addition, a benchtop setup was created for simulating the ballistocardiogram to enable rapid testing of the impact of changing design parameters.

The final proposed concept, called Sole 2.0, consists of a pair of running shoes with five water-filled bladders in each shoe, each connected to a pressure sensor. The smart shoe measures the pressure under the user's feet and captures the body's movement due to the ejection of blood. In addition, the sole 2.0 allowed a BCG scale to be connected to act as a reference for the captured BCG and to attach a PPG sensor for calculating pulse transit time and aiding in segmentation.

To validate the concept, a test was run with 14 participants. Participants were measured in three test periods with one resting period between tests two and three. A cold pressor test was used to increase the participants' blood pressure during test period two. Blood pressure was increased to enable the calculation of pulse travel time. The results from the testing are not fully processed. However, the preliminary results from two of the fourteen datasets show great promise, with the BCG being captured in all measurement periods and the inverse relationship between pulse transit time and blood pressure being shown.

Sammendrag

Denne masteroppgaven beskriver utviklingen av et par med smartsko for å måle ballistokardiografi. Masterprosjektet er en fortsettelse av en prosjektoppgave som lagde en prototype som et konseptbevis. Målet med masteroppgaven var å lage en smartsko som måler ballistokardiografi, samt validere resultatene gjennom testing. I tillegg, skulle alle «serendipitous» oppdagelser bli utforsket.

Skoene ble utviklet gjennom en iterativ metode med utgangspunkt i Wayfaring med prototyping som hovedverktøy. Prototyper ble laget for teste og validere resultater, samt fungere som et verktøy for å generere kunnskap og design spørsmål. Både divergerende og konvergerende prototyper ble laget for å nå den beste løsningen. I tillegg, ble en småskala testoppsett laget som simulerte ballistokariografi for å kunne raskt teste og validere påvirkningen til diverse design parametere.

Det endelige konseptet, Sole 2.0, består av et par med joggesko med fem væskefylte blærer i hver sko. Hver av blærene er koblet til en trykksensor. Smartskoen måler trykket under foten til brukeren og fanger opp bevegelsen til kroppen som følge av blodsirkulasjonen. I tillegg, er Sole 2.0 designet for å kunne kobles til en ballistokardiografi vekt som fungerer som en referanse og en PPG-sensor for å måle pulstransporttid og muliggjøre segmentering.

For å validere resultatene ble et forsøk gjort med 14 deltakere. Deltakerne ble i målt tre perioder med en hvileperiode mellom test to og test tre. I testperiode to ble en «coldpressor» test utført for å øke blodtrykket til deltagerne. Blodtrykket ble økt for å gjøre det mulig å beregne pulstransporttid. Resultatene fra forsøket er ikke behandlet, men foreløpige resultater er svært lovende. To ut av 14 datasett er behandlet og ballistokardiografi er suksessfullt målt i alle testperiodene for de to datasettene. I tillegg er den beregnede pulstransporttiden korrekt invers relatert til blodtrykket.

Preface

This master thesis describes the development process of designing and testing a smart shoe solution for measuring ballistocardiography. The project was carried out from January 2022 to June 2022. The thesis has been written to fulfill the requirements for the degree of Master of Science from the Department of Mechanical and Industrial Engineering at NTNU. The supervisor for the project has been Martin Steinert.

The project has been carried out at TrollLabs at NTNU in Trondheim.

Acknowledgments

I would like to thank TrollLabs and its community for support and help in solving the challenges in the thesis. I would especially like to thank the TrollLabs community for making the entire process fun. In particular, I want to thank Torjus Steffensen for sharing infinite his wisdom in developing solutions for measuring medical signals. Thank you to Martin Steinert for creating TrollLabs and its fantastic community.

I would like to thank my parents and my siblings for all the help and support throughout my degree. Finally, I want to thank my wife, Ida for her encouragement, love an,d patience.

Contents

Figures

Tables

Abbreviations/Symbols

NTNU	Norges teknisk-naturvitenskapelige universitet
BCG	Ballistocardiography
PPG	Photoplethysmography
PTT	Pulse travel time
PWV	Pulse wave velocity

1 Introduction

Cardiovascular diseases are one of the leading causes of death in the world, contributing to as many as 17.9 million deaths a year (WHO n.d.). Many cardiovascular diseases can be prevented by implementing lifestyle changes such as reduced tobacco use, increased physical activity, and improved diet. Early detection of cardiovascular diseases is vital to implement treatment as early as possible. Noninvasive methods for continuous monitoring of cardiovascular health are one method for early detection. One promising method for continuous assessment of cardiovascular health in a noninvasive manner is Ballistocardiography (BCG).

This master thesis is a continuation of a project thesis completed in the autumn of 2021. The project thesis explored the solution space of sensor setups for measuring physical signals related to the heart. The project thesis culminated in a promising proof-of-concept prototype for measuring BCG by measuring the pressure under a person's foot. This master thesis continues the project thesis with the following goal:

"Develop and test a smart shoe for continuous cardiovascular monitoring using Ballistocardiography."

For developing and testing the smart shoe BCG concept, many subgoals need to be achieved. The concept from the project thesis needs to be understood better. What parameters for the setup are important, and what limitations exist. In general, the design challenges in measuring BCG are little understood. For testing the smart shoe, a more comprehensive range of users need to be tested to validate its function on all users and understand how variations such as weight, height, age, and form of the feet might impact the result. In addition, other design solutions and serendipitous findings should be explored.

The master thesis has been completed at TrollLabs at NTNU in Trondheim. TrollLabs research and prototype lab with a multidisciplinary research group at NTNU. Trollabs focuses on creating new radical ideas and improving the fuzzy front-end of engineering design.

The master thesis is split up into four main parts and eight sections. Part one introduces the problem statement, necessary background information, and theory and consists of sections 0 and 2. Part two covers the development process culminating in a functioning version of the shoe and consists of sections 3 and 4. Part three covers the testing of the prototype and results from the tests and consists of sections 5 and 6. Part 4 covers the discussion of the results, prototype, and development process as well as the conclusion and consists of sections 7 and 8. The culmination of the master thesis is a paper for IEEE Sensors 2022 to be submitted on the 18. of June 2022. The draft of this paper is in Appendix A.

2 Theory and background

2.1 Recap of the project thesis

The master thesis is a continuation of the project thesis described in Appendix B. The project thesis was carried out from August 2021 to December 2021. The project thesis explored the solution space related to wearable physiological sensors.

Ballistocardiography was discovered as a potential candidate for continuous non-invasive measurement of cardiovascular health. The project thesis culminated in a proof-of-concept prototype. The proof-of-concept prototype consisted of two air-filled bladders connected to a corresponding pressure sensor. The bladders were placed under an MDF plate, one under the heel and the other under the front of the foot of the user. The proof-of-concept prototype gave a signal which, after segmenting, using a PPG as a reference, and bandpass filtering the pressure, a clear repeating pattern was discovered. The waveform of the repeating signal was similar to the BCG waveform but with a less pronounced J peak. It was concluded that it was highly likely that it was the BCG being captured due to the similar waveform and matching timing with the pulse. Using two air-filled bladders also allowed for measuring the pressure distribution, which was seen as a possible point of interest.

The final prototype had many flaws, and the testing had much to be desired. Motion fragments were still a big issue and meant the user had to stand very still. The amount of noise in the signal was still significant. In addition, a problem was discovered with the air-filled bladders compressing and killing the signal due to compressing completely or blocking the connection to the sensor. This led to a minimum size of the air-filled bladders so as to have enough volume for it to take the static weight of the user. The proof-of-concept prototype was also only tested on one person, which meant that differences in weight, height, form of the foot, and other characteristics might heavily impact the results. Little knowledge was also gained about how the various design parameters affected the result.

2.2 Ballistocardiography

Ballistocardiography is the measurement of the body's movement due to the ballistic forces created by the ejection of blood at each cardiac cycle (Giovangrandi et al. 2011). It is a non-invasive measurement method for assessing cardiovascular health (Omer T. Inan et al. 2015). BCG was a popular area of research from the 1940s to the 1980s (Giovangrandi et al. 2011). It fell out of favor due to several reasons. One issue was the lack of standard measuring techniques which led to differences in the measured signal from one measurement method to another. Two other issues were a lack of understanding of the physiological correlation to the different parts of the signal and a primary focus on clinical diagnostic where other measurement methods were superior(Giovangrandi et al. 2011). It has, in the recent two decades increased in popularity due to an increased interest in out-of-clinic measurement and advances in sensor technology and computers, enabling new methods for measuring BCG (Giovangrandi et al. 2011).

Figure 1 shows an example of a BCG waveform that has been redrawn based on the Starr BCG which is a longitudinal BCG (Starr et al. 1939). Peaks and valleys are referenced by letters. The peaks and valleys are not associated directly with underlying events but represent the combined mechanical pulse response of the body and vascular system due to the ejection of blood at each cardiac cycle(Omer T. Inan et al. 2015). The valleys and peak corresponding to the IJK complex is the most prominent feature of the BCG and the easiest to recognize for design purposes.

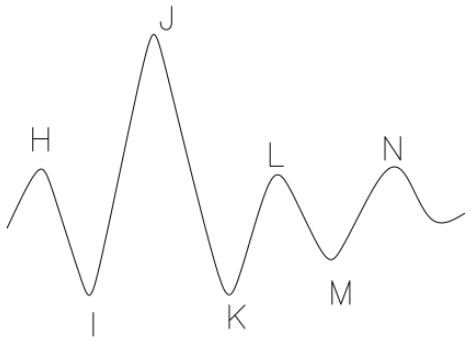


Figure 1 - BCG waveform redrawn based on the Starr BCG (Starr et al. 1939)

With increased interest in out-of-clinic assessment of cardiovascular health, BCG has seen a resurgence in interest as a possible non-invasive continuous measuring method. Many methods for capturing the BCG have been created. (Koivistoinen et al. 2004) use EMFI sensors in a chair to capture the BCG. (O. T. Inan et al. 2009), (Martin et al. 2016) and (Campo et al. 2017) use a modified bathroom scale that captures the variation in load sensor deformation. (Mora et al. 2020) use an accelerometer attached to a bed to capture the movement of the body and the BCG. (Omer T. Inan et al. 2015) and (Yao et al. 2019) use on body accelerometers to capture the BCG.

BCG has been used to measure multiple physical parameters. (Campo et al. 2017), (Martin et al. 2016), (C.-S. Kim et al. 2018) and (Pinheiro, Postolache, and Girão 2009) have used ballistocardiography to compute pulse wave velocity. Pulse wave velocity is an important marker for cardiovascular health due to its relation to blood pressure and arterial stiffness (Pereira, Correia, and Cardoso 2015). (Mack et al. 2009) measure heart rate and breathing rate, which can be used for sleep analysis.

2.3 Development methodology

2.3.1 Wayfaring

The Fuzzy Front end of product development is the early phase of development before requirements are set and a concept is considered ready for development (J. Kim and Wilemon 2002). The fuzzy describing the unclear and unknown part of the situation where opportunities and the process is unknown. The knowledge gap is significant for the designers and the design path is unknown. Big ideas cannot easily be deductively deduced. Wayfaring is one method for navigating the design space during the early phases with high uncertainty.

Wayfaring was first proposed by Steinert and Leifer as hunter-gatherer model for navigating the design space. It is a nonlinear dynamic model where designers use

iterative prototyping and abductive reasoning to learn and move through the design space(Steinert and Leifer 2012). The hunter-gatherer model has been further developed as the wayfaring model by (Gerstenberg et al. 2015). The wayfaring model improves the hunter-gatherer model by introducing multi-directional prototyping where multiple directions are developed synchronously, and multiple disciplines are included. The wayfaring model uses probing by prototypes and iterative design-build-test cycles as its framework. The designer probes the design space and takes steps in promising directions as knowledge increases.

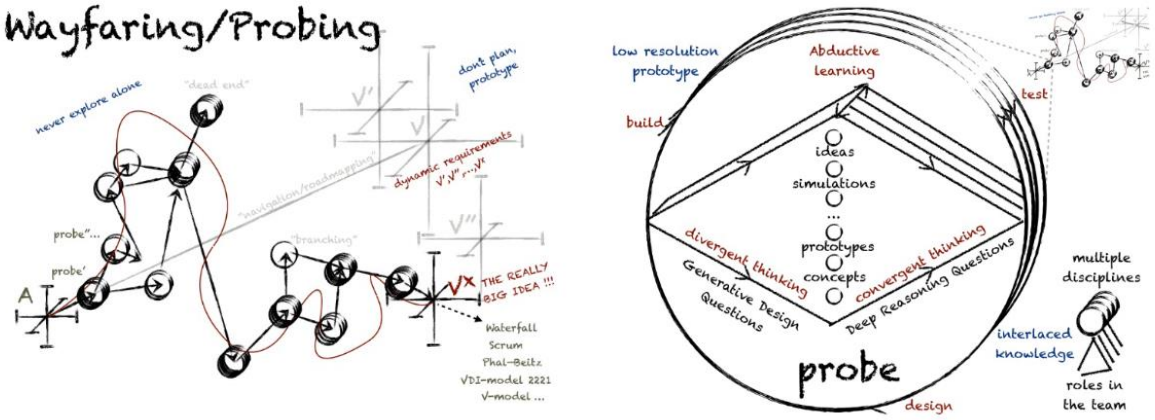


Figure 2 - Wayfaring and probing figures from (Gerstenberg et al. 2015)

Probing in the wayfaring model consists of cycles of divergent and convergent thinking. Divergent thinking is enabled through generative design questions. These are open-ended questions with the goal of removing design constraints and inspiring creative exploration. The convergent thinking is done through deep reasoning questions where the performance and intended function of concepts and solutions are evaluated. Prototypes are an essential part in answering both the generative and deep reasoning questions.

The exploratory fashion of navigating the design space using wayfaring enables serendipity to influence the designer and lets the designer change direction often. This is important as the knowledge of the design space increases with each probing cycle. The explorative fashion of wayfaring also enables the designer to elicit "unknown unknowns". Unknown unknowns are challenges which are not articulated and must be discovered in the design process (Sutcliffe and Sawyer 2013). Wayfaring and prototypes can be used to dynamically create requirements as knowledge increases. By using dynamic requirements, it is easier for the designer to handle unknown unknowns through the design process (Kriesi et al. 2016).

2.3.2 Prototyping

Protoypes are an essential part of the product development process (L. S. Jensen, Özkil, and Mortensen 2016). (Lauff, Kotys-Schwartz, and Rentschler 2018) describe a prototype as tools for enhanced communication, learning and informed decision making. They argue prototypes are essential for learning about the design space and the technical elements as well as making informed decisions about the viability of the product. Prototypes can be an embodiment of a concept or idea or created as an exploratory tool.

Within product development the exploration of known unknowns and the discovery of unknown unknowns are important (Sutcliffe and Sawyer 2013). Prototypes are one tool for these tasks. (M. B. Jensen, Elverum, and Steinert 2017) introduce the term prototrial where prototypes are actively used to explore and understand the design space. They argue that prototypes can be used for divergent thinking and generating concepts. (Elverum and Welo 2015) introduce directional and incremental prototypes. Directional prototypes are not purely for validation and verification but serve to explore the impacts of larger design changes. Incremental prototypes answer the classic design concerning validation and verification. (Kriesi et al. 2016) use prototypes as an essential part of the wayfaring process and argue that the use prototypes can help uncovering and handling unknown unknowns. Prototyping critical functionalities led to them discovering unknown problems in a cheap and effective manor and avoiding costly rework.

In general, prototypes can be used for validation and verification, but another common usage is generating knowledge and design questions by creating prototypes and exploring how they work. They can answer deep reasoning questions such as "is this better than..?" but also serve as generative tools answering questions such as "how many ways can we …?".

3 Development process

The development process of this thesis is a continuation of the development process of the project thesis in Appendix B. Due to having a proof-of-concept prototype the challenge for the development process consisted of both an exploratory and an incremental improvement challenge. The proof-of-concept prototype showed a possible method for capturing the BCG, but due to the low understanding of the concept other better solutions might exist. As such a combination of a wayfaring approach and a more incremental testing approach were used. The main process consists of a convergent process on the proof-of-concept prototype solution and a divergent process looking for other better solutions.

3.1 Bencthop-setup for simulating BCG

The proof-of-concept prototype developed in December 2021 left many questions on how design choices impacted the results and to what degree they impacted. For further learning and exploration of the design space, larger prototypes with higher quality would have to be made. This slows down the iteration process if all learning is to be done through larger prototypes. The project thesis revealed the need for faster ways to explore how design choices impact results. For assessing how smaller design choices affect the results a setup for simulating BCG with easily swappable parameters was needed.

The BCG is the repetitive movements which arise from the ejection of blood from the heart as described in 2.2. The concept for the proof-of-concept prototype in the project thesis is that these movements will lead to changes in the pressure under a person's feet. The pressure under a person's foot is a sum of the pressure due to weight and movement with the weight acting as a static pressure and movements acting as a variable pressure. Some of the variable pressure is due to movement related to BCG. Simulating this varying pressure is interesting, since it will increase the understanding of the solution space and it is relatively easy to simulate the varying pressure at a small scale. A method for simulating this varying pressure was designed using a smart pump on an evaluation kit (Evaluation Kit, tppventus, Cambridge UK.

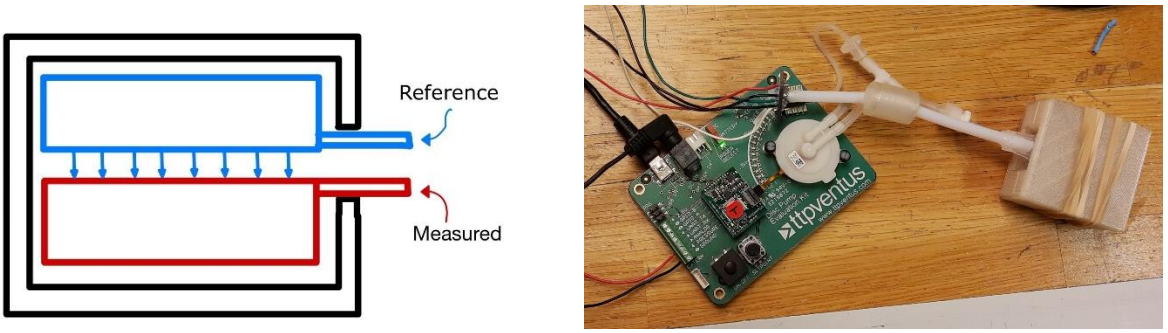


Figure 3 - Benchtop setup

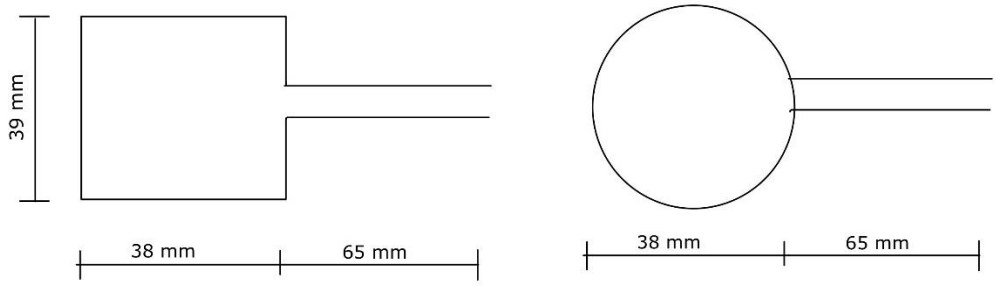
The bench-setup simulation consists of a smart pump from ttpventus, a BMP388 pressure sensor (BMP388, Bosch Sensortec GmbH, Germany), a signal generator and two

bladders. One bladder connected to the smart pump, and one connected to the BMP388 sensor. The bladders are created by cutting sheets out of weldable plastic and using a soldering iron to weld the sheets together. Both bladders are enclosed within a 3D printed box which ensures good contact between them. The smart pump varies the pressure in one bladder which then applies a varied pressure to the bladder the BMP388 sensor is measuring the pressure off. The smart pump can measure its pressure and functions as a reference for the pressure measured by the BMP388. A waveform for the pressure variation was generated using a waveform generator attached to the evaluation kit. The waveform used was a Meyer wave due to its similarities to the IJK complex of the BCG waveform. It is important to note that the smart pump had sustained some damage prior to use and as could not mimic the Meyer wave perfectly.

When designing the bench-top setup it was important to facilitate easy change of variables. Aspects such as varying the size and surface area of the measured bladder, including material between the two bladders and varying pipe length was initially seen as interesting aspects to vary. The box was created with extra space filled in with MDF which meant free space was available if needed. A special connector was created for quickly swapping the bladder connected to the BMP388. The connector consisted of a 3D printed tube coated with silicone. Two pipes could be connected using this as the silicon created an airtight seal between the pipe and the 3D printed tube. The tube had a lip on each side to make sure the silicon was not allowed to expand or escape.

3.2 Initial parameter testing

Initially five parameters were identified from the proof-of-concept prototype as important parameters to understand. These were pipe length, shape of bladder, the effect of applying silicone on sensor, and the combination of applying silicone on the sensor with water in the bladder. The bladder parameters are shown in Figure 4. To evaluate these parameters six test were run. One as a reference, and one for each variation of parameter. At this point the problem was still an exploratory problem and not an optimization problem. As such the goals were to identify parameters which had major impact on the result and were interesting to further research. Therefore, only visual inspection by plotting reference versus measured was done.



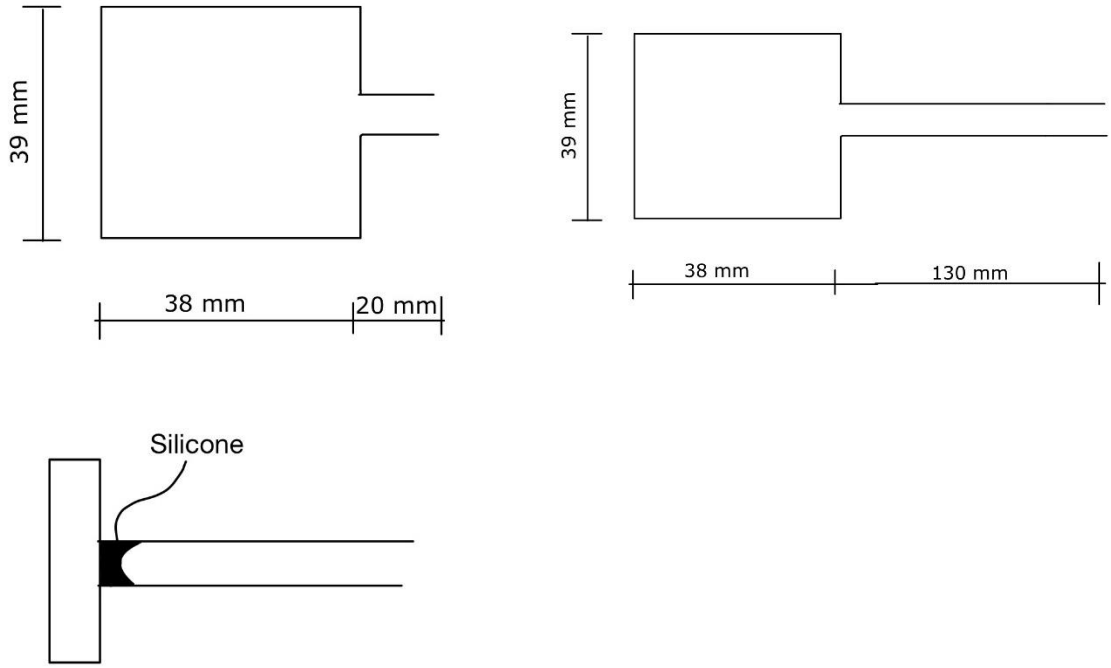


Figure 4 - Bat testing parameters

The results of the tests are shown in Figure 5. Test 1 was a reference test for comparing other tests. Test 2 checked the circle shaped bladder. This saw a slight increase in the amplitude of the measured pressure, which was concluded to most likely be due to the reduced surface area of the circle compared to the square. This was a key takeaway as it meant reducing the surface of the bladder would increase the amplitude of the BCG signal and make it easier to detect. Test 3 and 4 checked a short and a long pipe length. These tests showed no major impact. Test 5 tested the impact of applying a coating of silicon to the sensors and test 6 tested using water instead of air in the bladder with silicone on the sensors. Both of these parameters showed minimal impact on the results.

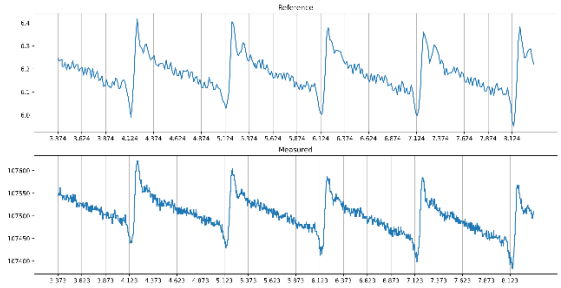


Figure 5a – Normal

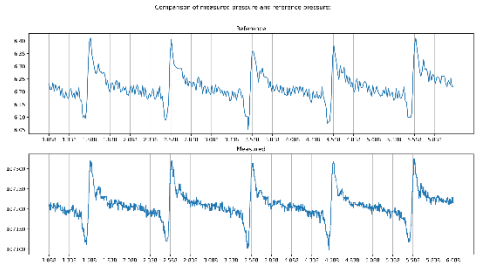


Figure 5b - Circle

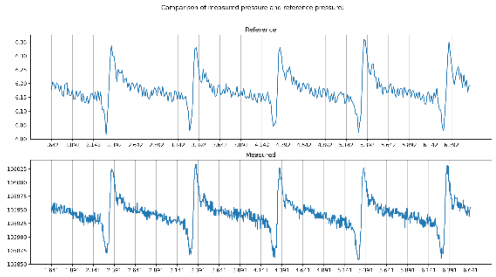


Figure 5c - Short pipe

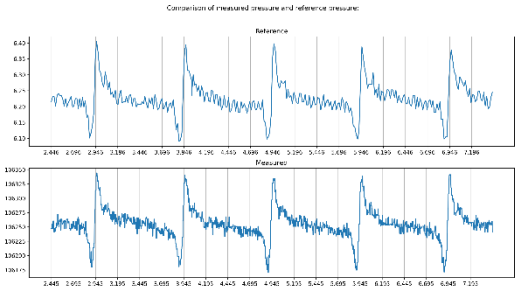


Figure 5d - Long pipe

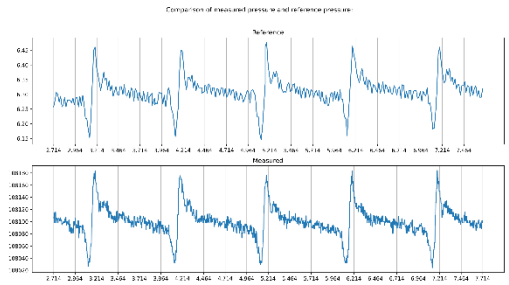


Figure 5e - Silicone no water

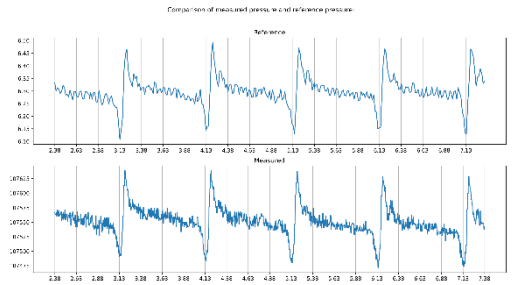


Figure 5f - Silicone with water

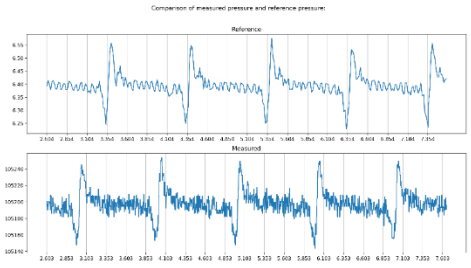
Figure 5 - Parameter testing

3.3 Sole 0.5

With the knowledge from the bench top simulation a new higher fidelity prototype was made with the goal of either acquiring a working prototype or uncovering further design questions. To create this prototype some more design questions had to be answered. It was desirable to have a material between the bladders and the foot to improve stability and lessen the impact of motion fragments. The impact of having a material between the bladder and the foot was uncertain and had to be tested. In addition, the impact of placement of the bladders was also unknown.

The bench top setup was used to test the impact of having something between the bladder and the foot. Two test were run, one with a 0.6mm metal sheet between the reference bladder and the measurement bladder and another with a 6mm MDF sheet. The 0.6 mm showed no major impact. However, the 6mm MDF plate gave noticeably worse results. The conclusion was that material between the foot and the bladders need to be thin to reduce absorption of the forces.

Metal plate:



Mdf 6 mm:

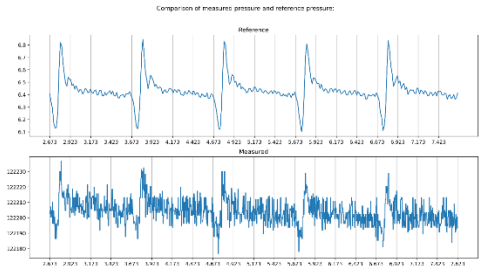


Figure 6 - Comparison of material between reference and BMP388

The placement of sensors were chosen by checking where the pressure under the feet was largest as the assumption was that this is where most of the forces were transferred to the ground. This was checked by placing plastic pipes under the plate which deformed when standing on them. The areas with the largest deformations had the largest forces. The deformation was the biggest at the heel and at the front of the foot. Four places of interest were initially chosen. The heel and front of the foot due to the aforementioned largest forces and the middle of the foot as well as an extra sensor on the front. The middle and the extra on the front were chosen for two reasons. The assumption that he best locations were the areas with the largest forces might be wrong and it was seen as interesting to measure pressure distribution which could have extra interesting uses. The four placements are shown in Figure 7.

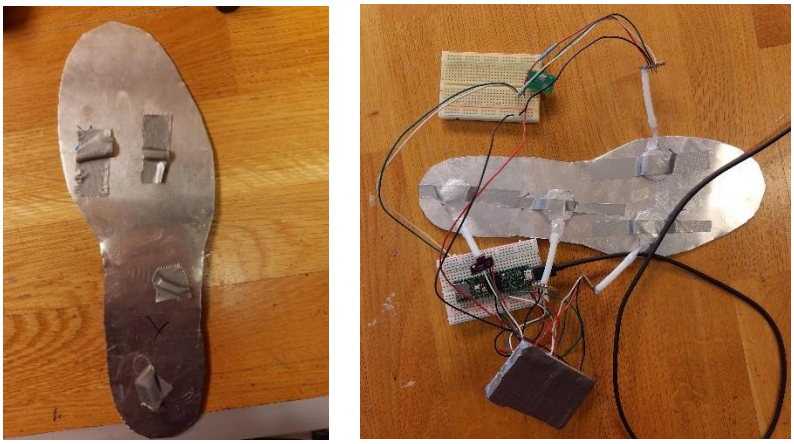


Figure 7 - Sole 0.5 Sensor placements and wiring

The results of test are shown in Figure 8. The results of the test were disappointing with no clear sign of the BCG being captured even after filtering. The results were bandpassed filtered with a bandpass of [3, 10]. These poor results were likely due to the bladders compressing almost entirely and the plate mostly touching the floor. The hypothesis was that most of the forces were transmitted to the floor and creating little to no pressure difference. This was due to air being compressible and the volume of air being small. In addition, the bladders were prone to leak air which made testing difficult. The production of the bladders was difficult as air easily escapes and getting a high enough volume of air in the bladder was highly challenging.

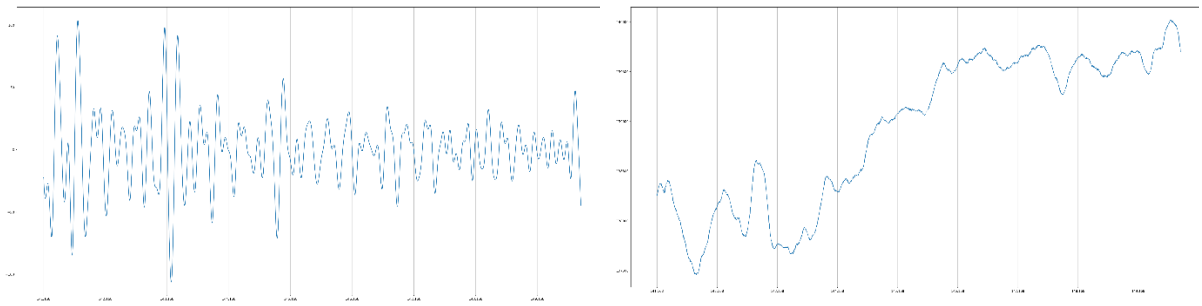


Figure 8 - Results from sole 0.5 test

3.4 Diverging prototypes

As design fixation is a large and important problem in product development a focus was made on always looking for alternative methods for capturing the BCG. Two such methods were using inductance variation and using the varying electrical resistance in a carbon fiber silicone composite due to deformation.

3.4.1 Inductance based prototype

Inductance measurement was seen as one possible way of capturing the BCG. The idea was that it was possible to attach a magnet to the shoe sole and an inductance meter under the magnet. With the variation in forces the shoe sole would deform increasing and decreasing the distance between the inducation measuring unit and the magnet. This was tested using a LDC1612 (LDC1612, Texas Instruments, Texas USA) inductance meter and a normal magnet. Upon separating the inductance meter from the magnet with a soft material it was possible to measure the change in inductance when the force on the magnet was manually changed and the soft material deformed. This was tested in the bench-top setup. The magnet was separated from the induction meter with a deformable foam. This achieved somewhat promising results as seen in Figure 9. However, the signal seemed somewhat noisier than with the BMP 388 sensor tests.

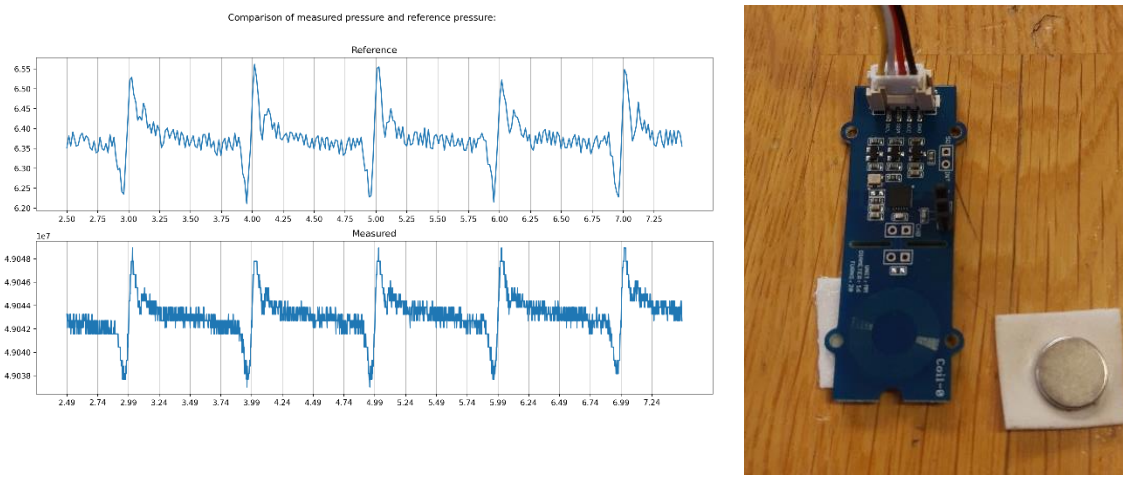


Figure 9 - Inductance test in bench top setup

The inductance concept was tested in a deconstructed running shoe as shown in Figure 10. A part of the sole was cut out and the inductance meter was placed near the bottom of the sole. Then most of the cutout was returned on top of the inductance meter. The magnet was glued on top of the cutout. The testing showed poor results with no clear signal being captured. The inductance setup was tested again with a setup similar to the sole 1.0 testing setup as seen in Figure 10. This still showed poor results. The main

theory for the poor results is that the force related to the BCG is very small and the material does not deform enough to give a noticeable signal compared to noise. Exchanging the material for a softer one or lowering the amount of material between the inductance meter and the magnet might improve the results. No fundamental flaws were found with this concept, and it was deemed a optimization problem. However, due to it not showing any greater potential than the concept using pressure sensors it was not explored further due to time limitations.

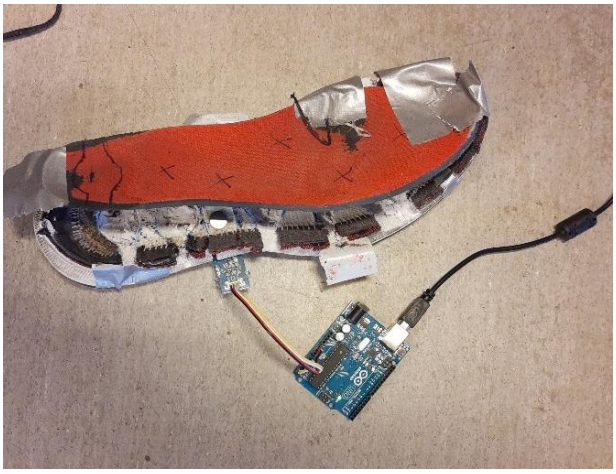


Figure 10 - Inductance test setup

3.4.2 Variable resistance in a carbon fiber and silicone composite

Carbon fiber in a silicone composite was another interesting concept for measuring BCG. It was based on another project at TrollLabs where the usage of short carbon fiber strands in silicone matrix is used as a sensor (Vestad and Steinert 2019). When the carbon fiber enforced silicone is deformed the electrical resistance changes. As such a concept was to measure this change in resistance when a person stands on a part made of the material.

A 20x20x3mm cube of the carbon fiber enforced silicone was donated from the other project. Preliminary test by manually applying forces to deform the cube gave promising results. Very small forces were enough to give a noticeable output. A test was run using the cube of the carbon fiber enforced silicone. The setup was based on the testing setup from sole 1.0 which had shown potential. The cube is placed under a MDF plate directly under the heel of the foot with a 3 mm MDF plate supporting the front of the foot. This is shown in Figure 11. The signal was fairly noisy, and no clear signal was captured. The main theory behind the disappointing results was that the force related to BCG is small and gave very small deformations in the material. This combined with the fact that the cube gives relatively noisy results ends up giving poor results. A softer material as the matrix could improve this. This is also likely an optimization problem. However, due to the setup not showing any major improvements over the BMP388 setup it was explored further due to time limitations.

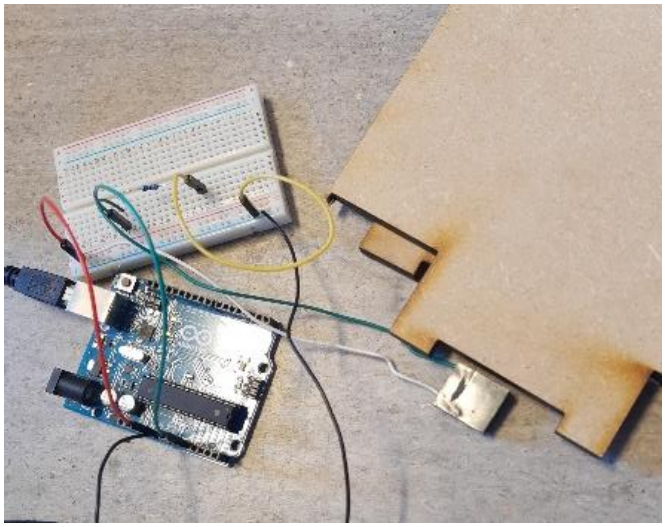


Figure 11 - Testing setup for testing carbon fiber sensor

3.5 Piston-cylinder prototype

As the production of the bladders was challenging iteration speed slowed. Another concept with a larger scale and easier produce parts was needed. A concept was developed inspired by gas springs where a piston-cylinder setup would replace the air-filled bladders. Piston-cylinder pairs were quickly created by deconstructing syringes. The casing of the syringe was cut in half and glued to an acrylic plate creating an airtight seal. The piston part of the syringe was cut to a shorter length and glued to an MDF plate. A hole was cut in the cylinder to connect the BMP388 sensor, which enabled pressure measurements. This setup showed greater promise in ease of creation.

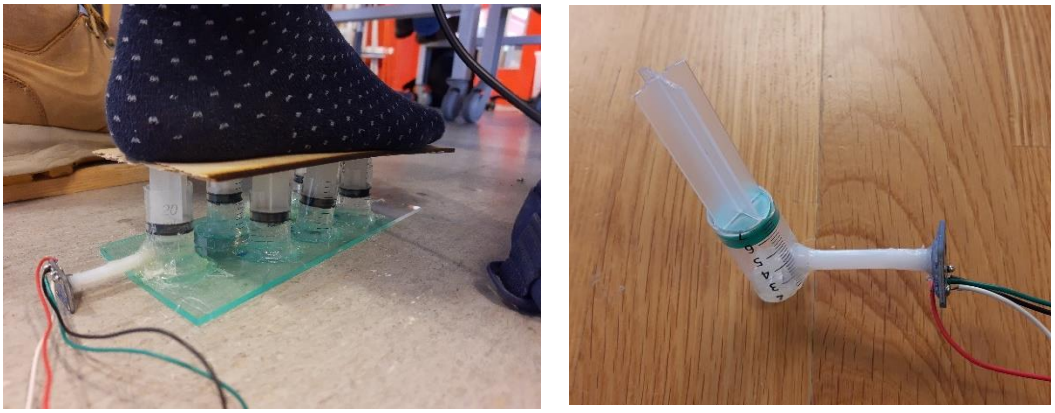


Figure 12 - Piston cylinder setup

This setup gave a few key insights. It showed how much the air would compress when full bodyweight was applied. It became apparent that at no point would there be enough air in one of the bladders to ensure that the plate would not touch the ground and disturb the BCG signal. Water had thus far not been used due to complicating the design as water and electronics generally don't match. However, it became apparent that some sort of incompressible substance was needed with water being the easiest available. Further tests were conducted with water in the cylinders. A BMP388 with silicon applied was used to measure the pressure in one of the piston housings under the heel. The pressure signal was filtered using a bandpass filter with a bandpass of [3, 10] Hz. This

gave a very promising result shown in Figure 13. Clear peaks and valleys can be detected. These appear about every 0.8 seconds which match well with a pulse of about 80 beats per minute which is a realistic pulse to have in this scenario.

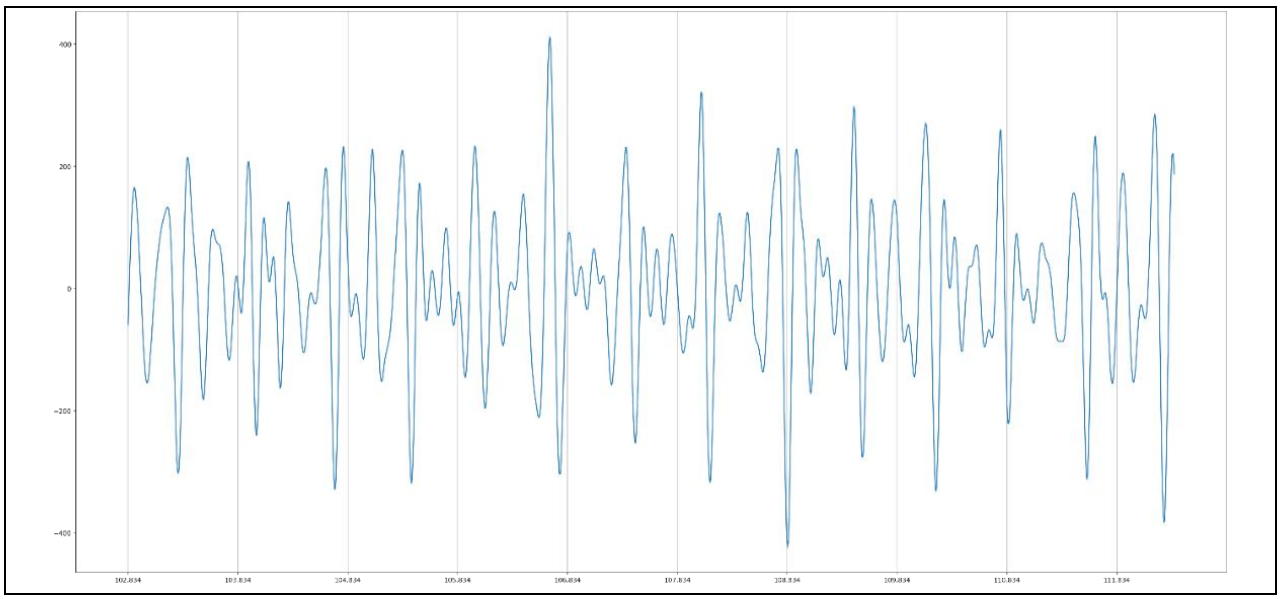


Figure 13 - Results from piston cylinder setup tests

The gas spring prototype gave very promising results. Miniaturizing this prototype into a size which would fit in a shoe sole was a promising approach to the task. One concept for creating this was 3D printing gas springs in a small size using a SLS printer as SLS printers can create fully artight structures. However, this proved difficult as the the tolerances of the parts were small and even small warping in the 3D printed parts would create leaks or introduce friciton which would dampen the signal. This warping proved fatal to the concept. This concept was developed in parallell with the bladder concept. The production quality of the bladders increased withouth the warping issue being resolved. As the primary limitation for this development process is time this concept had to be scrapped. It however, showed great promise and could be good solution for future work.

3.6 Sole 1.0 - High fidelity prototype two

The gas spring prototype gave very promising results and paved the way for another high-fidelity prototype using the bladder concept. One of the primary takeaways from the piston-cylinder testing was that water had to be used as compression was a big issue. Testing this improvement two more test were conducted using a bladder filled with water instead of air. The first test consisted only of one water filled bladder under the heel and MDF as support under the front of the foot. This gave surprisingly poor results considering the setup was fundamentally almost identical to the piston-cylinder setup.

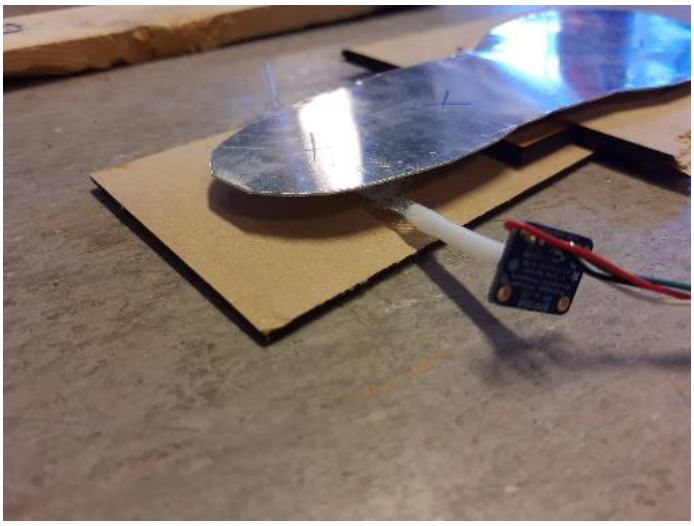


Figure 14 - Testing water-filled bladder

One of the main theories for why the results were still not satisfactory was that the back end of the metal plate was unsupported which led to many small motions in the foot to stabilize the user. This could be creating enough noise to where the signal was difficult to capture. This was tested quickly by using three sensors and supporting the metal plate by adding MDF everywhere under the plate. This gave better results which added credibility to the theory of small motions being the issue.

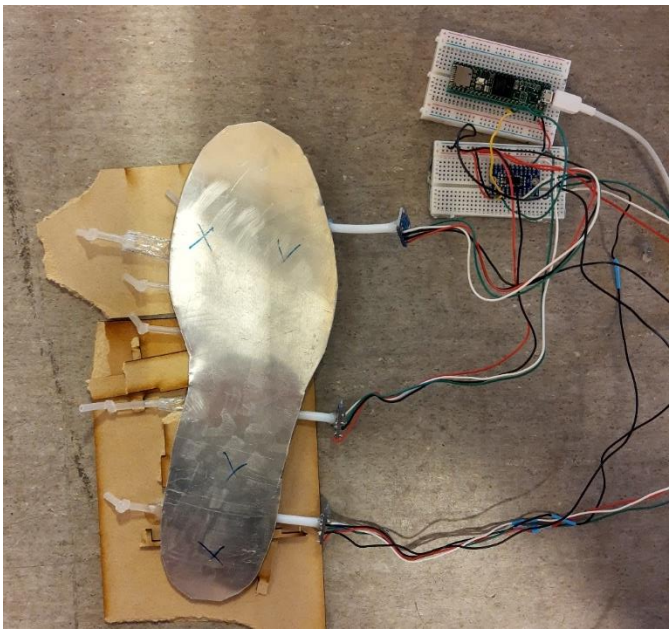


Figure 15 - Testing multiple water-filled bladders with extra support

It was of interest to create a prototype which maximized the stability for the user. This is a solved problem as high-quality shoes exist. Therefore, a prototype using a pair of shoes as foundation was wanted. However, two design questions stood out before this prototype could be developed. The first was how the support material around the water filled bladders would affect the results. The other was testing in a more formal method what positions for the water-filled bladders would be optimal.

The bench top setup was again used to gain a better understanding for how the support material would affect the results. The hypothesis was that soft infill materials might

deform and absorb some of the motions that was desirable to detect. Four different materials with different hardness was tested. MDF, a hard foam, a medium hard foam and a soft foam were tested. The setup for testing the hard foam is shown in Figure 16 with the updated water filled bladders attached to the BMP388 in Figure 17. The results are shown in Figure 18. No major differences were found which opened the design space considerably.

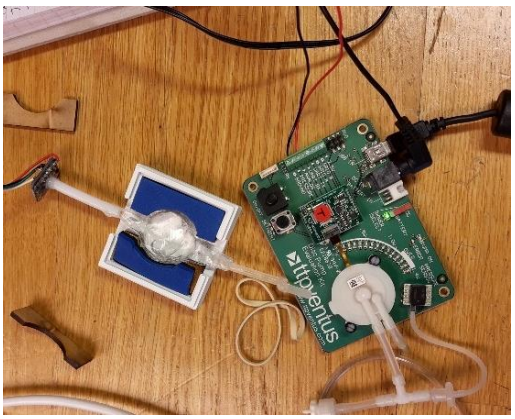


Figure 16 - Benchtop test of infill material hardness

Figure 17 - Updated design of water-filled bladder with BMP388

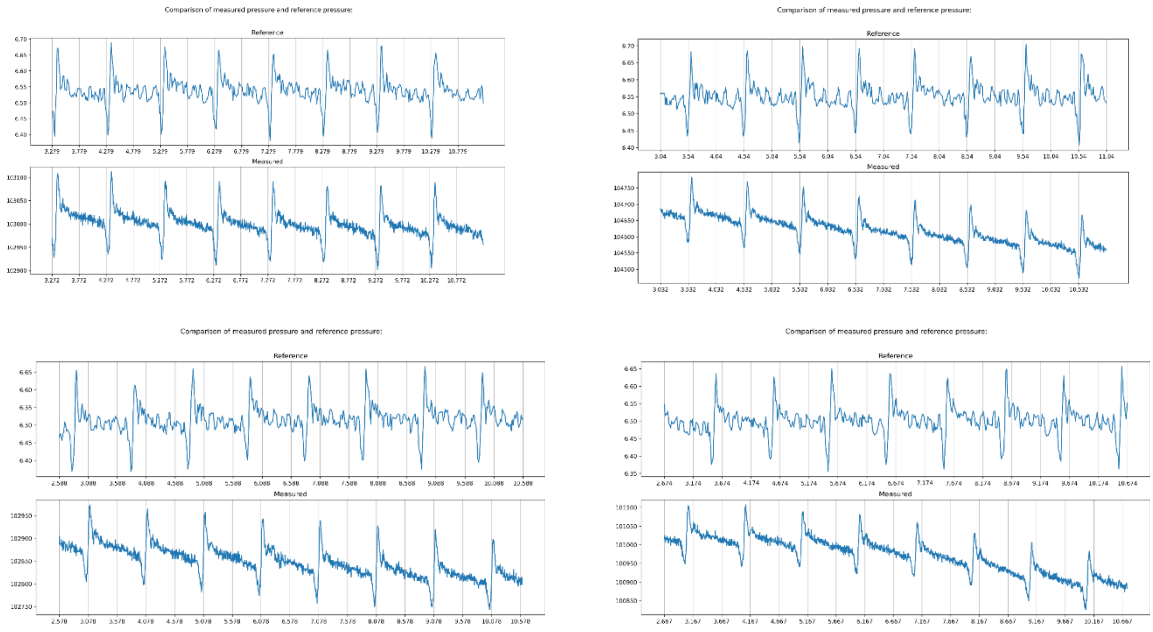


Figure 18 - Infill material testing results

Furthermore the placements of the water filled bladders were of interest. The first hypothesis was that the areas with the most pressure would absorb most of the movement and be the best placements. However, this was not certain as these areas would also have the highest variations in pressure due to natural stabilization in the foot.

As such six placements were chosen as interesting areas to test. These six positions are shown in Figure 19.

A high fidelity prototype was created by disasembling an old running shoe and using three BMP388s. As mentioned six locations were of interest for sensor placement. These six locations were marked on the shoe sole and a hole was dug for the water bladders to fit in as well as channels for the pipes connecting the sensors to lay in. The design of the water bladders and sensors were identical to the design in Figure 17.

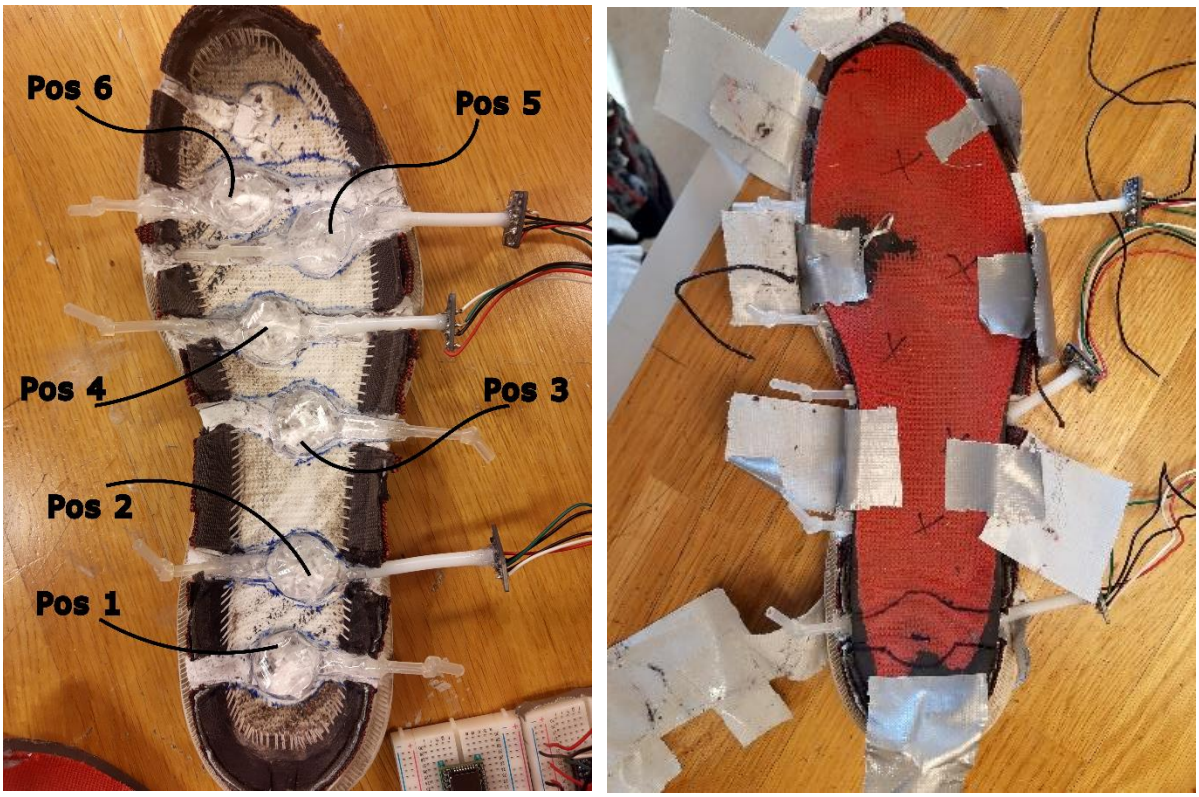
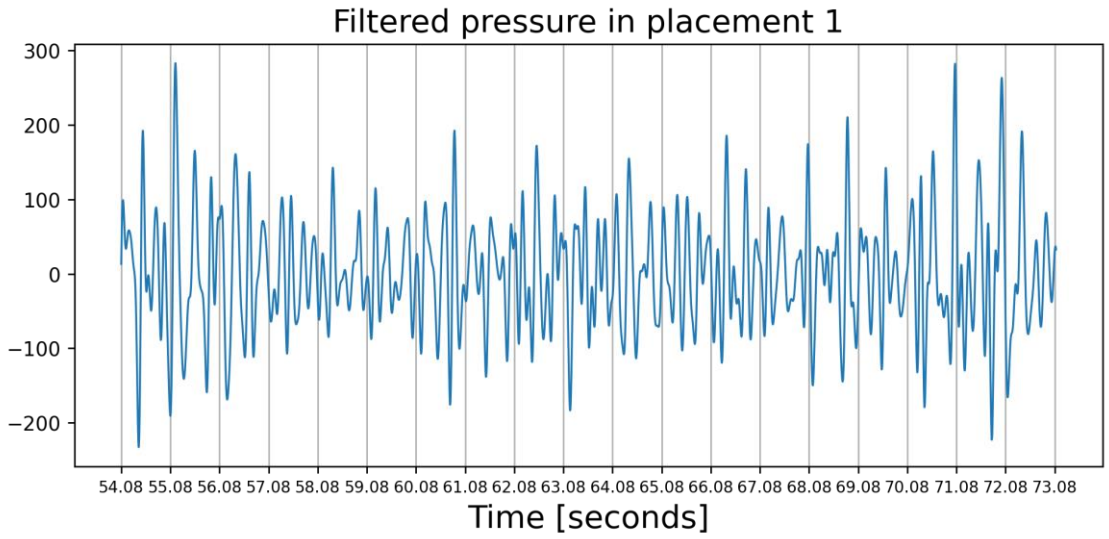
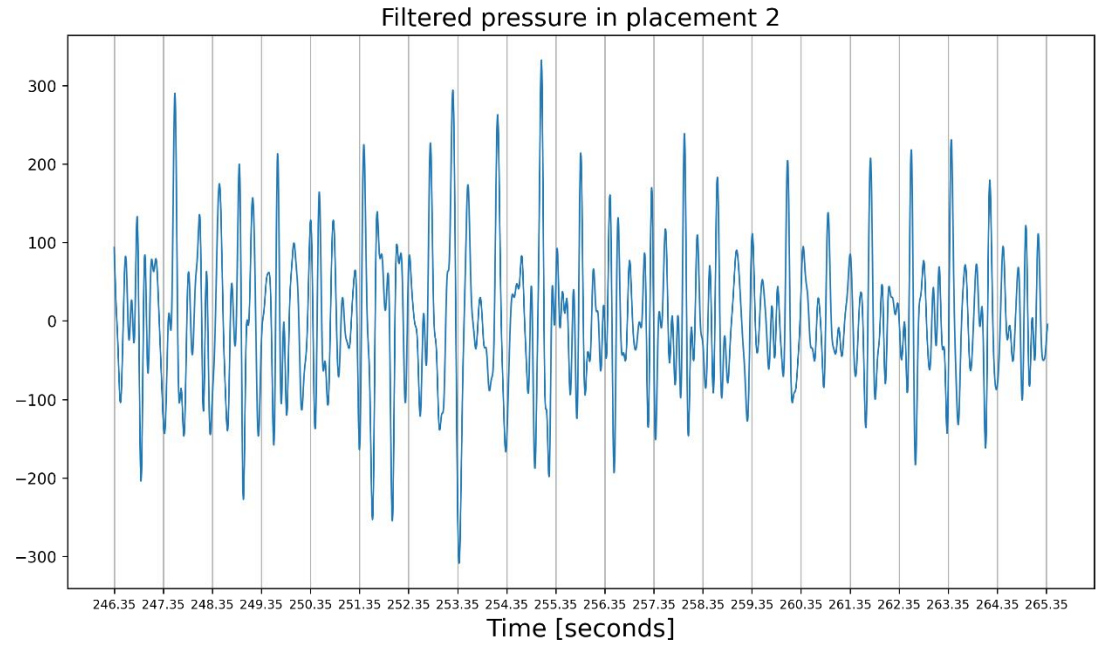
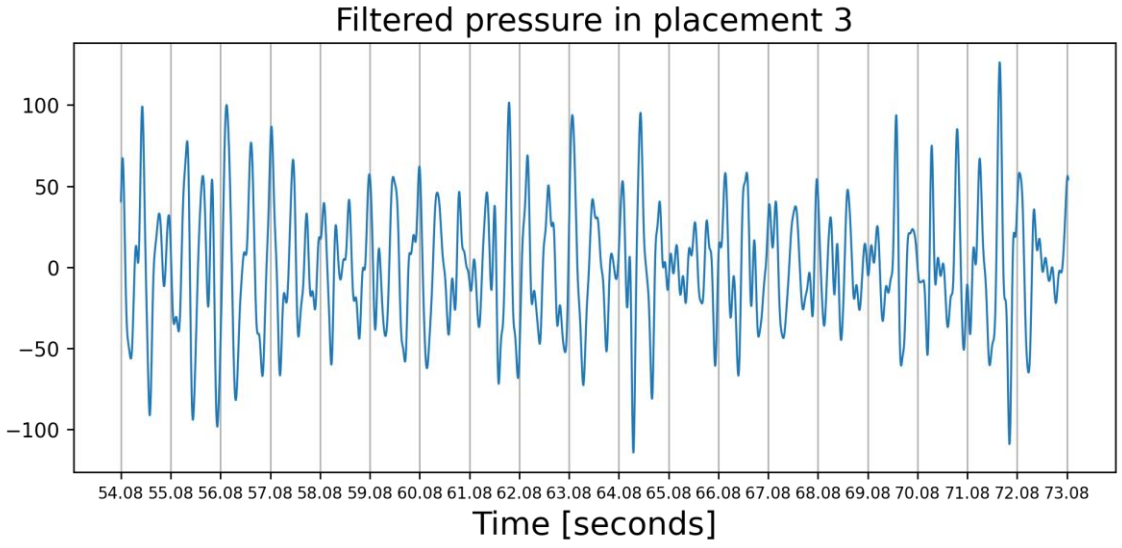


Figure 19 - Sole 1.0 prototype

Figure 20 shows the results of testing sensor location. Position one and two are the most promising for capturing the BCG with tendencies to repeating patterns with clear peaks and vallies with a frequency of the repeating patterns matching a pulse between 70-90. Position three, four, five and six are not as promising for capturing the ballistocardiogram. However they are still valuable as another interesting feature of the shoe sole is the possibility of capturing the pressure distribution.


Filtered pressure in placement 1
300
200
100
0
−100
−200
54.08 55.08 56.08 57.08 58.08 59.08 60.08 61.08 62.08 63.08 64.08 65.08 66.08 67.08 68.08 69.08 70.08 71.08 72.08 73.08
Time [seconds]


Filtered pressure in placement 2
300
200
100
0
−100
−200
−300
246.35 247.35 248.35 249.35 250.35 251.35 252.35 253.35 254.35 255.35 256.36 257.35 258.35 259.35 260.35 261.35 262.35 263.35 264.35 265.35
Time [seconds]


Filtered pressure in placement 3
100
50
0
−50
−100
54.08 55.08 56.08 57.08 58.08 59.08 60.08 61.08 62.08 63.08 64.08 65.08 66.08 67.08 68.08 69.08 70.08 71.08 72.08 73.08
Time [seconds]

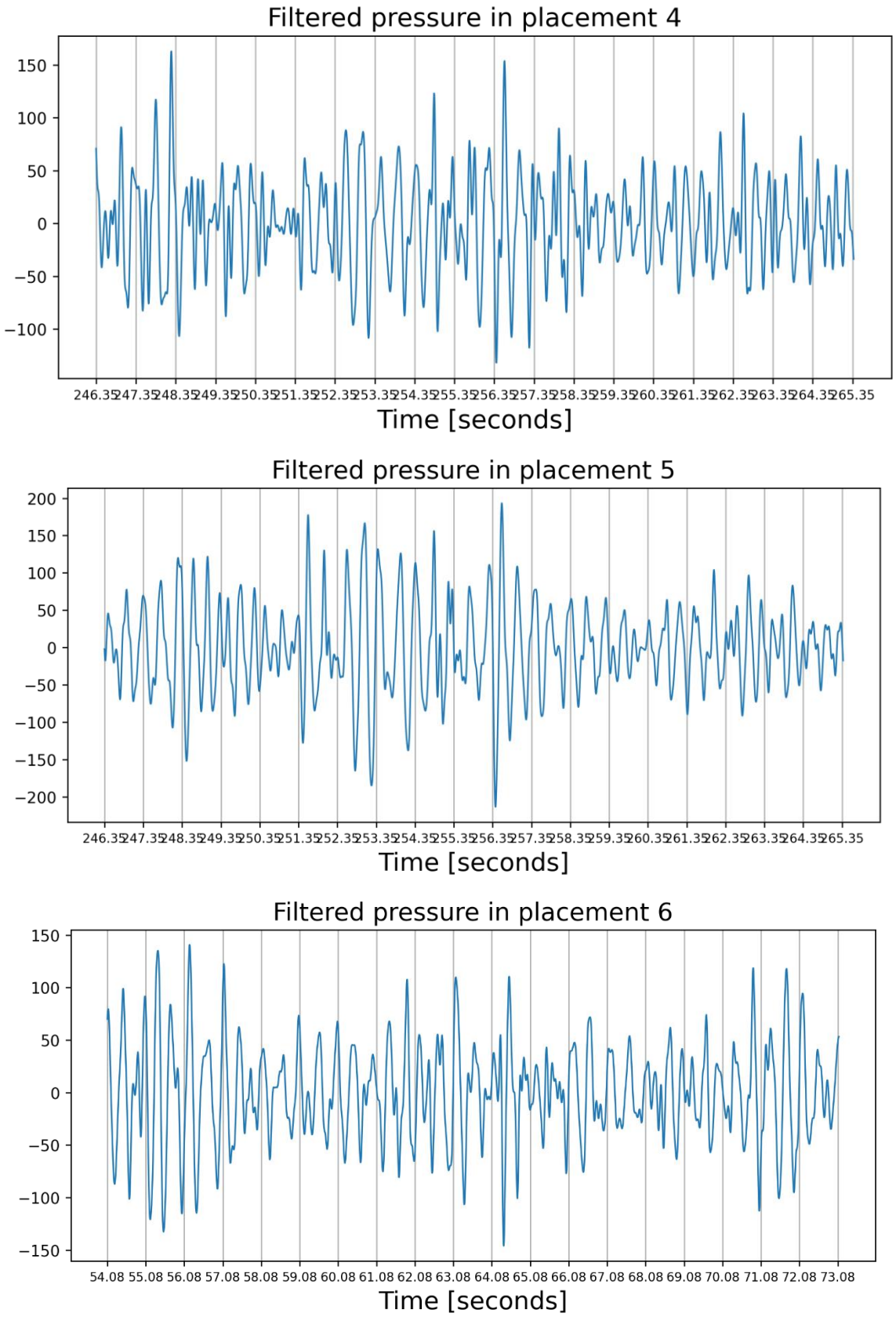


Figure 20 - Results from placement testing

Throughout the process of designing and the testing the sole prototypes an extra interesting idea was to enable the capture of the pressure distribution under the foot of

users. The initial idea was using it for gait analysis. However, during a show and tell of one of the prototypes it was highlighted how the pressure distribution could be used for monitoring the pressure distribution under the feet of people who have neuropathy in their feet. From conversations with medical personnel, it was uncovered that people with neuropathy struggle with getting sores under their feet without discovering it. To prevent and treat sores under the feet special shoe soles are developed to shift weight away from areas with sores. Tracking the pressure distribution under a person foot and looking for changes in the distribution could possibly lead to earlier detection. In addition, a version of the shoe with a higher resolution of pressure sensors could be used to speed up the process of designing special shoe soles.

3.7 BCG scale

To evaluate the results of the high fidelity tests a gold standard for comparison was needed. The BCG scale as discussed in 2.2 and is a well-known method for capturing the BCG and yields results of a high quality. As it is also a cheap method for measuring BCG it was seen as a perfect reference for comparison. As such a bathroom scale was deconstructed and changed to output its raw signal to a microprocessor. This was done by connecting the load cells to a HX711 load cell amplifier. The HX711 gives an output of the weight by connecting the load cells in a wheatstone bridge. The wiring is shown in Figure 21.

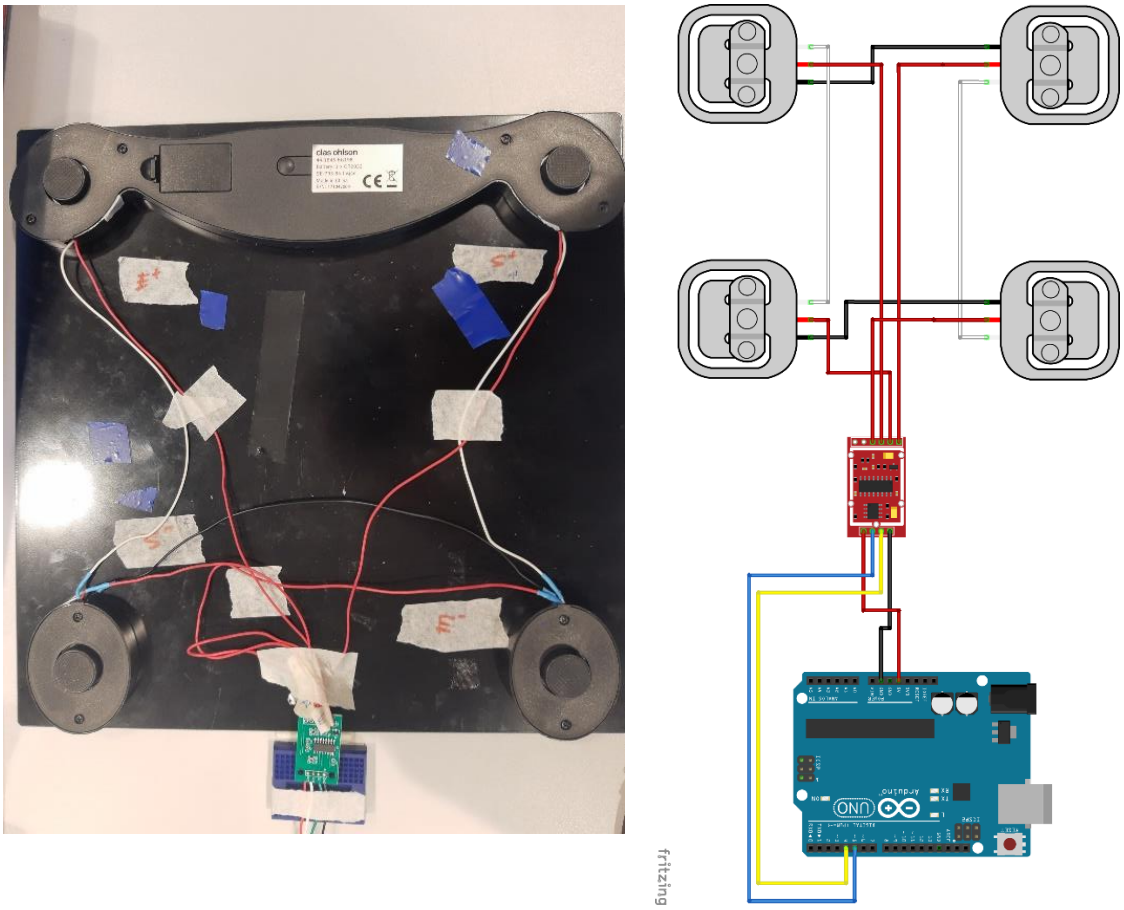


Figure 21 - Wiring diagram and setup of BCG scale

The results from the BCG scale were of high quality with the peak of IJK complex being possible to see without filtering the signal. A segmented, Z-score normalizes, unfiltered example is shown in Figure 22. Here a clear IJK complex is shown and the ballistocardiogram is easily recognized. By standing on the BCG scale with the BCG shoes it is possible to compare the results.

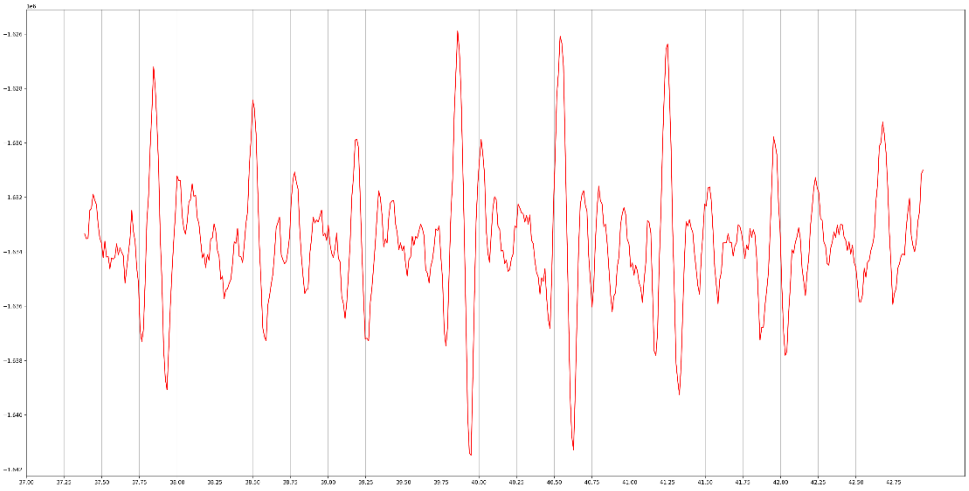


Figure 22 - Results from testing the BCG Scale

4 Sole 2.0

The final setup is based on the results from Sole 1.0. It is designed for testing with a wide range of users to validate results. It consists of a pair of running shoes in size 45 each equipped with five water bladders and correspondingly five BMP388 sensors. The five BMP388 sensors on each shoe are placed on the outside of the shoe to protect them from damage and simplify design. The pipe connecting the sensors on the outside to the water bladder on the inside is 24mm long for all sensors except position 4. The final design is shown in Figure 23. Placement of water bladders is shown in Figure 23. These positions are based on the results from testing Sole 1.0. Placements one and two were shown to be optimal for measuring BCG. Placements three, four and five are included to enable pressure distribution measurement and incase of serendipitous findings.

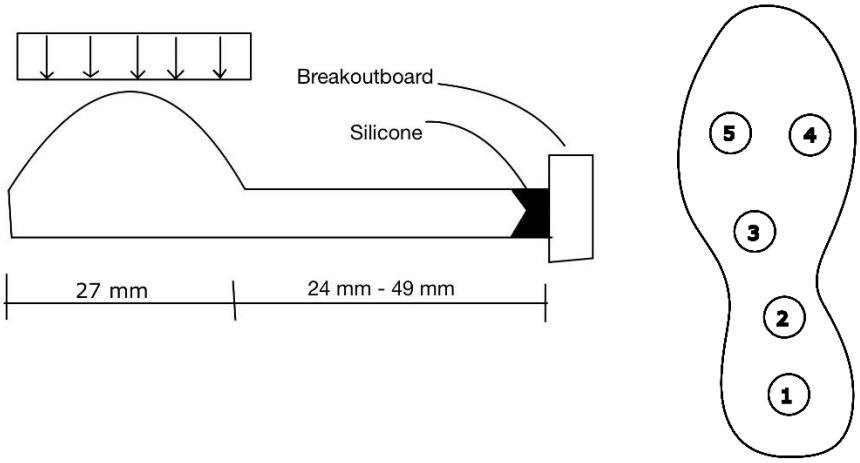


Figure 23 - Sole 2.0 bladder design and placement from Appendix A.

As shown in Figure 24 all five sensors are connected to a multiplexor with pairs of two and two sensors sharing one I2C port. The multiplexer is in turn connected to a microprocessor (Teenzy 4.1, PJRC, USA) which reads sensor output and writes it to the serial port of computer. On the computer the serial port is read using serialplotter and saved as a CSV file. In addition, the BCG scale can be connected as shown in the wiring diagram. A PPG sensor can also be connected as shown in the wiring diagram. The BCG Scale is for comparing the pressure-based BCG and the PPG is used as a reference for segmentation.

The water bladder design used is shown in Figure 23. It is made using plastic bladders meant for being welded and a soldering iron. The plastic bladders are cut into shape using stencils which form a circle with a rectangle attached. The stencils are of two different sizes with the diameter of the circles being 27 mm and 37 mm. Pairs of two plastic pieces of different size are welded together by introducing 4-8 folds into the larger of the two plastic pieces. The folds ensure the two plastic pieces have the same diameter and by introducing folds into the larger piece the bladder is naturally going to hold a dome shape which eases the process of getting water into the bladder. For connecting the water bladder to the pipe super glue is used before water is added. A soft pipe is added to the other side of the plastic bladder and is used for adding water. After water is added, the soft pipe is glued shut.

The sensors are modified slightly by adding silicone to the sensor IC. The metal lid on the BMP388 is removed and a plastic pipe with a diameter of 3 mm is glued onto the breakout board encompassing the sensor IC. The silicone is poured into the pipe using a syringe which with some effort will naturally fall onto the sensor IC. However, for the longest pipe it was necessary to cut a hole in the pipe right above the breakout board for air to escape through. After the silicon has gotten to the bottom the hole is glued shut.

The water filled bladders are lowered into the shoe sole by about 2-3 mm. This is done to get a more natural feel for the shoe, make it more stable to stand on, and decrease the maximum pressure measured by the sensors as the sensors can max out when moving. Holes are drilled in the shoe for the pipes connecting the water bladders to the sensor to go through. For the four forward sensors the shoe sole is cut down to the drilled hole so the pipes can be lowered into. For the water bladder under the heel a trench is not cut due to the way the shoes are built. Instead, the sensor is glued onto the pipe after it has been placed into the shoe sole.

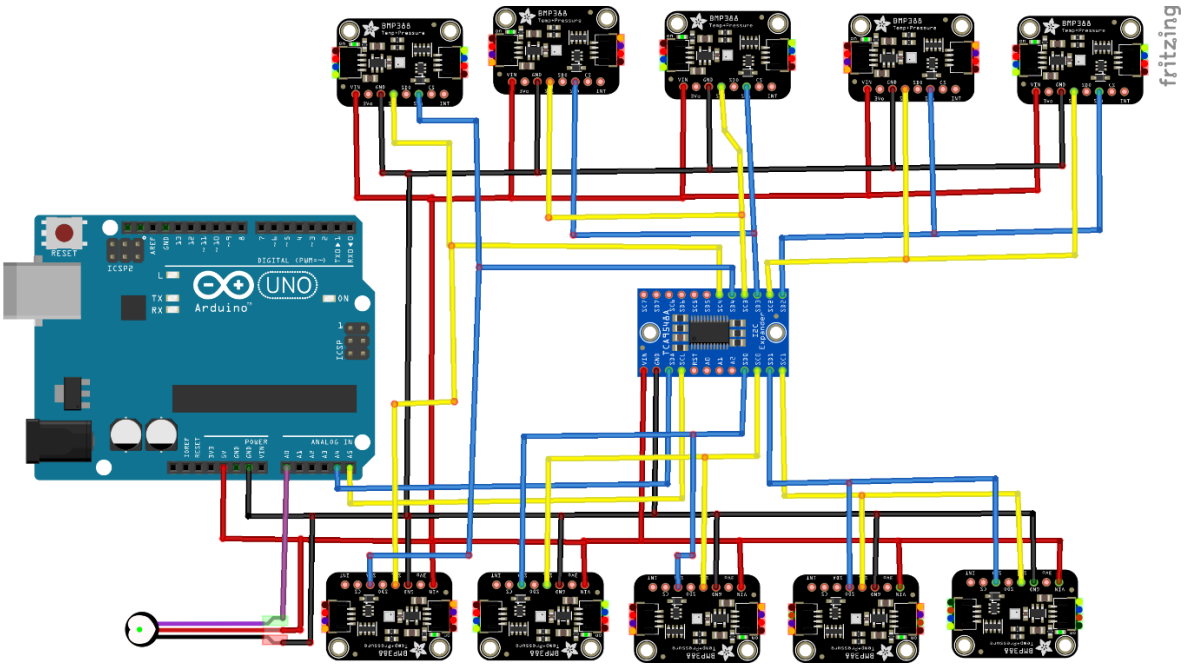


Figure 24 - Wiring for BMP388 sensors and PPG sensor for Sole 2.0

Figure 25 - Shoe 2.0 pictures

5 Testing

To validate the Sole 2.0 prototype, testing with a wider range of users was important. The prototypes have mainly been tested on the developer of the prototypes and can as such be specialized to that person bodytype. Variations in height, weight, size of feet, gender, age and more can affect the results in unknown ways. Therefore, a test was conducted on a wider range of people. The goal was to capture the BCG and validate the results. To achieve this, it was necessary to include a PPG sensor for segmentation of the pressure signal and a finger cuff for measuring blood pressure. By segmenting with the PPG the waveform could be captured and by using the PPG as timing comparison with the BCG signal the Pulse Travel Time (PTT) can be calculated. Then the BCG can be validated by comparing the changes in PTT to the changes in blood pressure as they are inversely related (Pereira, Correia, and Cardoso 2015). To induce changes in blood pressure a cold pressor test was done all participants.

5.1 Testing procedure and setup

Tests were run with 14 participants, of which 7 were female and 8 were male. All participants in the tests were between 22-29 years old. All participants gave written informed consent, and the study was approved by NSD with reference number 250185. All participants were interviewed about their gender, age, height, weight and shoe size as these parameters were hypothesized to affect the results. The detailed result of the interviews are shown in Table 1 with the median result and ranges given.

Table 1 - Results from interviews (Appendix A)

	Female	Male
Gender	7	8
Age	23.5 (22-26)	25.5 (23-29)
Height	171 (162-175)	182.5 (170.5-193)
Weight	67 (53-93)	80 (64 – 86)
Shoe size	39 (36-40)	44.5 (42.5 – 45)

Participants were instrumented with the BCG shoes on their feet. A PPG sensor (PulseSensor, World Famous Electronics llc, USA) was clipped on their left index finger and a finger cuff (MLT382, ADInstruments, United Kingdom) was placed on their left middle finger. During testing participants were asked to stand on the BCG scale. The testing was done in three test periods, noted as T1, T2 and T3, with one resting period between T2 and T3. During T1 the participants were instructed to still on the BCG scale for 1.5 minutes. After 1.5 minutes T2 was conducted with a cold pressors test. Participants were asked to put their right hand into a bucket containing water holding five degrees Celsius for 1 minute when they felt ready. After 1 minute had passed a resting period of 3 minutes was conducted to normalize blood pressure. For T3 the participants were again asked to stand on the BCG scale for 1.5 minutes.

5.2 Dataprocessing

The PPG, BCG shoes and BCG scale were all sampled synchronously using a microcontroller at 180hz. The BCG was double sampled due to the HX711 amplifier being limited to 90 hz in testing. The blood pressure was sampled at 200hz on the same computer as the microcontroller was writing its data to. The blood pressure measurement was synced with the other measurements by using the computer as a common timing system.

To remove noise and achieve a satisfactory result most of the data had to be filtered. The PPG was filtered using a Savitzky golay filter with a polyorder of three and a segment size of 0.15s. In addition to filtering noise this also made the PPG differentiable. The pressure signals were filtered using a forward-backward digital filter with a bandpass of [3,10] Hz and an order of 12. The scale was sampled at 90Hz by removing every other sample. The blood pressure was filtered using a Savitzky-Golay filter with a polyorder of 3 and a windowsize of 35 seconds.

The PPG was used for segmenting the pressure and scale signal. It was segmented by naively locating local peaks on the PPG with a spacing of at least 0.5 seconds between each peak. After segmentation each segment was normalized using z-score normalization. The segments were validated automatically by comparing segment sizes and removing segments that differentiated by more than 5% from the median segment size. The segments were also validated manually by plotting the PPG segments on top of each other and comparing the waveform as well as plotting the identified peaks for visual inspection as shown in Figure 26. Each segment of the PPG were calculated from a peak of the PPG to the next peak with a padding of 0.5s on each side. The BCG shoes and scale was segmented using the identified time segments from the PPG segmentation. The BCG shoe, BCG scale and PPG used windows of 25 seconds within each measurement period.

The changes in PTT was calculated by comparing the spacing between the peak of the PPG average and the J peak of the BCG average. The peak locations was computed using a naïve peak detection algorithm and manually reviewed after. The corresponding blood pressure for each window of measurements was calculated by finding the mean of the systolic blood pressure measured by the finger cuff.

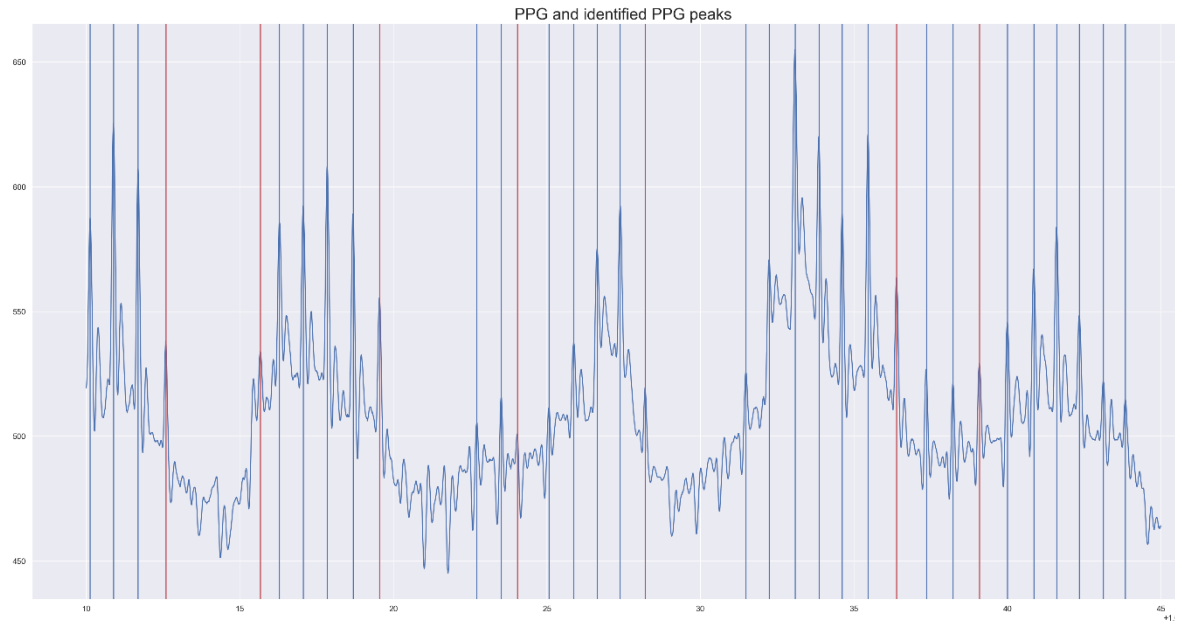


Figure 26 - Identified PPG peaks with wrong segments marked in red

6 Testing results

6.1 Preliminary ballistocardiography findings

The data from the Ballistocardiography tests are not fully processed yet, due to the time needed manually review all of the segmentation and calculations. The results of the fully processed data will be submitted to the IEEE Sensors 2022 conference. However, preliminary results for two participants are finished. The data has been processed as described in 5.2.

6.1.1 Test 1

Figure 27 shows the results of the three best sensors for the first measurement period for a 24-year-old male. The sensor in position 1 and 2 on the right foot and position two on the left foot where the sensors which successfully captured the BCG. Figure 28 shows the BCG captured on the sensor in position 2 on the left foot plotted against the BCG scale result. The shoe BCG has two noticeable differences from the scale BCG. The J peak is less pronounced, and the timing is deviating with almost 0.1 seconds.

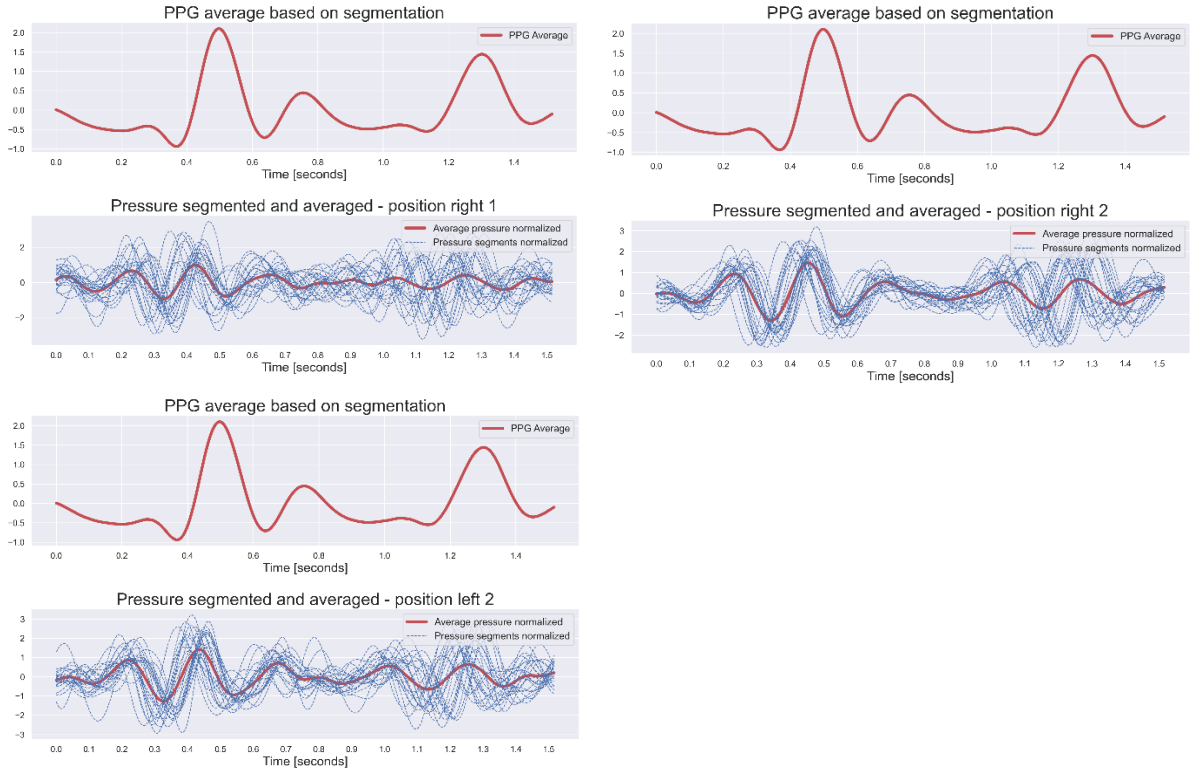


Figure 27 - Ballistocardiogram for a 24-year-old male during T1

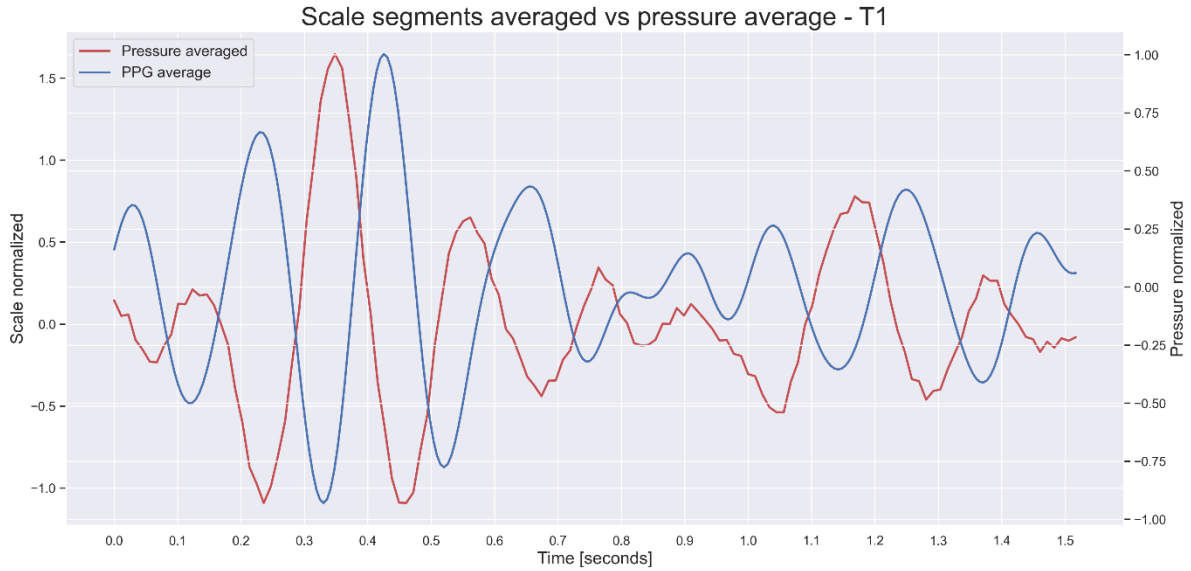


Figure 28 - Shoe BCG vs Scale BCG for a 24-year-old Male for T1

Figure 29 shows the results of the four best sensors for the first measurement period for a 23-year-old female. Position one and two on both the right and the left foot showed the best results for the first measurement period. Figure 30 shows the BCG captured on the sensor in position one on the right foot plotted against the BCG scale result. The shoe BCG has two noticeable differences from the scale BCG. The H peak is more pronounced one the shoe BCG and the timing is deviates by a bit over 0.5 seconds.

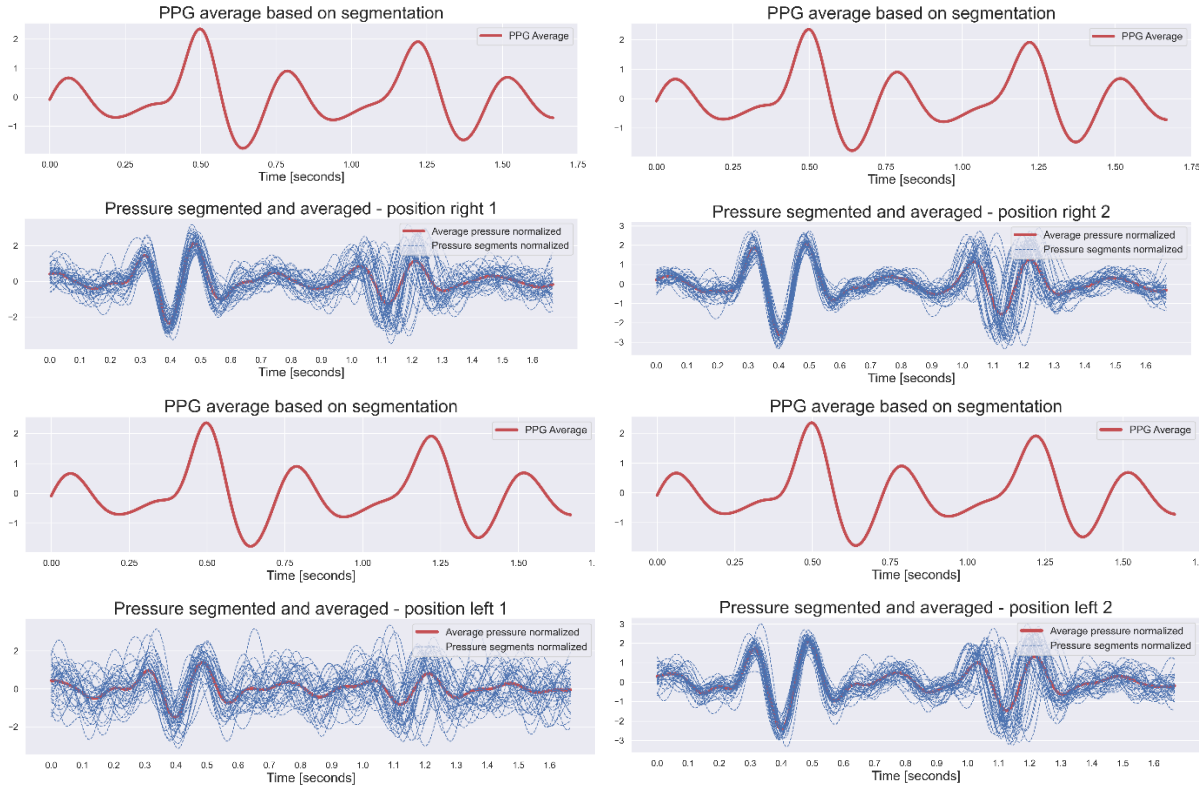


Figure 29 - BCG shoe results for a 23-year-old female

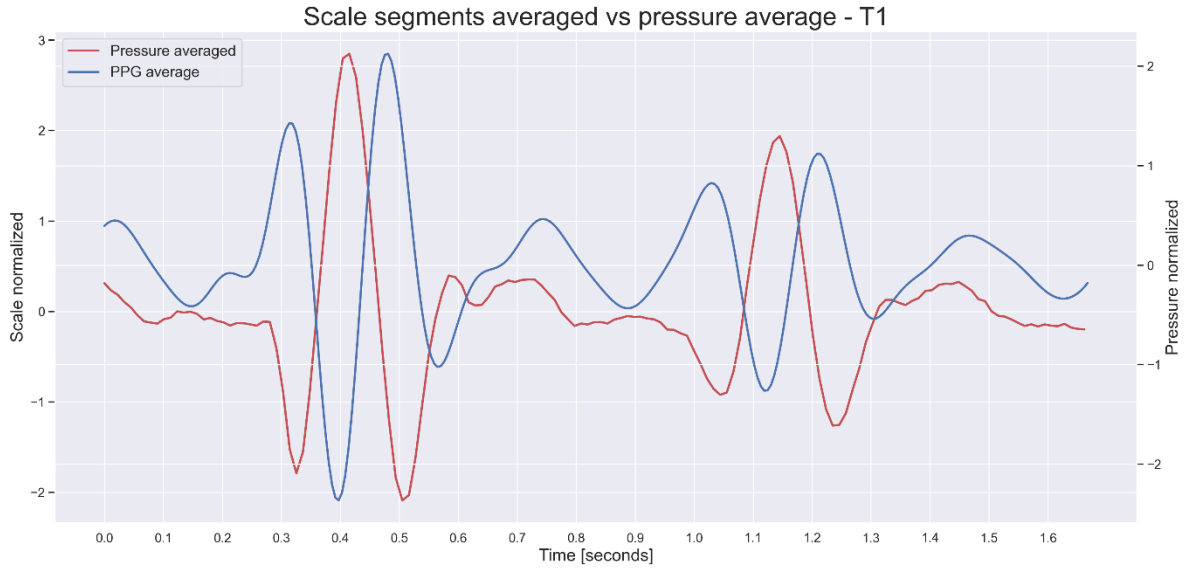


Figure 30 - BCG shoe vs BCG scale for a 23-year-old female

6.1.2 Test 2

Figure 31 shows the results of the four best sensors for the second measurement period for a 24-year-old male. The sensor in position one on the right foot and position one, two and three on the left foot where the ones who successfully captured the BCG. Figure 32 shows the BCG captured on the sensor in position 2 on the left foot plotted against the BCG scale result. The shoe BCG has two noticeable differences from the scale BCG. The J peak is less pronounced, and the timing is different with almost 0.1 seconds.

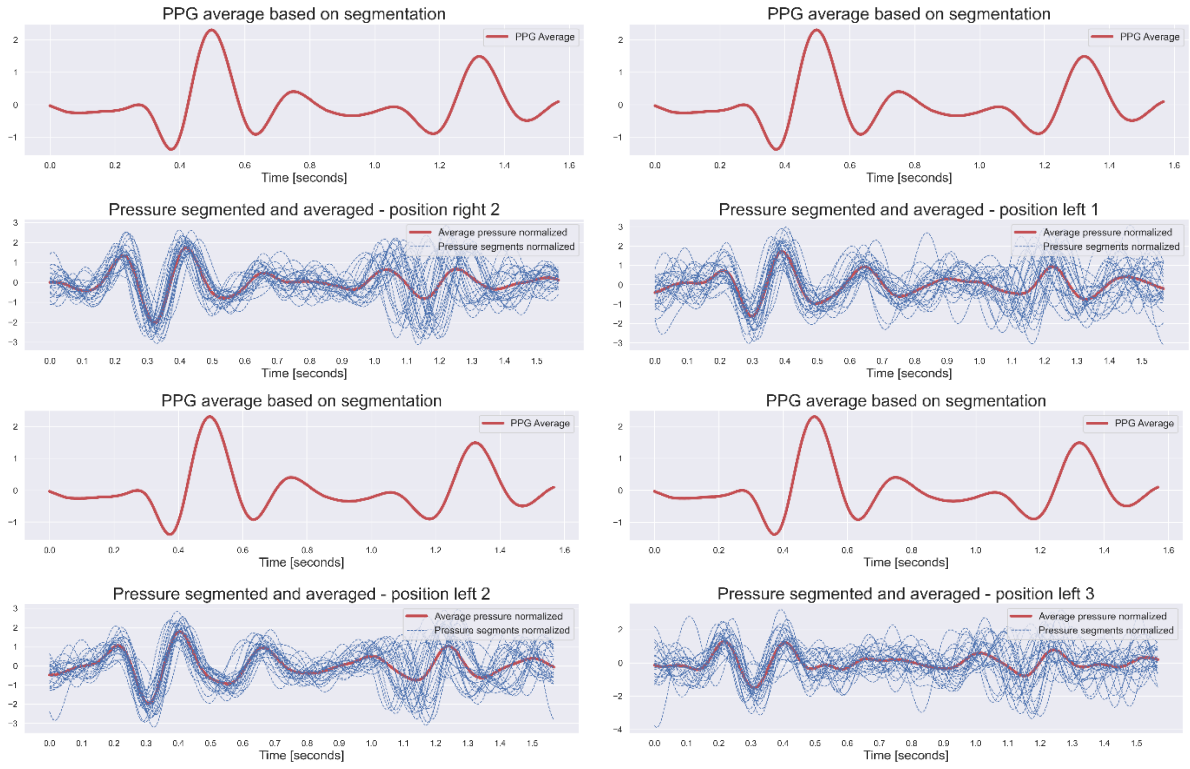


Figure 31 - Shoe BCG vs PPG for a 24-year-old male for T2

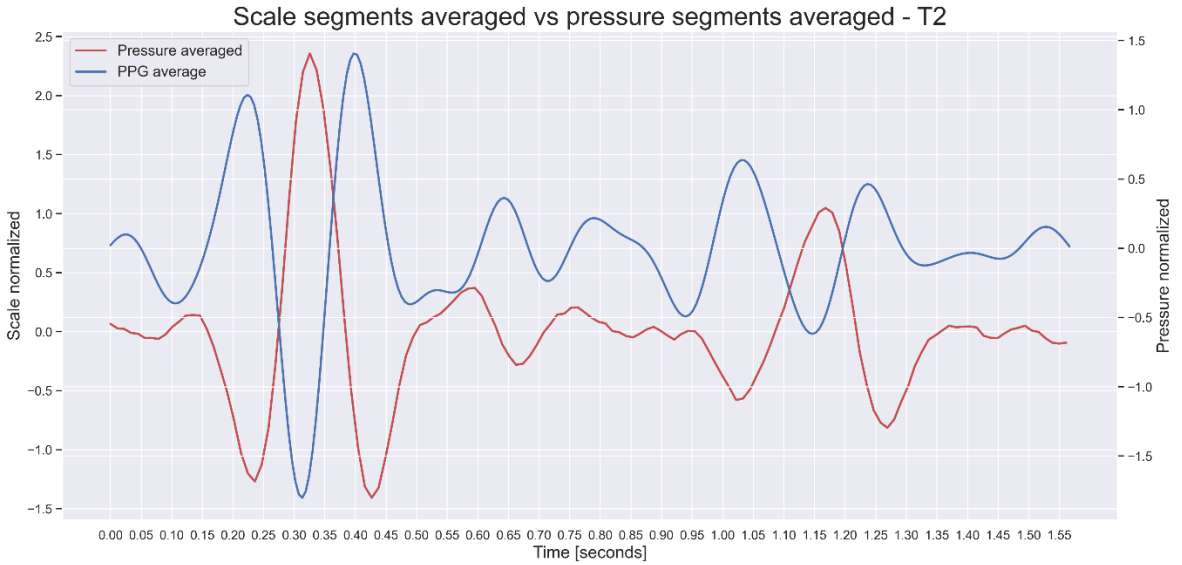


Figure 32 - Shoe BCG vs Scale BCG for a 24-year-old male for T2

Figure 33 shows the results of the four best sensors for the second measurement period for a 23-year-old female. Position one and two on both the right and the left foot showed the best results for the second measurement period. Figure 30 shows the BCG captured on the sensor in position one on the right foot plotted against the BCG scale result. The shoe BCG has two noticeable differences from the scale BCG. The H peak is more pronounced one the shoe BCG and the timing is different with a bit over 0.5 seconds.

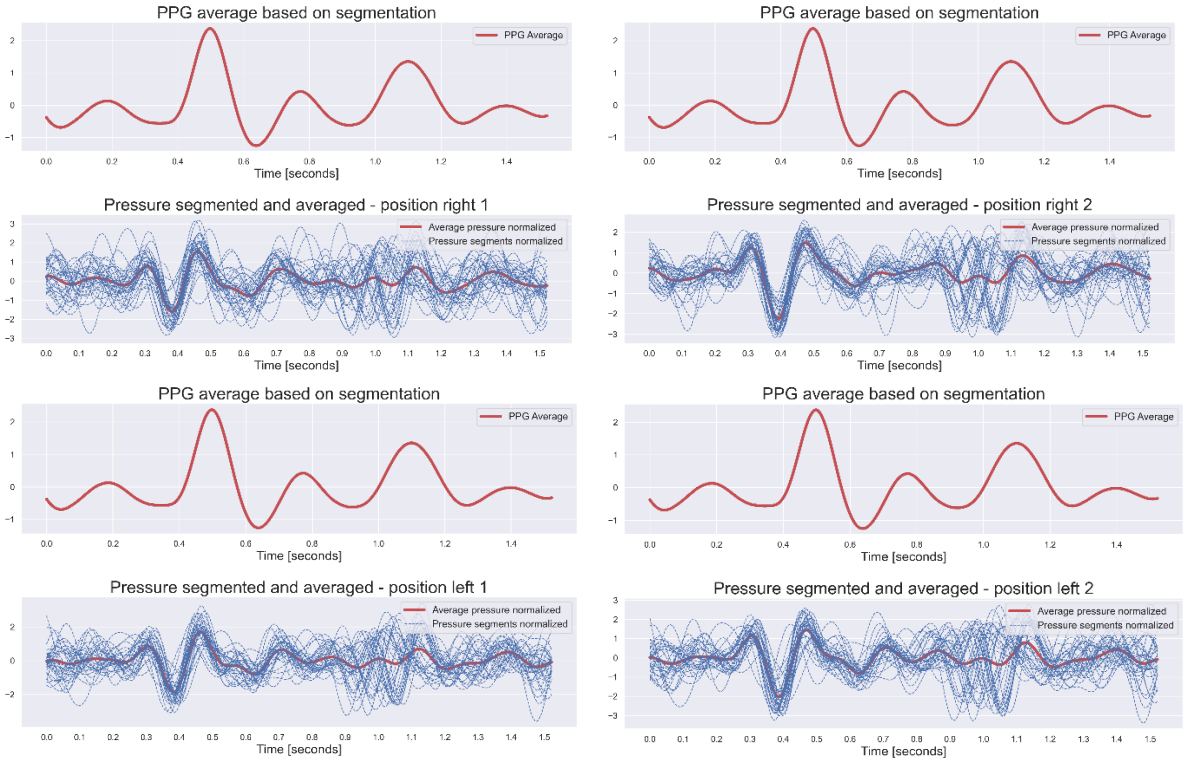


Figure 33 – Shoe BCG vs PPG for T2 for a 23-year-old female

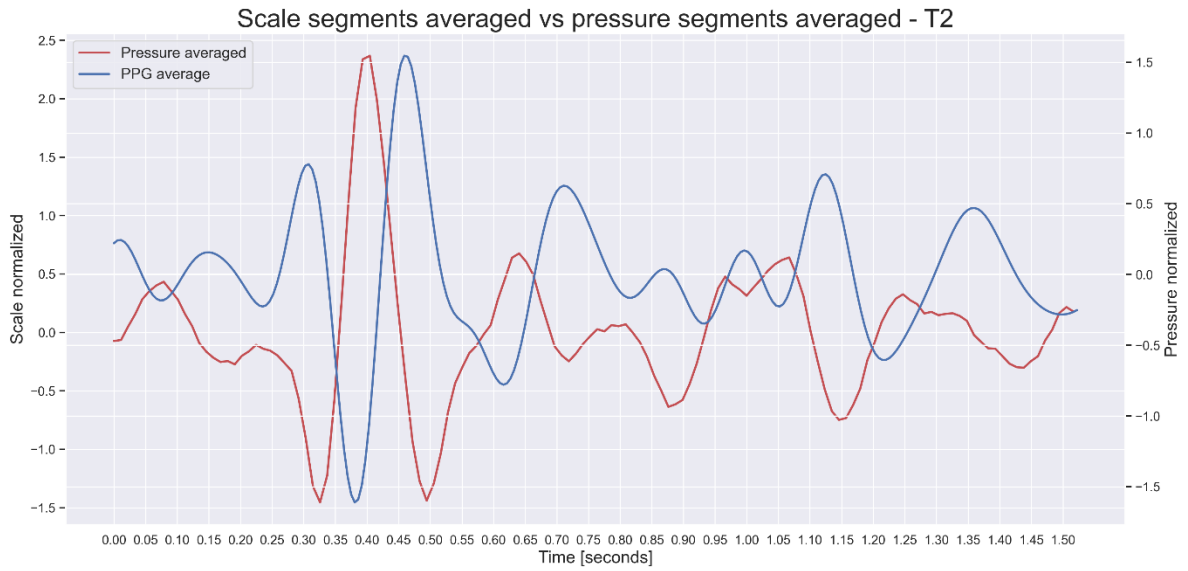


Figure 34 – Shoe BCG vs Scale BCG for T2 for a 23-year-old female

6.1.3 Test 3

Figure 35 shows the results of the four best sensors for the third measurement period for a 24-year-old male. The sensor in position one and two on both the right and the left foot where the ones who successfully captured the BCG. Figure 36 shows the BCG captured on the sensor in position two on the left foot plotted against the BCG scale result. The shoe BCG has two noticeable differences from the scale BCG. The J peak is less pronounced and the timing is different with almost 0.1 seconds.

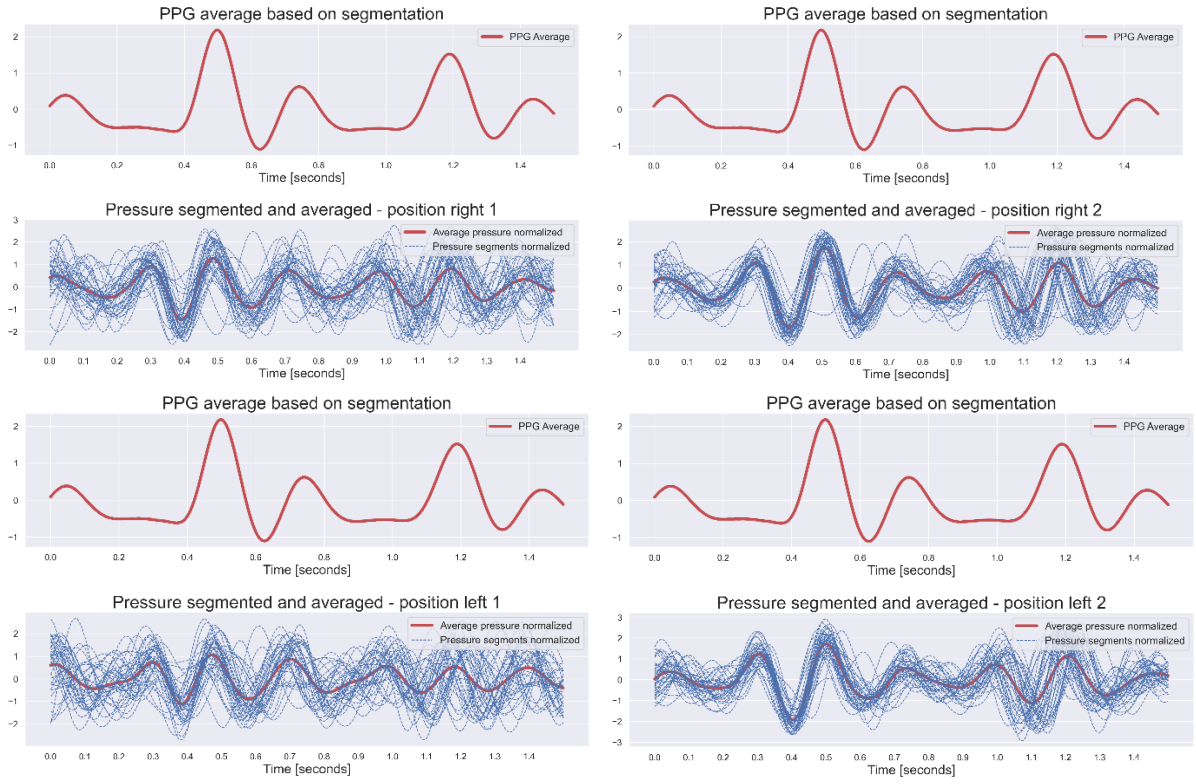


Figure 35 - Shoe BCG vs PPG for T3 for a 24-year-old male

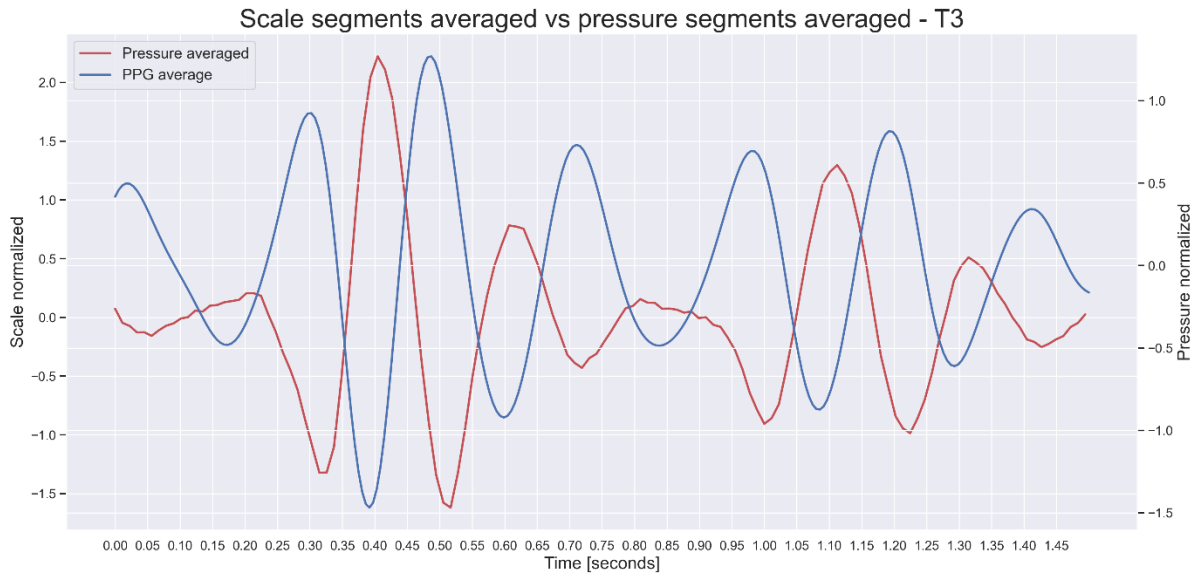


Figure 36 - Shoe BCG vs Scale BCG for T3 for a 24-year-old male

Figure 37 shows the results of the four best sensors for the third measurement period for a 23-year-old female. Position one and two on both the right and the left foot showed the best results for the first measurement period. Figure 38 shows the BCG captured on the sensor in position one on the right foot plotted against the BCG scale result. The shoe BCG has two noticeable differences from the scale BCG. The H peak is more pronounced one the shoe BCG and the timing is different with a bit over 0.5 seconds.

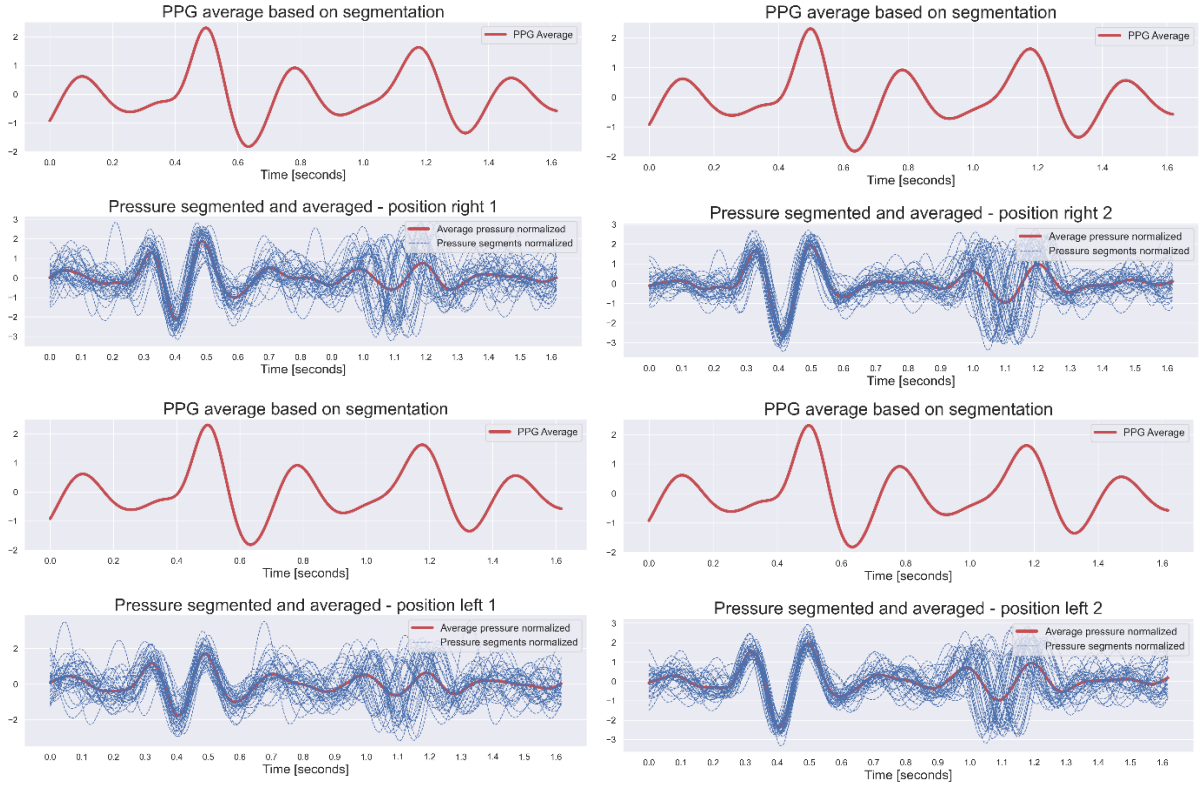


Figure 37 - Shoe BCG vs PPG for T3 for a 23-year-old female

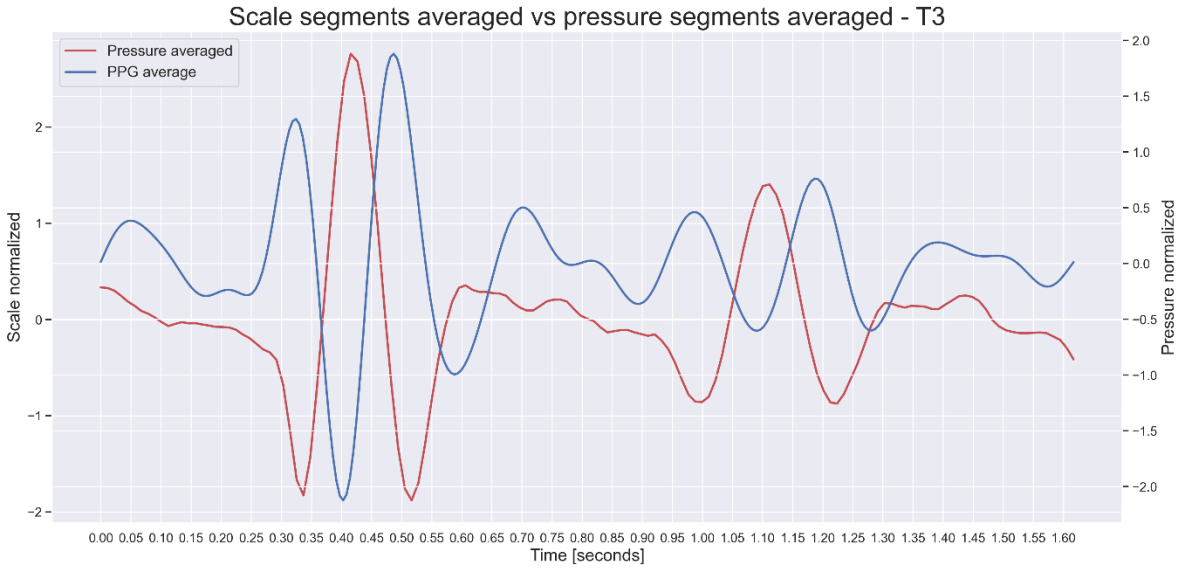
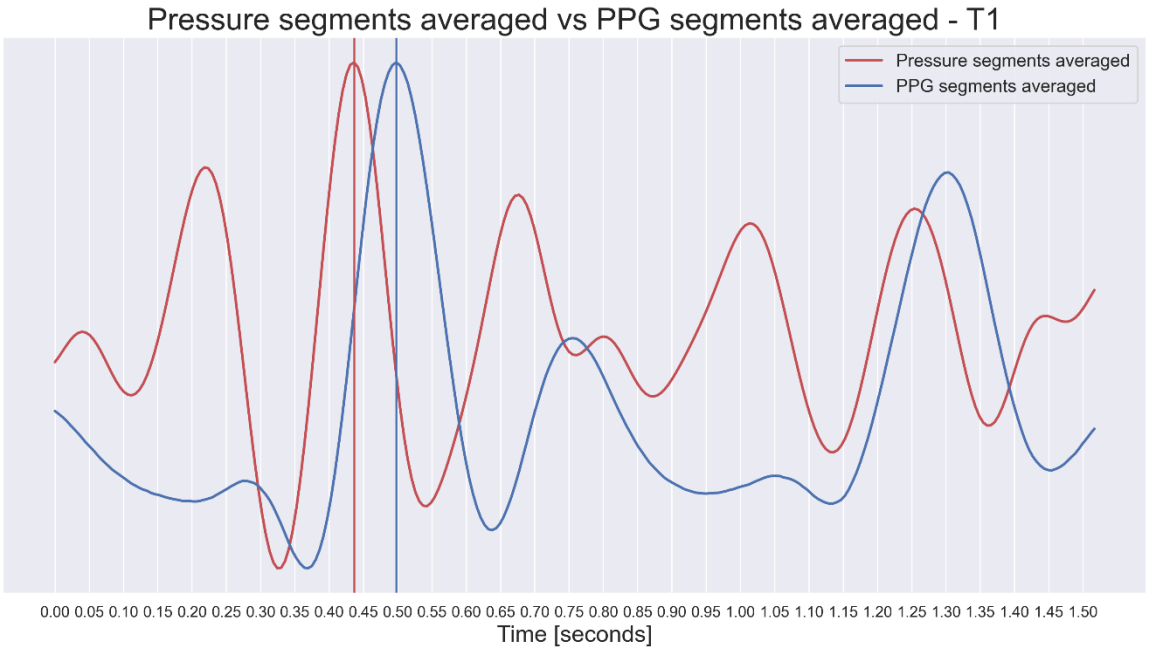


Figure 38 - Shoe BCG vs Scale BCG for a 23-year-old female

6.1.4 PTT

Figure 41 shows the detection of the J peak of the shoe BCG and the detection of the PPG peak for testing period one, two and three for a 24 year old male participant. The J peak happens at 0.436 seconds for T1, 0.403 seconds for T2 and 0.504 seconds for T3. The PPG peak happens at 0.498 seconds for all three testing periods due to the segmentation method.



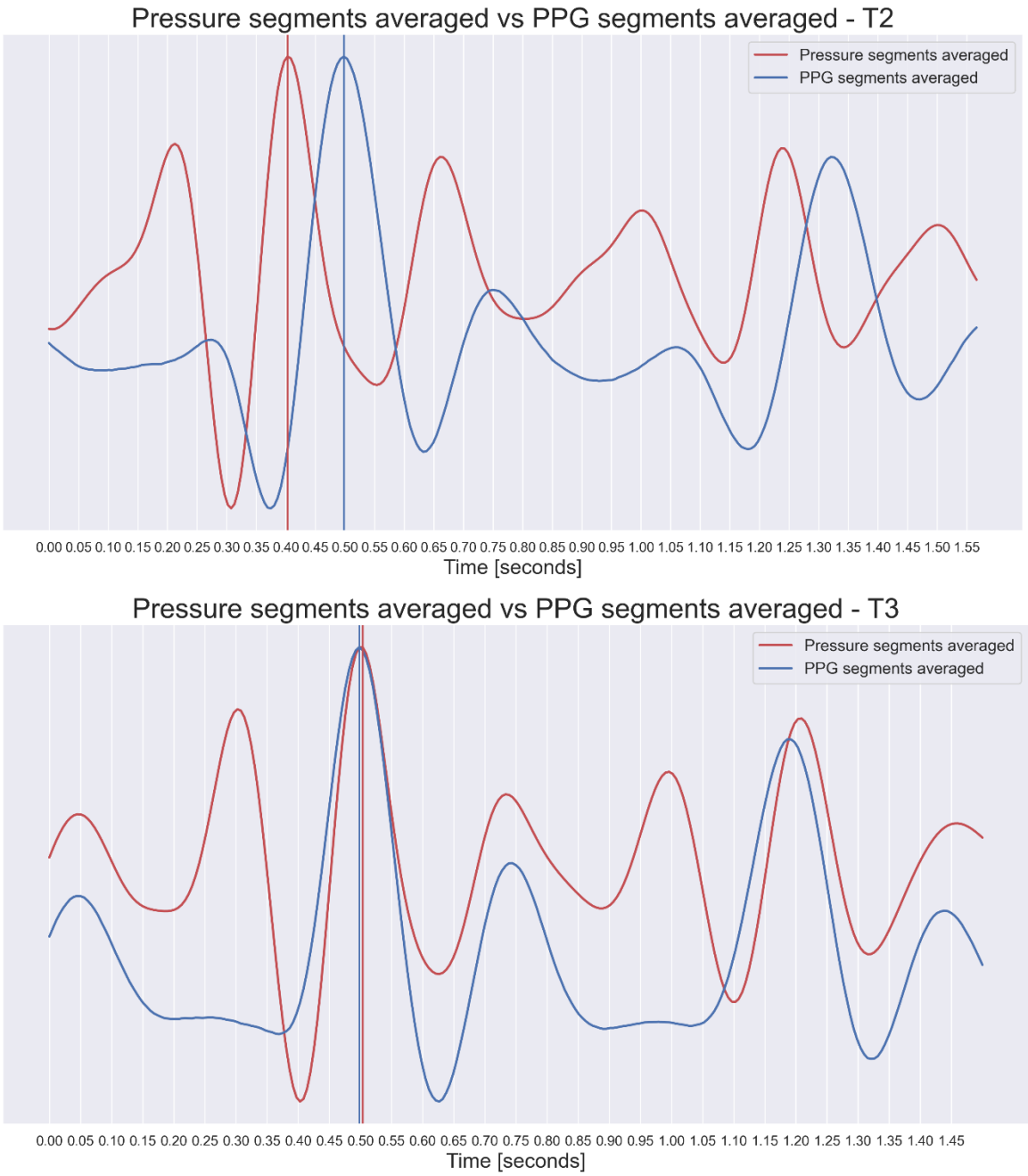
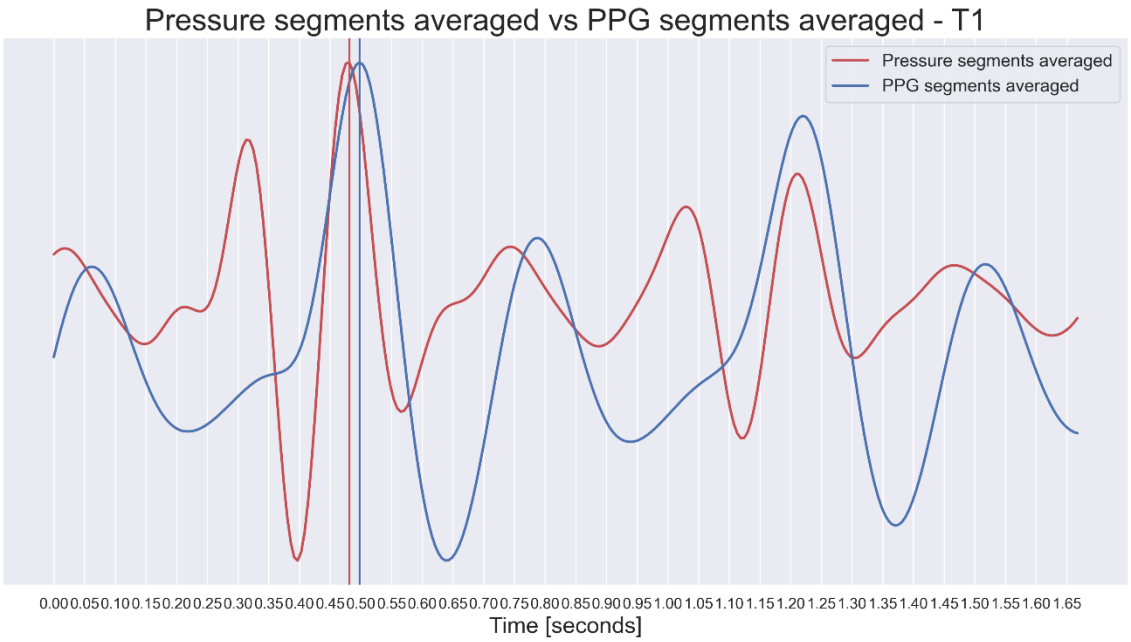
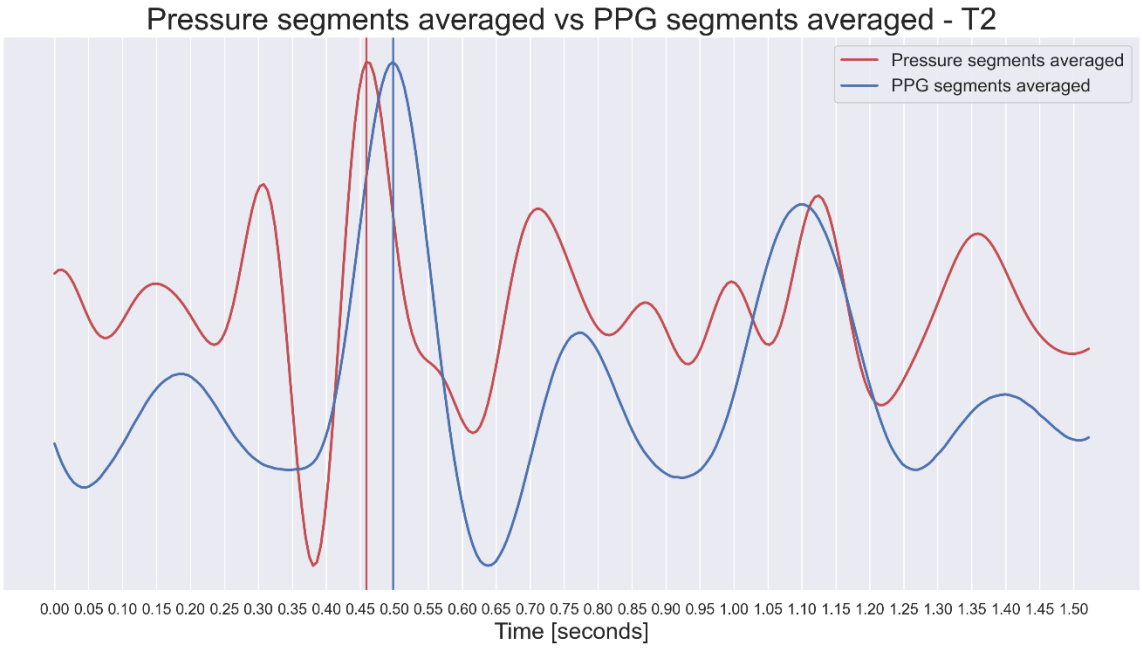


Figure 39 - Peak detection for calculating PTT change for T1, T2 and T3 for a 24 year old male

Figure 40 shows the detection of the J peak of the shoe BCG and the detection of the PPG peak for testing period one, two and three for a 23 year old female participant. The J peak happens at 0.481 seconds for T1, 0.458 seconds for T2 and 0.487 seconds for T3. The PPG peak happens at 0.498 seconds for all three testing periods due to the segmentation method.


Pressure segments averaged vs PPG segments averaged - T1
Pressure segments averaged
PPG segments averaged
0.000.050.100.150.200.250.300.350.400.450.500.550.600.650.700.750.800.850.900.951.001.051.101.151.201.251.301.351.401.451.501.551.601.65
Time [seconds]


Pressure segments averaged vs PPG segments averaged - T2
Pressure segments averaged
PPG segments averaged
0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90 0.95 1.00 1.05 1.10 1.15 1.20 1.25 1.30 1.35 1.40 1.45 1.50
Time [seconds]

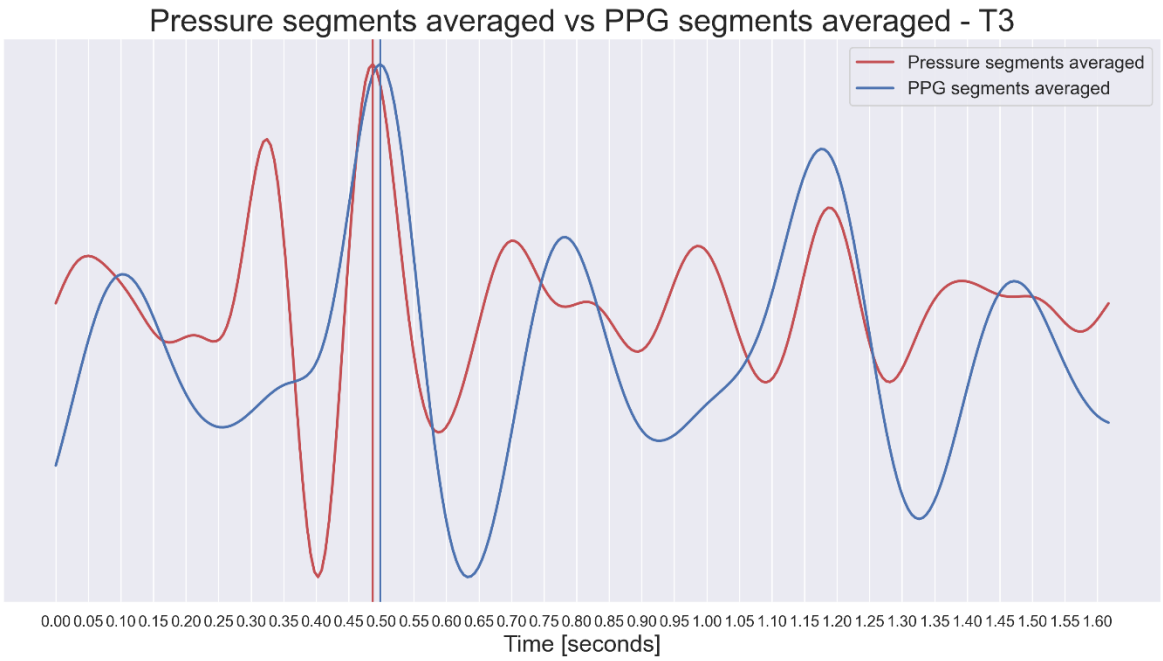


Figure 40 - Peak detection for calculating PTT change for T1, T2 and T3 for a 23-year-old female

Figure 41 shows the PTT versus blood pressure for the 24-year-old male and the 23-year-old female. The PTT and blood pressure is inversely related for both participants with the PTT decreasing in T2 when the blood pressure increases and increasing again in T3 when the blood pressure decreases.

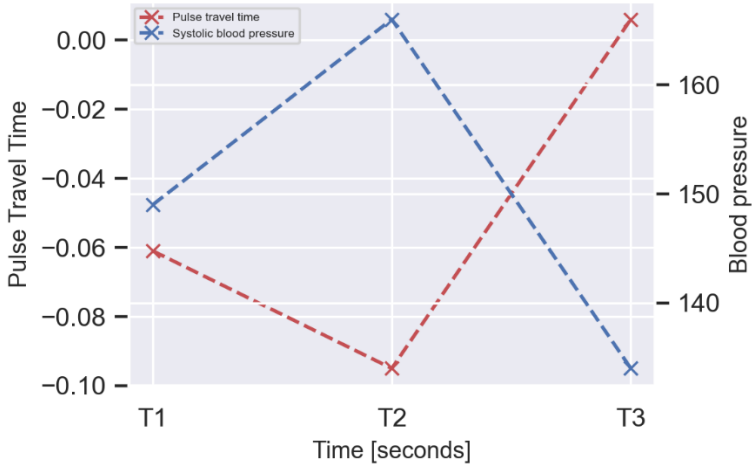


Figure 40a - PTT vs blood pressure for a 24-year-old male

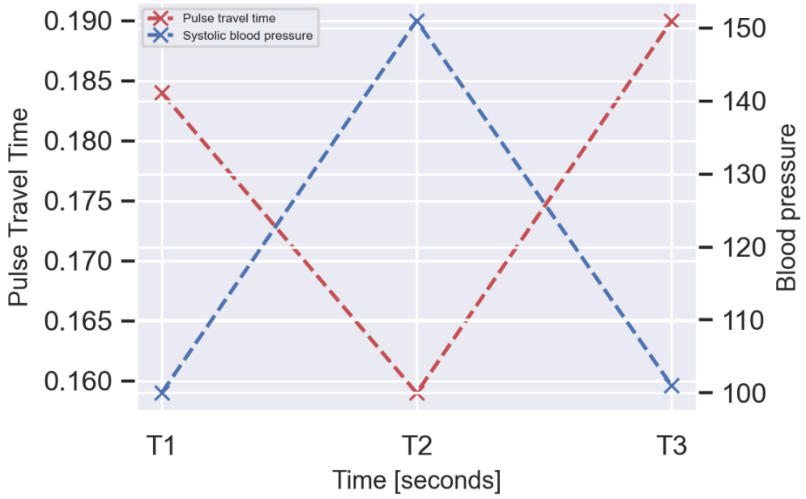


Figure 40b - PTT vs blood pressure for a 23-year-old female

Figure 41 - PTT vs Blood pressure the 24-year-old male and the 23-year-old female

6.2 Pressure distribution

Pressure distribution was briefly tested on one subject while walking and standing. Figure 42 shows the pressure distribution while walking. The different timing of the pressure sensors is apparent. Figure 43 shows the pressure distribution while standing still. However, the measurements were not calibrated, and each sensor has their own baseline value.

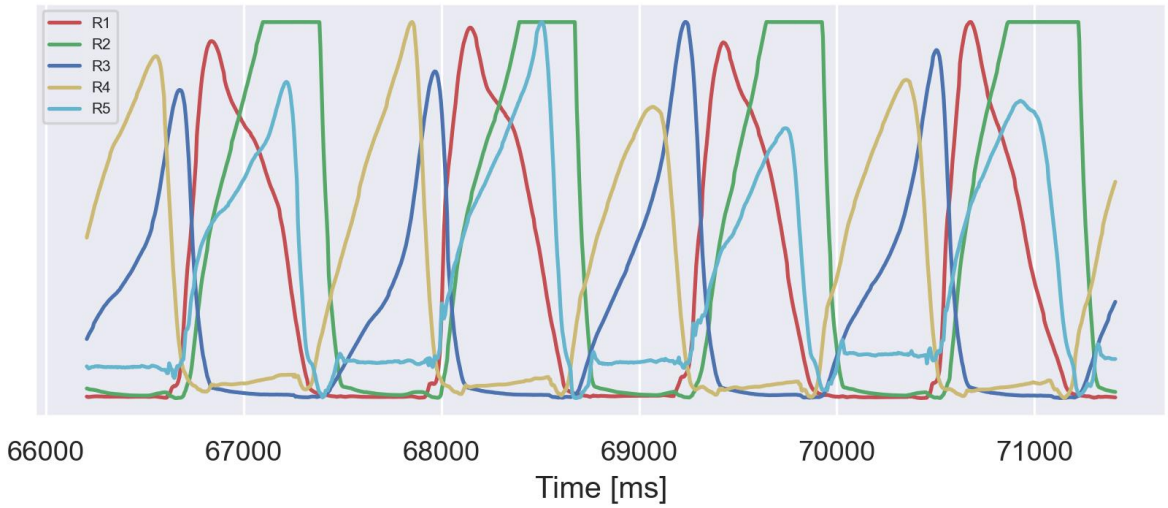


Figure 42 - Pressure distribution when walking

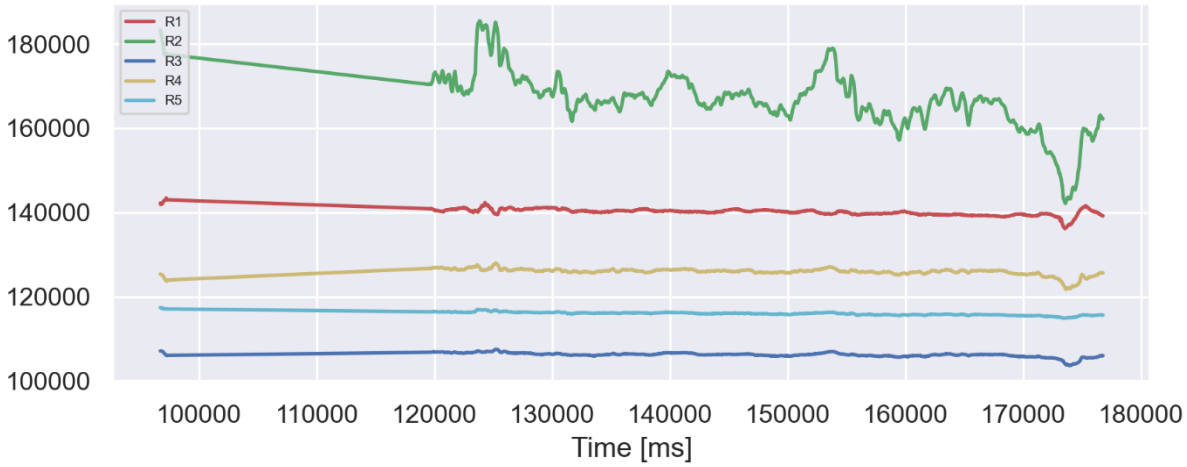


Figure 43 - Pressure distribution while standing

7 Discussion

7.1 Discussion of development process

The main tool for the development phase was prototyping as well as the bench top setup. As established in 2.3 prototyping is an important tool in product development and an essential part of wayfaring. It was used a lot during this development process and contributed greatly to learning, but some limitations were also uncovered. Due to the small magnitude of the signal compared to the considerable amount of noise the prototypes needed to be of a relatively high quality to achieve a good learning outcome. This meant that the creation of each prototype took a considerable amount of time. As quick iterations are highly important in wayfaring to efficiently explore the design space this slowed the development process. As such Wayfaring might have some limitations when applied to fields that require a high degree of quality of the prototypes. This is not to say that wayfaring and prototyping was not valuable as even the generation of low fidelity prototypes did generate some learning and design questions.

The bench top setup did generate a lot of knowledge, mostly in uncovering which parameters did not matter. Creating experimental setups which emulated the physical world in a good manner proved incredibly valuable for quickly testing and iterating through parameters. This might be because the development process was a mixture between an optimization problem and an exploratory problem. The optimization issue was larger than anticipated due to the concept appearing solid, but the results were still poor for many iterations.

7.2 Discussion of results

The results shown in 6.1 shows great promise for capturing the BCG. The findings are only preliminary as all the data has not been processed yet, but both processed datasets show huge promise. The BCG waveform is clearly captured in both participants. Compared to the scale BCG the J peak is often less pronounced with the H peak being more pronounced. Why this happens is difficult to conclude on but there might be some dampening occurring. The timing of the shoe BCG vs scale BCG is also deviating a bit by about 0.05 seconds to 0.1 seconds. This can be due to an error in the processing of data or due to some underlying difference between what is being captured. One such reason could be that the shoe BCG captures the force due to the movement while the scale BCG captures the deflection in the load cells under the scale. Some delay might be happening from the force occurring to the deflection in the load cells.

Position one and two on both of the feet is the standout sensor positions capturing the BCG in five out of six analyzed periods. In addition, sensor position three does sometimes capture the BCG. Sensor position four and five contribute little to the BCG measurement. However, some more analysis needs to be done before concluding. All 14 datasets need to be analyzed and the results for position four and five need to be further analyzed. They might capture some other part of waveform or require a different filtering process.

The PTT is calculated for both the participants and gives very promising results. The PTT decreases with increased blood pressure in T2 and increases when the blood pressure stabilizes in T3. This is exactly what is expected to happen if it is the BCG being captured. The measurement of PTT also shows a real-world application for the shoes other than capturing BCG waveform as PTT can be used for continuous noninvasive measurement of cardiovascular health to its relation to pulse wave velocity and in turn blood pressure and arterial stiffness.

However, some limitations have been discovered both in testing procedure and the results. The variation in age for the participants is very small and the variation in weight is also relatively small. As such there might be unknown effects which could worsen the results for users of different ages and bodytypes. The impact of the weight, height, shoe size and age has not been reviewed and might lead to some interesting results. The testing procedure is a controlled scenario where the participants were asked to stand still. This does not necessarily reflect a real-world scenario where there will be more motions in the feet. The window size for measuring BCG is 25 seconds which is a long time for a user to stand still in the real-world. Therefore, it is difficult to conclude on the real-world application of the shoes without before real-world testing has been done.

The pressure distribution was captured by the shoes. However, very little testing was done on the capabilities. The biggest problem with the pressure distribution is that no calibration was conducted. Each sensors have their own baseline value when no pressure is applied. Without first calibrating it is impossible to conclude whether the pressure distribution is correct. The pressure distribution while walking is interesting. The timing of the different sensors is easily captured which can show how a person is walking.

Even with the promising results from the two first datasets no conclusion can be made before all 14 datasets are processed. There might still be some unknown factors which can impact the remaining 12 datasets.

7.3 Discussion of Sole 2.0

Sole 2.0 has given very promising preliminary results. However, it does have some limitations. The water bladders are prone to breaking when walking. A sturdier version needs to be created for this to have real world applications. In addition, the bladder version might not be the most optimal solution. For instance, the piston-cylinder setup showed great promise as a sturdier and easier to produce solution. The main takeaway from the bladder solution is to only measure a small area where the forces are the largest to get the largest pressure change due to BCG. This improves the signal magnitude compared to noise and makes the measurement of BCG easier. Future work on the concept should be to further test other solutions for localized pressure measurement. In addition, the electronics need to be miniaturized to enable testing in real world scenarios.

8 Conclusion

This master thesis has created a smart shoe for measuring BCG in a noninvasive and continuous manner. The smart shoe, called, Sole 2.0 consists of a pair of shoes with five water filled bladders placed in each shoe. The water filled bladders are each connected to a BMP388 barometric pressure sensor which measure the pressure in the bladders. By measuring the change in pressure under the foot he movements of the body due related was possible to capture. The two key nuggets in designing the smart shoes were the placement of the sensors and the advantage of only measuring the force in a small area where it was the biggest. The sensor position on the heel of the foot and slightly in front of the heel were the most optimal for measuring BCG. The small surface area for measuring the pressure change in the area with the largest forces meant the pressure change was as large as possible which meant the magnitude of the signal was relatively good compared to the noise.

The smart shoes were tested on 14 participants of which 7 were female and 8 were male. Only preliminary results for two participants have been generated due to the time needed to manually validate segmentation and find the optimal measurement periods. However, the results from the two processed data sets are promising. The BCG was captured for both datasets in all three time periods. In five out of six sets both position one and two on the left and right foot managed to measure the BCG. The results were validated by manual inspection of the waveform and the computation of PTT. The waveform was similar the BCG scale waveform with a slightly less pronounced J peak and a more pronounced H peak. The calculated PTT was inversely related to the blood pressure decreasing when the blood pressure increased in T2 and increased when blood pressure stabilized in T3.

However, some limitations remain before real world use can be determined. The testing scenario was a controlled test in a lab environment. In real world there might be other aspects which affect the results such as more movement in the user and vibrations from other sources. In addition, it is impossible to conclude on the success of the results before all 14 datasets have been analyzed. The results are promising enough to conclude that the BCG shoes show promise as a method for noninvasive continuous measurement of cardiovascular health.

References

Campo, David, Hakim Khettab, Roger Yu, Nicolas Genain, Paul Edouard, Nadine Buard, and Pierre Boutouyrie. 2017. "Measurement of Aortic Pulse Wave Velocity With a Connected Bathroom Scale." *American Journal of Hypertension* 30 (9): 876–83. https://doi.org/10.1093/ajh/hpx059.

Elverum, Christer W., and Torgeir Welo. 2015. "On the Use of Directional and Incremental Prototyping in the Development of High Novelty Products: Two Case Studies in the Automotive Industry." *Journal of Engineering and Technology Management* 38 (October): 71–88. https://doi.org/10.1016/j.jengtecman.2015.09.003.

Gerstenberg, Achim, Heikki Sjöman, Thov Reime, Pekka Abrahamsson, and Martin Steinert. 2015. "A Simultaneous, Multidisciplinary Development and Design Journey – Reflections on Prototyping." In *Entertainment Computing - ICEC 2015*, edited by Konstantinos Chorianopoulos, Monica Divitini, Jannicke Baalsrud Hauge, Letizia Jaccheri, and Rainer Malaka, 409–16. Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-24589-8_33.

Giovangrandi, Laurent, Omer T. Inan, Richard M. Wiard, Mozziyar Etemadi, and Gregory T.A. Kovacs. 2011. "Ballistocardiography — A Method Worth Revisiting." In *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 4279–82. https://doi.org/10.1109/IEMBS.2011.6091062.

Inan, O. T., M. Etemadi, R. M. Wiard, L. Giovangrandi, and G. T. A. Kovacs. 2009. "Robust Ballistocardiogram Acquisition for Home Monitoring." *Physiological Measurement* 30 (2): 169–85. https://doi.org/10.1088/0967-3334/30/2/005.

Inan, Omer T., Pierre-Francois Migeotte, Kwang-Suk Park, Mozziyar Etemadi, Kouhyar Tavakolian, Ramon Casanella, John Zanetti, et al. 2015. "Ballistocardiography and Seismocardiography: A Review of Recent Advances." *IEEE Journal of Biomedical and Health Informatics* 19 (4): 1414–27. https://doi.org/10.1109/JBHI.2014.2361732.

Jensen, L. S., A. G. Özkil, and N. H. Mortensen. 2016. "PROTOTYPES IN ENGINEERING DESIGN: DEFINITIONS AND STRATEGIES." *DS 84: Proceedings of the DESIGN 2016 14th International Design Conference*, 821–30.

Jensen, Matilde B., Christer W. Elverum, and Martin Steinert. 2017. "Eliciting Unknown Unknowns with Prototypes: Introducing Prototrials and Prototrial-Driven Cultures." *Design Studies* 49 (March): 1–31. https://doi.org/10.1016/j.destud.2016.12.002.

Kim, Chang-Sei, Andrew M. Carek, Omer T. Inan, Ramakrishna Mukkamala, and Jin-Oh Hahn. 2018. "Ballistocardiogram-Based Approach to Cuffless Blood Pressure Monitoring: Proof of Concept and Potential Challenges." *IEEE Transactions on Biomedical Engineering* 65 (11): 2384–91. https://doi.org/10.1109/TBME.2018.2797239.

Kim, Jongbae, and David Wilemon. 2002. "Focusing the Fuzzy Front–End in New Product Development." *R&D Management* 32 (4): 269–79. https://doi.org/10.1111/1467-9310.00259.

Koivistoinen, T., S. Junnila, A. Varri, and T. Koobi. 2004. "A New Method for Measuring the Ballistocardiogram Using EMFi Sensors in a Normal Chair." In *The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 1:2026–29. https://doi.org/10.1109/IEMBS.2004.1403596.

Kriesi, Carlo, Jørgen Blindheim, Øystein Bjelland, and Martin Steinert. 2016. "Creating Dynamic Requirements through Iteratively Prototyping Critical Functionalities." *Procedia CIRP*, 26th CIRP Design Conference, 50 (January): 790–95. https://doi.org/10.1016/j.procir.2016.04.122.

Lauff, Carlye A., Daria Kotys-Schwartz, and Mark E. Rentschler. 2018. "What Is a Prototype? What Are the Roles of Prototypes in Companies?" *Journal of Mechanical Design* 140 (6). https://doi.org/10.1115/1.4039340.

Mack, D.C., J.T. Patrie, P.M. Suratt, R.A. Felder, and M. Alwan. 2009. "Development and Preliminary Validation of Heart Rate and Breathing Rate Detection Using a Passive, Ballistocardiography-Based Sleep Monitoring System." *IEEE Transactions on Information Technology in Biomedicine* 13 (1): 111–20. https://doi.org/10.1109/TITB.2008.2007194.

Martin, Stephanie L.-O., Andrew M. Carek, Chang-Sei Kim, Hazar Ashouri, Omer T. Inan, Jin-Oh Hahn, and Ramakrishna Mukkamala. 2016. "Weighing Scale-Based Pulse Transit Time Is a Superior Marker of Blood Pressure than Conventional Pulse Arrival Time." *Scientific Reports* 6 (1): 39273. https://doi.org/10.1038/srep39273.

Mora, Niccolò, Federico Cocconcelli, Guido Matrella, and Paolo Ciampolini. 2020. "Accurate Heartbeat Detection on Ballistocardiogram Accelerometric Traces." *IEEE Transactions on Instrumentation and Measurement* 69 (11): 9000–9009. https://doi.org/10.1109/TIM.2020.2998644.

Pereira, Tânia, Carlos Correia, and João Cardoso. 2015. "Novel Methods for Pulse Wave Velocity Measurement." *Journal of Medical and Biological Engineering* 35 (5): 555–65. https://doi.org/10.1007/s40846-015-0086-8.

Pinheiro, Eduardo, Octavian Postolache, and Pedro Girão. 2009. "Blood Pressure and Heart Rate Variabilities Estimation Using Ballistocardiography." In *In Proceedings of the 7 Th Conf. on. Telecom*, 125–28.

Starr, Isaac, A. J. Rawson, H. A. Schroeder, and N. R. Joseph. 1939. "STUDIES ON THE ESTIMATION OF CARDIAC OUPTUT IN MAN, AND OF ABNORMALITIES IN CARDIAC FUNCTION, FROM THE HEART'S RECOIL AND THE BLOOD'S IMPACTS; THE BALLISTOCARDIOGRAM." *American Journal of Physiology-Legacy Content* 127 (1): 1–28. https://doi.org/10.1152/ajplegacy.1939.127.1.1.

Steinert, Martin, and Larry Leifer. 2012. "'Finding One's Way': Re-Discovering a Hunter-Gatherer Model Based on Wayfaring." *International Journal of Engineering Education* 28 (January): 251–52.

Sutcliffe, Alistair, and Pete Sawyer. 2013. "Requirements Elicitation: Towards the Unknown Unknowns." In *2013 21st IEEE International Requirements Engineering Conference (RE)*, 92–104. https://doi.org/10.1109/RE.2013.6636709.

Vestad, Håvard, and Martin Steinert. 2019. "Piezoresistive Chopped Carbon Fiber Rubber Silicone Sensors for Shedding Frequency Detection in Alternating Vortex Streets." In *2019 IEEE SENSORS*, 1–4. https://doi.org/10.1109/SENSORS43011.2019.8956632.

WHO. n.d. "Cardiovascular Diseases." Accessed June 3, 2022. https://www.who.int/health-topics/cardiovascular-diseases.

Yao, Yang, Sungtae Shin, Azin Mousavi, Chang-Sei Kim, Lisheng Xu, Ramakrishna Mukkamala, and Jin-Oh Hahn. 2019. "Unobtrusive Estimation of Cardiovascular Parameters with Limb Ballistocardiography." *Sensors* 19 (13): 2922. https://doi.org/10.3390/s19132922.

Appendix A

Draft for the conference article to be submitted to IEEE Sensors 20222.

Windows to the Sole: Wearable Ballistocardiography Using Incompressible Fluid Sensors

Authors' Names

line 1 (of *Affiliation 1*): Dept., Organization, City, Country
line 2: (of *Affiliation 2*): Dept., Organization, City, Country
E-mail address of the corresponding author
ORCID number of the corresponding author (*optional*)

***Abstract*—**

***Keywords*—**

I. Introduction

The ballistocardiogram (BCG) is the recording of the ballistic forces generated by ejection of blood from the ventricle during the onset of systole. First rigorously investigated in the first half of the 20th century, as a non-invasive cardiovascular measurement the BCG has seen a modest resurgence as a convenient method of recording heart- and respiratory rates [1]. In combination with the photoplethysmogram (PPG), attempts have been made to define a robust estimate of the pulse wave velocity (PWV) via multi-messenger time delay [2], [3].

Partly to deal with the notoriously noisy character of BCG signals, typical measurement scenarios have often relied on stationary measurements. These have included bathroom weigh-scales, beds, and chairs ([4] + bed/chair); these "full-body" BCG setups rely on measuring force, for example via strain gauges or electret films. This has the advantage of comparatively high signal-to-noise ratio (SNR). There is however a healthcare monitoring motivation for integrating BCG-based measurements in wearable devices, to which end attention has been focused on wrist- and earbud-based approaches built on accelerometry [2], [5]. These have the strong benefit of convenience and mobility but suffer a loss of signal quality in return. Here, we propose a "best of both worlds" solution: by integrating a series of high-sensitivity liquid pressure sensors in the sole of a shoe, we demonstrate that the BCG can be reliably recorded with reasonable quality. In combination with PPG, we use ensemble-averaged waveforms to estimate pulse transit times. We verified the performance of the smart shoes by performing a cold pressor test on 15 volunteers (r_{female} = 0,46). Blood pressure was recorded concurrently using the volume-clamp method.

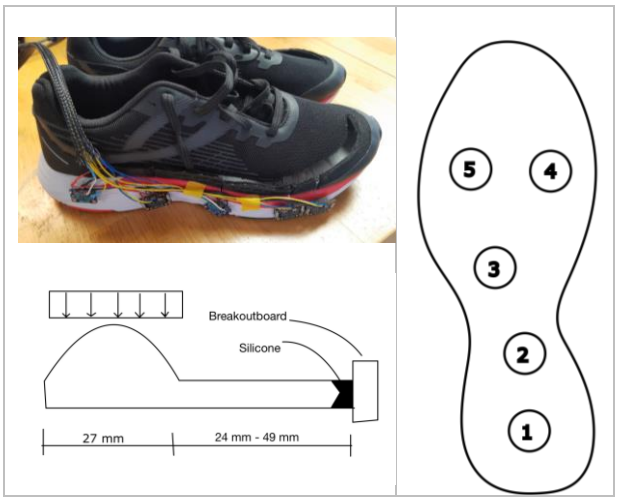
II. Method

A. Ballistocardiogram shoe design

The smart shoes consist of a standard pair of running shoes. Each of the shoes is fitted with five BMP388 pressure sensors on a breakout board from Adafruit. Pressure sensors were used due to being easy to acquire, easy to interface with and had a relatively low noise and low drift. Five sensors were found to provide a satisfactory response range.

Each sensor unit consists of one atmospheric pressure sensor (BMP388, Bosch Sensortec GmbH, Germany) connected via a flexible plastic tube to a water filled bladder which is fitted into the modified sole. The plastic tube is adhered to the sensor board, surrounding the sensor IC. The pressure sensor IC is delidded and a thin layer of silicone is applied in the end of the tube, creating a waterproof interface to the sensor. The protective silicone appeared to have little impact on the signal quality during testing. Water is used due to being incompressible which was necessary due to the large forces being applied to the bladder due to bodyweight. The BMP388s are connected via I2C multiplexer to a MCU (Teensy 4.1, PJRC).

The water-filled bladders have a diameter of 27 mm and an approximate surface area of 1550 mm^2. The bags are made from plastic bags made for welding which are cut into plastic sheets of two different sizes . The smallest with a diameter of 27 mm and the larger with a diameter of 37 mm. The two sheets are thermally welded together. Four to eight folds are then introduced into largest plastic sheet to achieve the same circumference on both of the sheets. This introduces a natural dome shape to the plastic bags which greatly simplifies the process of filling the bladder cavity.



B. Experimental protocol

Fifteen participants were included in the study of which 8 were male and 7 were female. All participants gave written informed consent after being informed of the study and its procedures. The study was approved by NSD with reference number 250185. All participants were interviewed about their gender, age, height, weight and shoesize. The results of which are shown in Table 1.

Table 1 - Interview results showing median value and range

	Female	Male
Gender	7	8
Age	23.5 (22-26)	25.5 (23-29)
Height	171 (162-175)	182.5 (170.5-193)
Weight	67 (53-93)	80 (64 – 86)
Shoe size	39 (36-40)	44.5 (42.5 – 45)

Participants were instrumented with the ballistocardiogram shoes on their feet and a PPG sensor (PulseSensor, World Famous Electronics llc, USA) on their left index finger. The labchart Nano housing unit was placed on the participants left wrist and the corresponding finger cuff (MLT382, ADInstruments, United Kingdom) for blood pressure measurement was placed on the left middle finger. All participants were asked to stand on a BCG scale which acts as a reference for the BCG shoes. The BCG shoes capture the pressure under the participants feet.

The participants were first instructed to stand still on the BCG scale for 1.5 minutes. After 1.5 minutes had passed they were asked to place their right hand in the cold water when they felt ready. The water held 5 degrees Celsius. The participants held their right hand in the cold water for 1 minute. They were then given 3 minutes to rest. After 3 minutes of rest, they were instructed to stand still on the BCG scale for 1.5 minutes.

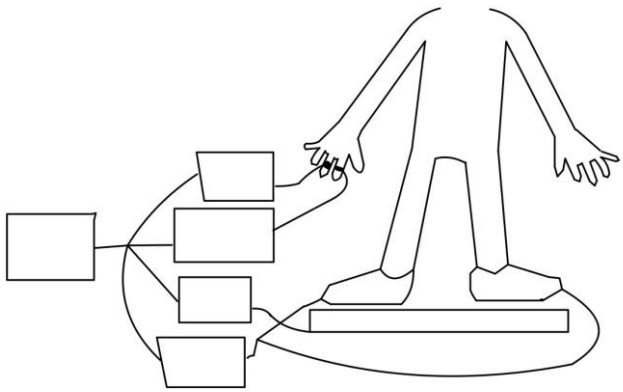


Figure 1 - Instrument setup on participants (Placeholder)

C. Data processing and analysis

The PPG, BCG shoes and BCG scale was sampled synchronously on the Teenzy 4.1 at 180 hz with the BCG scale being double sampled due to the HX711 amplifier being limited to 90 Hz in testing. The blood pressure was sampled at 200hz on the same computer as the Teenzy 4.1. The start time of testing for the Teenzy 4.1 and the finger cuff was logged by the software used. This starttime was used for synchronizing the measurements from labchart and the Arduino.

The PPG was filtered using a Savitzky–Golay filter with a polyorder of 3 and a segment size of 0.15s. This was done to remove noise and make the PPG differentiable. The pressure signals were filtered using a forward-backward digital filter using second order cascaded sections. The filter used a bandpass of [3,10]Hz and an order of 12. The scale was sampled at 90hz by removing every other sample. A corresponding PPG signal was used by removing the corresponding samples from the PPG. The blood pressure was smoothed by using a Savitzky-Golay filter with a polyorder of 3 and a window size of 35 seconds.

The PPG was used for segmenting the pressure and scale signal. After smoothing the PPG was segmented naively locating local peaks with a spacing of at least 0.5s between each peak. The segments were checked automatically by comparing segment size for each segment and the median segment size with segment sizes differing by more than 5% from the median were marked. The segmentation was also verified manually by manual inspection of peak detection and plotting each segment on each other to look for deviations. Each segment of the PPG were calculated from 0.3 seconds before the first detected peak to 0.5 seconds after the next detected peak. The BCG shoes and scale was segmented using the identified time segments for the PPG.

III. RESULTS

Figure 2 shows the results for the BCG shoe in the first measurement period for a 24 year old male with the PPG as a reference. Figure 3 shows the BCG results for the first measurement period for a 23 year old female. The waveform of the BCG is easily recognizable in both of the participants with little variation in the segments for the 23 year old female. For both participants sensor location 2 on the right shoe gave the best results. Location one and two one both feet gave the best results in all three measurement periods with position three giving good results in some of the participants. Position four and five gave no recognizable results.

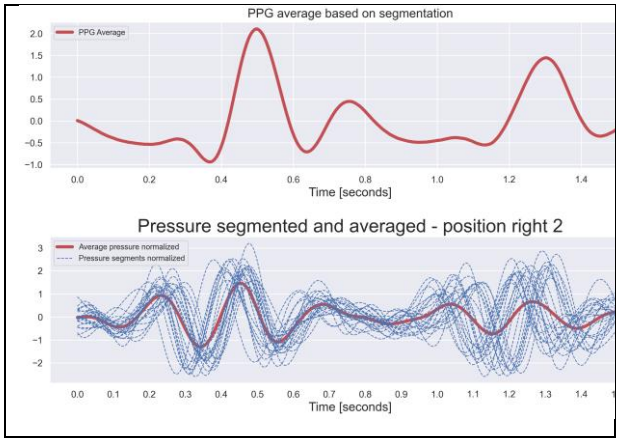


Figure 2 - BCG shoe results for a 24 year old male

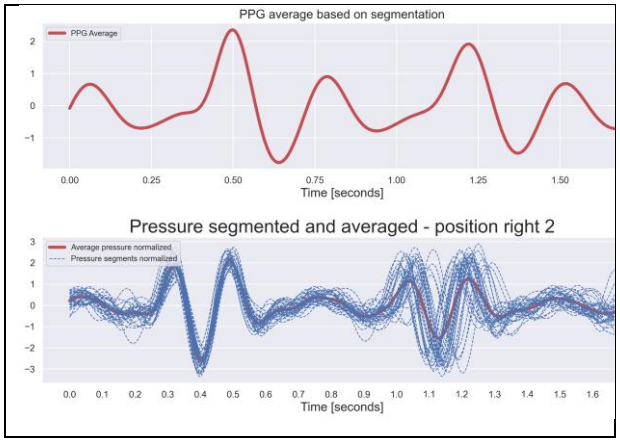


Figure 3 - BCG shoe results for a 23 year old female

Figure 4 shows the BCG shoe result for sensor position 2 on the right foot for measurement period one versus the BCG scale results. There are two notable differences between the BCG shoe and BCG scale results. The H peak is more prominent for the BCG shoe. In addition, the timing of the two BCG results is different by about 0.1 seconds.

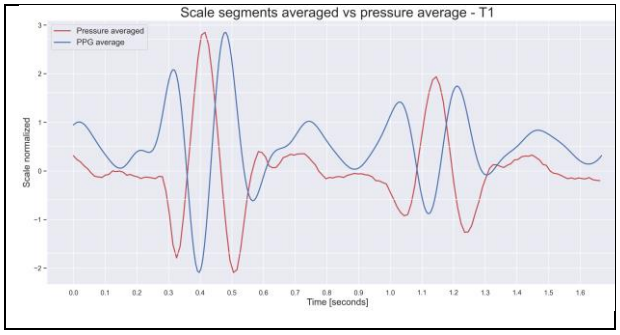


Figure 4 - The shoe BCG versus the scale BCG for a 23 year old female

Figure 5 shows the variation in pulse transit time (PTT) for measurement period one, two and three and the variation in blood pressure in the same three periods for all 14 participants. The PTT is inversely related to the blood pressure change in all 14 participant.

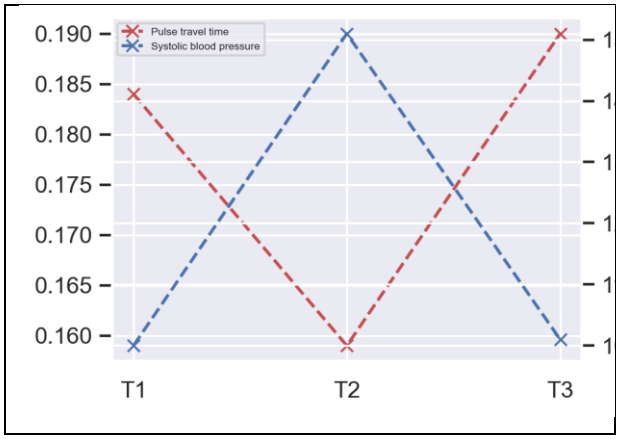


Figure 5 - Pulse transit time for the 14 participants versus blood pressure (Placeholder)

IV. Discussion

The results show that the BCG is successfully being captured by the BCG shoes. The waveform is slight different from the traditional scale BCG results with a more pronounced H peak. The sensor position one and two give the best results on both of the feet showing no major difference between left and right shoe. The PTT is calculated for all 14 participants and is inversely related to the blood pressure. This further strengthens the results in validating that it is the BCG being captured.

However, the study has some weaknesses. The participants had little variation in age and weight. It is therefore possible that some complications might arise in a broader population. In addition, the study was done in a controlled environment where the participants were instructed to stand still. Real world scenarios might have some complications which can give worse results. To evaluate whether the shoes have real world value further testing in real world scenarios would have to be done.

The design of the shoes also have some limitations. The bags are prone to leaking in when in activities such as walking and running. Only two of the five sensor positions on each foot gives consistently good results. If the only goal is measuring BCG only two sensors should be included in each shoe. The design is only one possible solution and most likely not the optimal. However, the principal of only measuring small areas with the largest effect does produce better results.

V. Conclusion

This has shown a new method for measuring BCG using smart shoes. The captured BCG has been used to calculate PTT and shown inverse relationship with blood pressure. The smart shoes show the optimal measuring area for measuring BCG under a person's feet. By only measuring in a small area where the force due to BCG is largest the magnitude of the signal is maximized to improve the signal to noise ratio. Further testing in real world scenarios is needed to further evaluate and improve the smart shoes. However, the smart shoe shows promise as a solution for wearable continuous measurement of BCG.

VI. REFERENCES

[1] O. T. Inan *mfl.*, «Ballistocardiography and seismocardiography: a review of recent advances», *IEEE J. Biomed. Health Inform.*, bd. 19, nr. 4, s. 1414–1427, jul. 2015, doi: 10.1109/JBHI.2014.2361732.

[2] P. Yousefian *mfl.*, «The Potential of Wearable Limb Ballistocardiogram in Blood Pressure Monitoring via Pulse Transit Time», *Sci. Rep.*, bd. 9, nr. 1, s. 10666, jul. 2019, doi: 10.1038/s41598-019-46936-9.

[3] S. Shin *mfl.*, «Posture-Dependent Variability in Wrist Ballistocardiogram-Photoplethysmogram Pulse Transit Time: Implication to Cuff-Less Blood Pressure Tracking», *IEEE Trans. Biomed. Eng.*, bd. 69, nr. 1, s. 347–355, jan. 2022, doi: 10.1109/TBME.2021.3094200.

[4] S. L.-O. Martin *mfl.*, «Weighing Scale-Based Pulse Transit Time is a Superior Marker of Blood Pressure than Conventional Pulse Arrival Time», *Sci. Rep.*, bd. 6, s. 39273, des. 2016, doi: 10.1038/srep39273.

[5] M. Etemadi og O. T. Inan, «Wearable ballistocardiogram and seismocardiogram systems for health and performance», *J. Appl. Physiol.*, bd. 124, nr. 2, s. 452–461, feb. 2018, doi: 10.1152/japplphysiol.00298.2017.

Appendix B

Project thesis

The master thesis is a continuation of the project thesis conducted from August 2021 to December 2021

Department of Mechanical and Industrial Engineering (MTP)

Norwegian University of Science and Technology (NTNU)

Project thesis 2021

Simon Gåseby Gjerde

Early phase development of a wearable setup for ballistocardiography

---

Supervisor(s): Martin Steinert, Federico Lozano

Co-supervisor(s): Torjus Lines Steffensen

December 20, 2021

Abstract

This project aimed to explore the solution space related to wearable physiological sensors using wayfaring and prototyping. The solution space exploration was focused on the early detection of cardiovascular diseases. Early detection and treatment have been identified as one of the most significant factors for reducing the impact of cardiovascular diseases. Wayfaring and prototyping were used to explore the design space, testing multiple different setups and technologies. Ballistocardiography was discovered as a possible solution for wearable continuous noninvasive monitoring of the cardiovascular system. Historically, issues with equipment size and motion artifacts disturbing the signal have held it back. However, computer and sensor technology advances have been promising in lessening these issues and widening the solution space. This project discovered and tested a new solution for measuring the ballistocardiogram by measuring the changes in pressure under a person's foot. The setup measures the pressure under the user's foot using a configuration of two barometric pressure sensors attached to two inflatable bags placed under the heel and front of the foot. The setup has been successfully tested in a lab environment. After filtering and averaging the signal, a clear and periodic signal with apparent similarities to the known ballistocardiogram waveform was achieved. The solution opens the possibility of measuring the ballistocardiogram in low-intensity activities such as standing and sitting by implementing a smart shoe sole. However, this method still has one of the same problems as earlier methods in terms of motion artifacts lowering the quality of the output signal. Further testing and optimization are needed to reduce the impact of this issue. In addition, further research with more participants is required to discover the maximum quality of the captured data and its potential for monitoring.

Table of contents

List of figures

Abbreviations

BCG	Ballistocardiograhpy
SCG	Seismocardiography
PPG	Photoplethysmogram
PSD	Power Spectral Density
MDF	Medium-density fiberboard
NTNU	Norges teknisk-naturvitenskapelige universitet

1 Introduction

Wearables are already impacting healthcare by enabling continuous monitoring outside of the clinical environment (Dunn et al., 2018). Many exciting solutions already exist, and the continued advancements in sensor and microcontroller technology keep widening the solution space for wearables. Common examples of wearables can be smartwatches, smart clothes, hearing aids, and smart shoes. One example of a mature solution already impacting people's lives is continuous glucose monitoring devices such as the Dexcom G5 and the Freestyle Libre 2, which significantly improve diabetes patients' ability to monitor their glucose levels.

The objective for this project thesis was to design, build and test custom sensor setups to develop the next generation of wearables.

The project scope focused on exploring the solution space for wearables monitoring the cardiovascular system. This ensured high-quality supervision throughout the process due to existing knowledge about the cardiovascular system at TrollLabs. The project's scope was also focused on developing one or more proof of concepts for sensor setups, due to the project's short time frame.

The project was developed at TrollLabs, with a focus on prototyping and wayfaring as tools for generating knowledge and ideas. TrollLabs is a research and prototyping laboratory at NTNU. It contains a multidisciplinary research group and has an overall objective of investigating and improving the fuzzy front end of engineering design. Wayfaring and prototyping are common tools and methods used TrollLabs to explore the solution space in the fuzzy front end of development.

The report is split into seven sections and can be divided into three main parts. Part one contains sections 1, 2, and 3, introducing the problem statement and necessary background theory for the project thesis. Part two of the report consists of sections four and five. They explain the most relevant work of the project thesis and present the final results from the project. Part three contains sections 6 and 7, which discuss shortcomings of the project and what can be done to address these as well as summarizing the project.

2 Prototyping theory

2.1 Wayfaring

Improvements: Language and conciseness. Low-resolution prototypes. A better explanation about why wayfaring is needed (complexity of the product development process)

The fuzzy front end of innovation is the early explorative phase of innovation before substantial resources are committed and structured processes are applied (Kim and Wilemon, 2002). Navigating the fuzzy front end can often be a challenging endeavor. One method for this navigation is wayfaring. (Steinert and Leifer, 2012) introduced the hunter-gatherer model for this explorative phase, which was later developed into the wayfaring model by (Gerstenberg et al., 2015). The wayfaring and hunter-gatherer model propose exploring the solution space as a wayfaring journey, with only a general direction or goal as a starting point. This is done by probing the solution space with iterations of a design-build-test cycle, as shown in Figure 2 (Gerstenberg et al., 2015). A probe in the wayfaring model is a cycle of designing, building, and testing an idea or a prototype, as shown in Figure 1. This is done in a converging and diverging fashion using generative design questions and deep reasoning questions. Generative design questions initiate creative and divergent thinking creating new ideas. Deep reasoning questions converge these ideas by measuring performance, feasibility, etc. This cycle creates and testes new knowledge which (Gerstenberg et al., 2015) call abductive learning. This learning is then used to perform abductive reasoning to take the next step in the solution space.

A significant advantage of wayfaring is its ability to discover unknown unknowns and dynamically create requirements. Unknown unknowns are challenges and requirements that are yet to be known (Sutcliffe and Sawyer, 2013). These can be a significant source of costs in development (Kennedy et al., 2014) and are essential to discover as early in the design process as possible. (Kriesi et al., 2016) show two examples of wayfaring being used to develop requirements iteratively, discover unknown unknowns, and avoid design loopbacks.

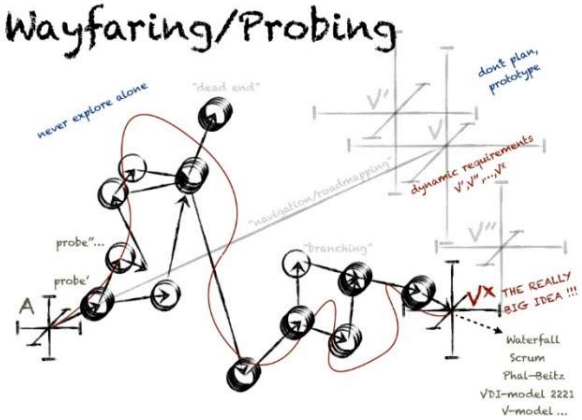


Figure 2 – Wayfaring from (Gerstenberg et al., 2015)

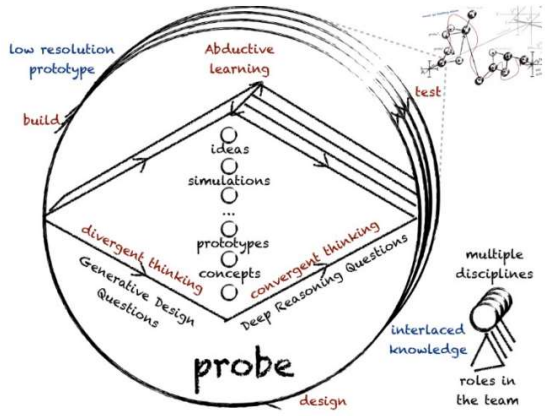


Figure 1 – The probing cycle from (Gerstenberg et al., 2015)

2.2 Prototyping

Prototyping is an essential part of product development (Jensen et al., 2016). (Schrage, 1996) stated that strong prototyping cultures produce strong products. However, the many settings and uses of prototypes have resulted in the creation of many different frameworks and principles for prototyping, with no universal framework being agreed on (Elverum and Welo, 2015; Jensen et al., 2016). Some examples of these are (Houde and Hill, 1997) who use prototypes for user-centric designs and as a tool for capturing user insights, (Lauff et al., 2018) who focus on prototypes as an aid in decision making, communicating and learning, and (Auflem et al., 2019) focus on prototypes for learning and ability to establish informed requirements in the fuzzy front end.

Prototyping is also an essential part of probing in the wayfaring model (Kriesi et al., 2016). (Steinert and Leifer, 2012) propose using prototyping as a tool for abduction and testing in their wayfaring model. (Gerstenberg et al., 2015) build on this idea by implementing prototyping as an essential part of their probing cycles. They focus on minimizing the time spent and maximizing learning by creating low-resolution prototypes testing critical functions. They also argue for creating prototypes that integrate multiple systems or disciplines to test and discover interdependencies. (Kriesi et al., 2016) further advance the use of prototypes in probing by analyzing their use for discovering critical functionalities and subsequent dynamic requirements. They present two cases where prototyping was successfully used in the probing cycles to test critical functionalities and develop dynamic requirements for the finished design based on those tests.

One last important factor of prototyping is its role in eliciting unknown unknowns and learning. Unknown unknowns are knowledge that the designer does not have and is unaware of missing(Sutcliffe and Sawyer, 2013). As stated in the wayfaring theory, discovering unknown unknowns is essential to avoid rework and reduce product development costs (Kennedy et al., 2014). The discovery of unknown unknowns relates to (Kriesi et al., 2016) method for developing dynamic requirements. They argue that prototyping lets them discover unknown problems and requirements faster than analytical methods.

3 Technical and physiological theory

3.1 Ballistocardiography

Ballistocardiography (BCG) is a method for measuring the body's movements due to the ballistic forces associated with the acceleration and deceleration of the blood in the body (Giovangrandi et al., 2011; Pinheiro et al., 2010). It was heavily studied between 1940 and 1980 but fell out of popularity due to several reasons. Technical difficulties such as device size and mechanical vibrations impacting the captured signal were one major problem, and the physiological interpretation of the signal was another major issue (Pinheiro et al., 2010). It has increased in popularity in the last two decades with modern sensors and computers opening new solutions for some of the old issues. Although its potential as a diagnostic tool has historically been lacking, its potential as a wearable to enable continuous monitoring of the overall health of the cardiovascular system in non-clinical settings is promising (Giovangrandi et al., 2011).

There exist many solutions for measuring BCG today. For instance, (Mora et al., 2020) use a triaxial accelerometer attached to a bedframe to capture the ballistocardiogram from a lying subject. (Koivistoinen et al., 2004) use EMFi sensors, an electromechanical film, applied to a standard chair to record the ballistocardiogram. (Inan et al., 2009) use a modified commercial electronic scale to capture the BCG signal. (Pinheiro et al., 2009) was able to use a chair-based BCG system in combination with ECG and PPG to noninvasively measure heart rate variability and pulse arrival time with BCG showing promise as a substitute for ECG.

One complication with using BCG is the variation in the waveform depending on the measurement method and individual differences (Inan et al., 2009; Pinheiro et al., 2010). One waveform used as a reference for the ballistocardiogram waveform is the Starr BCG (Pinheiro et al., 2010) seen in Figure 3. The most prominent features of the BCG waveform are the J-peak and the W-shape the HIJKL waves create.

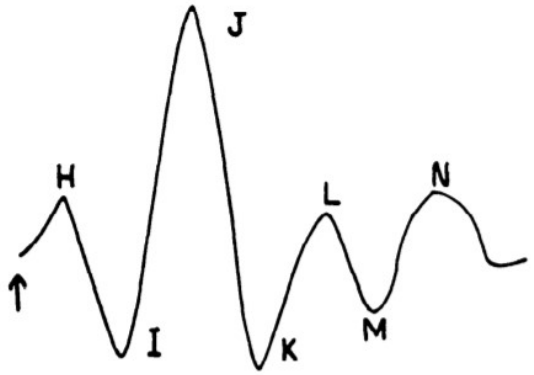


Figure 3 - Starr BCG waveform (Starr et al., 1939)

3.2 Simple signal analysis of physiological signals

Signal processing is important for analyzing physiological signals as many measurement methods also capture a significant amount of noise. This project has mainly used the power spectral density (PSD) plot and wavelet filtering to analyze the results. The power spectral density plot shows the energy of the signal for every frequency captured in the signal. By inspecting a PSD plot, it is possible to infer which frequencies the signal contains and their prominence.

Filtering using wavelets is done by decomposing the captured signal using wavelets and then reconstructing the signal with only the wanted frequency components. A wavelet is a waveform with a limited duration and an average value of zero. Wavelet decomposition using the discrete wavelet transform yields a set of approximation and detail coefficients. The detail and approximation coefficients relate to a specific frequency region depending on sampling rate frequency. The signal can be reconstructed using the inverse discrete wavelet transform. The signal is filtered by only including the detail and approximation coefficients relating to the desired frequency region.

4 Project work

This section explains the process of the project and is split into two main parts. Part one details the early wayfaring process towards an interesting concept or idea. Part two details the process of developing that initial idea or concept into a proof-of-concept prototype. However, it is important to note that the outlined process in this chapter is a simplified, more linear version of the actual process. For instance, Figure 4 shows a somewhat linear approach with technology as the driving force. In practice, the technology was not the only driving force. Interest in pulsewavevelocity (PWV) was as much driven by research into arterial stiffness and the interest in PWV drove interest in using PPG sensor for that purpose.

4.1 Part 1

As stated in the introduction the task was to build, design and test sensor setups for the next generation of wearables. Due to existing expertise in TrollLabs the solution space was focused mainly on wearables related to the cardiovascular system. This was to ensure better supervision. The preferred tools and method for exploring the solution space were benchmarking, wayfaring and prototyping. Figure 4 shows a simplified version of the wayfaring journey of the early phase exploration.

The exploration of the solution space was done in three related but different ways all united by the wayfaring journey. One method was benchmarking existing wearable sensor setups, another method was prototyping these setups or ideas for setups and the third was reading up on common cardiovascular diseases and their symptoms. This "three pronged" approach was done as each method had its advantages and disadvantages.

Researching common cardiovascular diseases was done for two main reasons. The first was that it was clear early in the project that a more thorough understanding of the cardiovascular system was needed. Most research articles, books and papers within a specific research field have their own "language" and being familiar with this language is important for effective research and communication. Therefore, starting at more basic leve focusing on learning about the cardiovascular system and its common diesases was necessary. The second reason for this approach was that a thorough understanding of the cardiovascular system and its diseases gave two important insights. The first is that a thorough understanding of the physical aspects enabled a better understanding of possible ways to measure the related physical signals. The second insight was understanding the "need" for the monitoring. Understanding which diseases were the most common and severe also gave an understanding of which diseases would have the biggest impact if prevented or treated early. One example of a common and severe disease is coronary artery disease (CAD). CAD occur due to buildup of plaque in the walls of the arteries supplying the heart with blood (CDC, 2021). This gave rise to the interest in researching pulse wave velocity and arterial stiffness.

Prototyping is one of the essential parts of the wayfaring model and based on personal experience one of the most effective ways of exploring the solution space. However, a very important aspect of prototyping is choosing the right resolution for the prototype, as mentioned in section 2.2. Early in the exploration phase it was discovered that due to the difficulty in capturing physiological signals a significant resolution was needed for the

prototypes. Simple testing such as learning how sensors worked and checking their capabilities was easily done through prototyping, and sometimes prototyping setups also worked. Two examples of this were applying silicon to a BMP 280 pressure sensor and testing its ability in capturing pulse, which worked really well, and strapping the sensor to the chest in an attempt to capture the seismocardiogram. This prototype did show promise in capturing the seismocardiogram but was not tested further due to similar solutions existing. But many times, evaluating a setups capability in measuring a physiological signal was challenging without building high resolution prototypes which would be too time consuming. In those situations, benchmarking other solutions was frequently used.

Benchmarking was mainly done through reading papers and examining existing commercial solutions. It was quickly evident that the most common commercial solutions, such as smartwatches, are well explored and not of much interest to this project. However, a lot of scientific research has been done on wearables and as such there existed many interesting state of the art solutions to benchmark. Benchmarking also helped identify were there was a gap in knowledge or execution. Ballistocardiogram was discovered as a potential interesting subject as a combination between using benchmarking and previous knowledge from my co-supervisor candidate at TrollLabs. It was a signal with a fair bit of interest in the literature and a clear cap in terms of viable solutions for wearable continuous monitoring.

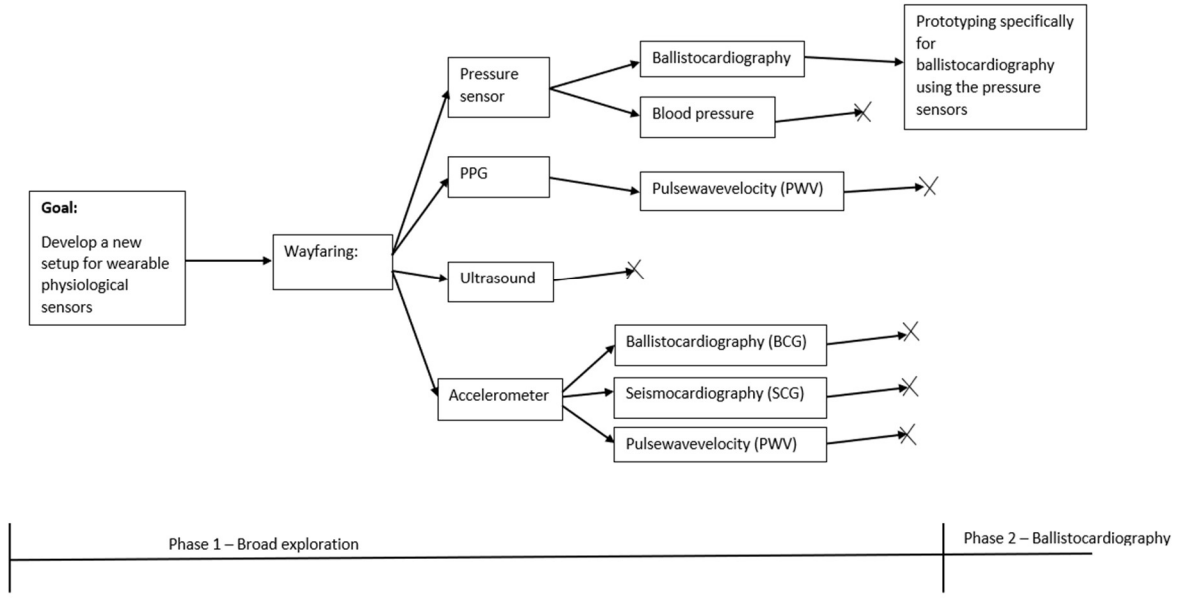


Figure 4 - Simplified wayfaring journey

4.2 Part 2 – Converging on BCG

At some point in the wayfaring process measuring BCG using pressure sensors was seen as an interesting concept. The increased focus on BCG was for a couple of reasons. One was that BCG was in literature seen as an interesting concept with large potential as a wearable setup. However, no definitive concept for wearable BCG was identified. As such a gap was identified. Using pressure sensors was interesting as in principle it would be similar to using an electronic scale, but with a potential for easier wearable integration in for instance a smartshoe. Once this converging started prototyping also ramped up and two main ideas for measuring the pressure was developed.

The first was using pressure sensors with silicon attached. The first idea was quickly proven to not be viable as the pressure sensors could only handle a certain amount of pressure and the force had to be fairly directly on top of the pressure sensor otherwise the silicon would distribute the force to the breakout board and other components.

The second idea was using air-filled bags attached to a pressure sensor under the user's foot. The first prototype testing this idea is shown in FIGURE NUMBER HERE. This was mainly a prototype to gain experience with the practical aspects. For instance, figuring out a way to attach the sensor non-permanently so it could be swapped out to other prototypes later and testing how difficult it is to stand still on an air-filled bag. After prototype 1, two more prototypes were quickly created. One using a shoe sole and one using a thin foam. The foam and shoe sole were included to reduce the amount of air in the bag. Prototype 2 was also tested in an existing running shoe, to test how this changed stability and look for unknown unknowns. None of these prototypes showed any promising results when analyzing the signal from the pressure sensor. Prototype 4 was meant to reduce the surface area in the bag compared to the force applied by standing on it with both feet. This however turned out to be too unstable stand on and showed that when the surface area was reduced stability was also reduced. While creating prototype 4 some more research was also done on the magnitude of the force applied to the air-filled bags due to the ballistic forces. Based on (Inan et al., 2009) it was estimated that the maximum force would be around 1-4N. An assumption was made that the pressure changes in the bag could be estimated with $\Delta P = F/A$, where F is the ballistic forces and A is the surface area of the bag. To reach a measurable pressure difference the surface of the bag had to be cut significantly.

Figure 5 - Prototype 1

Figure 7 - Prototypes 2-4

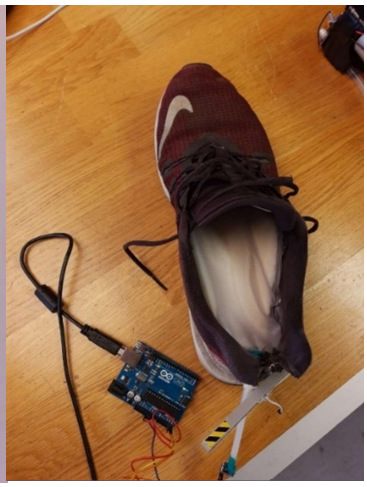


Figure 6 - In shoe testing

After learning how small the force affecting the bag due to the ballistic forces some significant changes were made. The pressure sensor was switched from a BMP280 to a BMP388 which changed the relative pressure accuracy to 0.08hPa from 0.12. As well as changing the sensor the design of the bag was significantly changed to reduce the surface area. Prototypes 5 and 6 were the first to use two small bags as contact points. The idea was to have the heel and front of the foot as contact points and the rest of the foot in the air. P5 did not work due to wrong dimensions. Prototype number 6 did work but was unstable and the pipe connecting the two bags was prone to leaking in the seams between pipe and bag. At this point an idea to create these bags using a soldering iron instead of the proprietary tools with the bags was thought of. This greatly increased the design freedom and sped up prototyping. Prototype number 7 did not have the correct dimensions to stand stable on. It was discovered that the bags had to be at very specific points under the heel and front of the foot to be stable to stand on. Prototype number 8 fixed this issue and gave some very promising results from the signal from the pressure sensor. When using spectral power density to analyze the power of each frequency of the signal it showed activity in the area between 1 and 10 Hz as shown in Figure 8. This was a clear indication of the concept working. However, the signal was still very noisy, and it was clear it needed improvement to have any practical use. A PPG sensor measuring the pulse at the ankle was introduced in the setup as shown in Figure 9. This made it possible to segment the signal from the pressure sensor into the pulse periods by comparing with the PPG sensor output. Prototype 9 was an attempt to reduce the area to force ratio by standing with both feed on the bag, but just as in prototype 4 standing with both feet on the same bag proved too unstable.

Figure 10 - Prototypes 5-9 from left to right and top to bottom

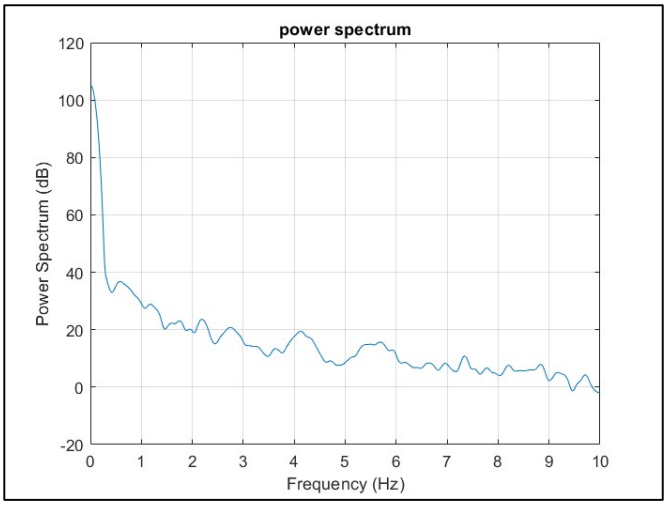


Figure 8 - Power spectrum density for prototype eight

Figure 9 - Setup for testing prototype eight with PPG

After prototype 8 showed promise more divergent thinking was needed to reduce the surface area further. An idea was to introduce a stiff plate the user would stand on with the bags under the plate. This greatly improved the possible design space for the bags and was a game changer. Lots of testing was done and some key requirements were discovered. The biggest was that the height of the bag had to be sufficient otherwise the plate would touch the ground and disturb the signal. Figure 13 shows an example of a design with a very small surface area but not enough height. In general, a too low surface area to volume ratio meant the plate would touch the ground. Another was that the plate easily damaged the connection to the pressure sensor. Therefore, the pressure sensor connection had to be moved away from the plate. The solution to both issues was to move away from the design with the two connected bags and instead use two bags and two pressure sensors, with two such bags shown in Figure 12. This added another potential benefit in that they could also show the pressure difference between the front

and back of the foot which could turn interesting later. Figure 11 shows an example of the final setup. This setup is further explained in section 5.

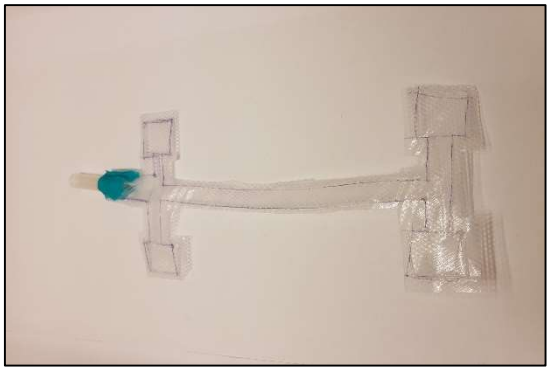


Figure 13 - Shape of the bag with insufficient height

Figure 12 - Correct bag shape

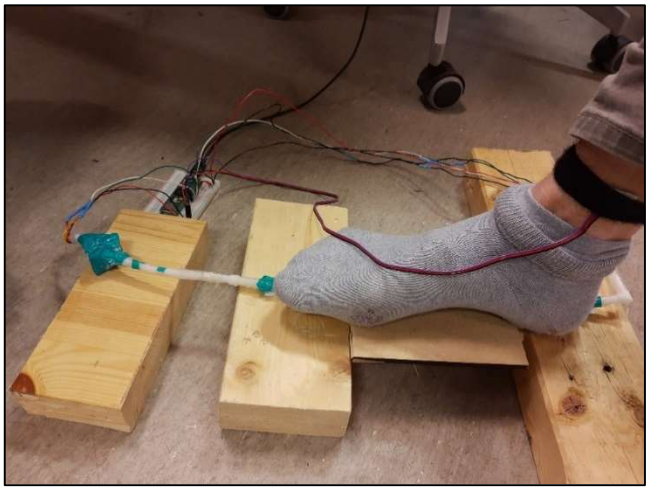


Figure 11 - Final testing setup

5 Final setup and results

The final setup consists of two BMP388 pressure sensors attached to two air filled bags. The user stands on a stiff plate made of MDF with one air filled bag under the heel and one air filled bag under the front of the foot. In addition, a pulse sensor, from World Famous Electronics llc, is attached to the ankle of the user. Figure 14 shows all components when not in use and Figure 11 shows the setup in use. Wiring schematic is shown in Figure 16. As shown in Figure 11 during lab testing the air-filled bags were placed on a wooden slab each and the stiff plate was suspended between the wooden slabs to ensure no contact between the ground and the MDF plate was made.

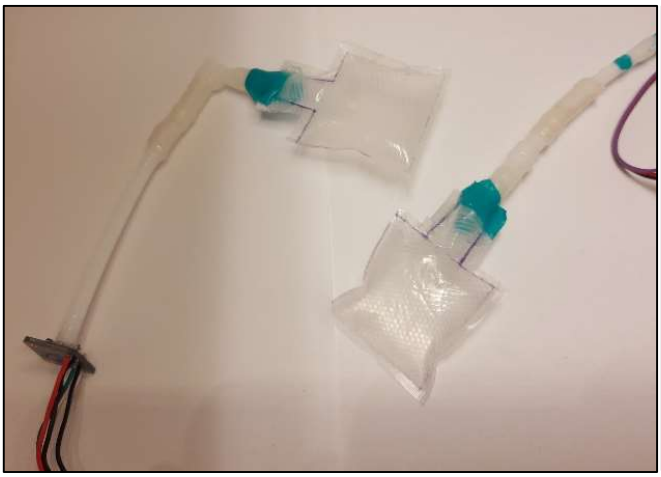


Figure 15 - Final bag design

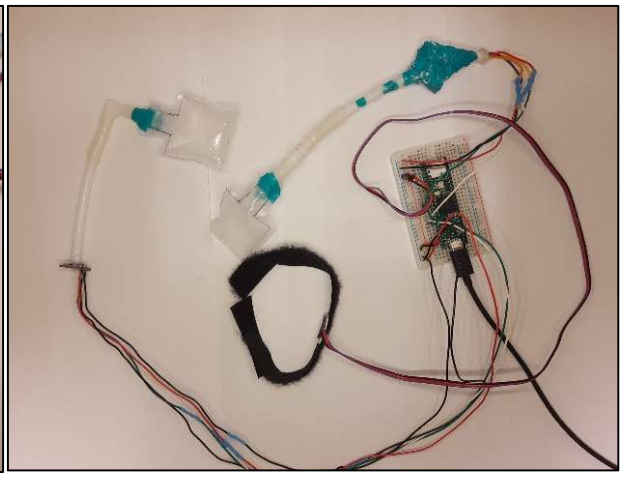


Figure 14 - All parts of final setup

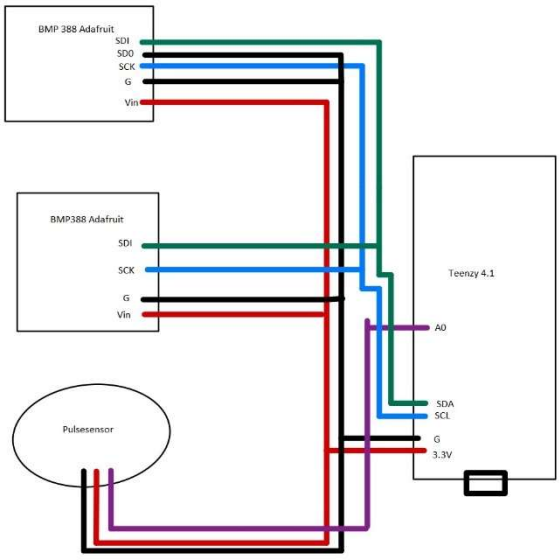


Figure 16 - Simple wiring schematic

Due to how noisy the signal still was the final results were then processed. Both the pulse sensor signal and the pressure sensor signals were decomposed using wavelet decomposition and rebuilt with only the relevant frequencies remaining. Then the pulsesensor signal was segmented by identifying the peaks of the signal with a minimum distance between each peak. This is a very naïve method that is dependent on the signal being clear. The identification of peaks was also confirmed with a visual inspection to ensure no mistakes were made. The pressure signal was then segmented using the time segments identified by the pulse segmentation. The mean signal of the pressure signal was computed using the segments. Figure 17 shows the pressure segments measured from the front of the foot with the mean signal overlayed.

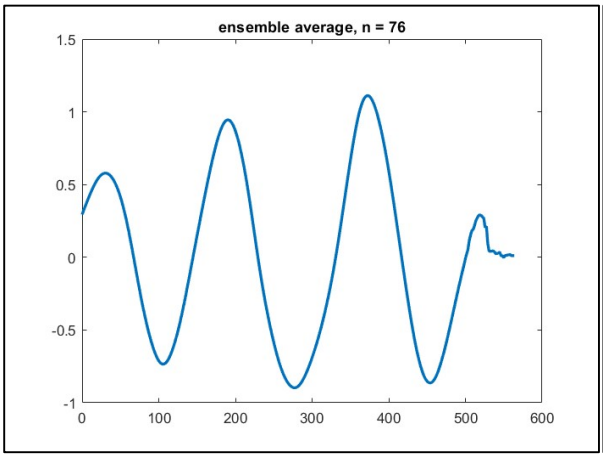


Figure 18 - Mean filtered pressure signal

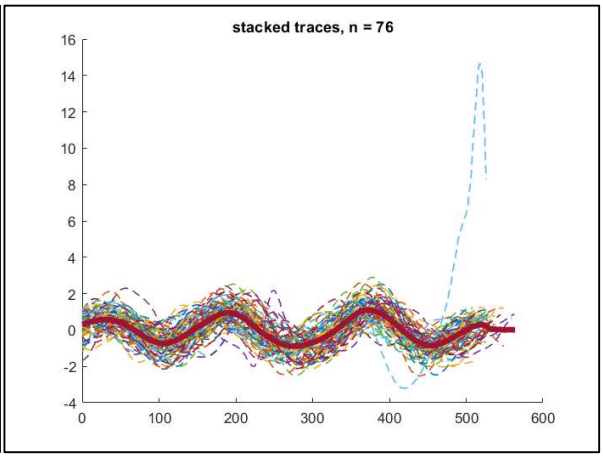


Figure 17 - Mean filtered pressure signal overlayed all pressure signal segments

The signals showed in Figure 18 and Figure 17 show a clear periodic signal which matches the timing of the pulse. This is a good sign of the setup capturing something related to the cardiovascular system which most likely is the ballistocardiogram. However, due to the variations in the waveform of the ballistocardiogram with different setups, equipment and variations from person to person, as discussed in section 3.1, it is somewhat difficult to compare the results to existing methods of ballistocardiography. The signal being segmented using a pulse sensor, with many other setups being tested using EKG also complicates the comparison somewhat. The J peak is not as prominent as desired, but the W shape is easily recognizable. Another promising aspect is the large correlation between the segments of the pressure signal. Based on the W shape, the measurement method and the segments repeating pattern it is concluded that this is the ballistocardiogram being captured.

6 Further work

As the current solution is still only a proof of concept there still a lot of improvements needed. Although the signal after filtering does show very promising results more improvements the quality could still be valuable as increased quality can possibly make it possible to measure in low-intensity activities such as walking. One obvious improvement that was not tested due to a lack of time is using a liquid instead of air. Air is compressable while a liquid like water is in the practical sense for this setup not compressable, which should decrease any dampening effect due to the air compressing and improve the signal. Motion fragments in the captured signal is another aspect that needs investigation. As discussed in section 3.1 motion fragments in the signal are a common problem in BCG. Filtering is one solution for handling motion fragments, but more solutions will most likely need to be investigated for it to be possible to measure the ballistocardiogram in the aforementioned low intensity activities.

In addition to the discussed improvements further testing is also needed before the method can be a potential tool monitoring the cardiovascular system. Testing with ECG for better segmentation and comparison with other methods is needed. The results also need to be possible to segment without other reference signals to have value on its own. Testing with more participants is also needed to test for possible unknown individual differences. The ballistocardiogram is, as discussed in section 3.1, somewhat different from person to person and this needs to be investigated for this setup as well to ensure it transferable to other users. In addition, differences in weight, form of the foot and general movement can possibly affect the measurements.

The current setup is also not wearable. Therefore, solutions for implementation in wearable equipment is needed. The most obvious solution is to implement the setup in a smart shoe or sole. The current setup was developed with this in mind, and such an integration should be possible. However, exploring this design space of integrating the setup into other wearable equipment might be interesting as other integration solutions most likely exist.

7 Summary

This project thesis has explored the solution space related to sensor setups for the next generation of wearables monitoring the cardiovascular system. Wayfaring, prototyping, and benchmarking were the main tools used to explore the solution space. Through these three methods ballistocardiography was identified as an interesting topic with pressure sensors as an interesting solution. Many iterations of prototypes were made, and a proof-of-concept prototype was developed. This consisted of two air filled bags with two barometric pressure sensors attached. The bags were placed under the front of the foot and the heel of the user. In addition, a stiff MDF plate was placed between the bags and the foot. Some key requirements were discovered during the wayfaring process. The surface area of the bags had to be very small. At the same time the volume to surface area ratio had to be kept from becoming too small as this led to the MDF plate potentially touching the ground and destroying the signal. The proof-of-concept prototype captured a repeating signal matching up with the signal captured by a pulse sensor. The segmented signal had a clear pattern. The mean signal calculated from the segments had the recognizable W-shape of a ballistocardiogram, but the characteristic J-peak was not as prominent as it usually is. It was still concluded with this being the ballistocardiogram being captured.

Many improvements remain before this can be a potential valuable tool. The signal needs to be improved as much as possible to gain the best possible results. The setup also needs to be segmented using ECG instead of PPG for better comparison between existing BCG methods. In addition, being able to segment the signal without using external input would greatly increase its stand-alone value. In addition, the setup is also only tested on one user. As such there might exist unknown individual differences which need to be tested for. Finally, this is only a proof-of-concept prototype and not currently wearable. Integration into for instance a smart shoe needs to be explored to enable continuous monitoring of the cardiovascular system.

References

Auflem, M., Erichsen, J.F., Steinert, M., 2019. Exemplifying Prototype-Driven Development through Concepts for Medical Training Simulators. Procedia CIRP, 29th CIRP Design Conference 2019, 08-10 May 2019, Póvoa de Varzim, Portgal 84, 572–578. https://doi.org/10.1016/j.procir.2019.04.202

BMP280 [WWW Document], n.d. . Bosch Sensortec. URL https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/ (accessed 12.19.21).

BMP388 [WWW Document], n.d. . Bosch Sensortec. URL https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp388/ (accessed 12.19.21).

CDC, 2021. Coronary Artery Disease | cdc.gov [WWW Document]. Cent. Dis. Control Prev. URL https://www.cdc.gov/heartdisease/coronary_ad.htm (accessed 12.20.21).

Dunn, J., Runge, R., Snyder, M., 2018. Wearables and the medical revolution. Pers. Med. 15, 429–448. https://doi.org/10.2217/pme-2018-0044

Elverum, C.W., Welo, T., 2015. On the use of directional and incremental prototyping in the development of high novelty products: Two case studies in the automotive industry. J. Eng. Technol. Manag. 38, 71–88. https://doi.org/10.1016/j.jengtecman.2015.09.003

Gerstenberg, A., Sjöman, H., Reime, T., Abrahamsson, P., Steinert, M., 2015. A Simultaneous, Multidisciplinary Development and Design Journey – Reflections on Prototyping, in: Chorianopoulos, K., Divitini, M., Baalsrud Hauge, J., Jaccheri, L., Malaka, R. (Eds.), Entertainment Computing - ICEC 2015, Lecture Notes in Computer Science. Springer International Publishing, Cham, pp. 409–416. https://doi.org/10.1007/978-3-319-24589-8_33

Giovangrandi, L., Inan, O.T., Wiard, R.M., Etemadi, M., Kovacs, G.T.A., 2011. Ballistocardiography — A method worth revisiting, in: 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Presented at the 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 4279–4282. https://doi.org/10.1109/IEMBS.2011.6091062

Houde, S., Hill, C., 1997. Chapter 16 - What do Prototypes Prototype?, in: Helander, M.G., Landauer, T.K., Prabhu, P.V. (Eds.), Handbook of Human-Computer Interaction (Second Edition). North-Holland, Amsterdam, pp. 367–381. https://doi.org/10.1016/B978-044481862-1.50082-0

Inan, O.T., Etemadi, M., Wiard, R.M., Giovangrandi, L., Kovacs, G.T.A., 2009. Robust ballistocardiogram acquisition for home monitoring. Physiol. Meas. 30, 169–185. https://doi.org/10.1088/0967-3334/30/2/005

Jensen, L.S., Özkil, A.G., Mortensen, N.H., 2016. PROTOTYPES IN ENGINEERING DESIGN: DEFINITIONS AND STRATEGIES. 84 Proc. Des. 2016 14th Int. Des. Conf. 821–830.

Kennedy, B.M., Sobek II, D.K., Kennedy, M.N., 2014. Reducing Rework by Applying Set-Based Practices Early in the Systems Engineering Process. Syst. Eng. 17, 278–296. https://doi.org/10.1002/sys.21269

Kim, J., Wilemon, D., 2002. Focusing the fuzzy front–end in new product development. RD Manag. 32, 269–279. https://doi.org/10.1111/1467-9310.00259

Koivistoinen, T., Junnila, S., Varri, A., Koobi, T., 2004. A new method for measuring the ballistocardiogram using EMFi sensors in a normal chair, in: The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Presented at the The 26th Annual International Conference of the IEEE

Engineering in Medicine and Biology Society, pp. 2026–2029. https://doi.org/10.1109/IEMBS.2004.1403596
Kriesi, C., Blindheim, J., Bjelland, Ø., Steinert, M., 2016. Creating Dynamic Requirements through Iteratively Prototyping Critical Functionalities. Procedia CIRP, 26th CIRP Design Conference 50, 790–795. https://doi.org/10.1016/j.procir.2016.04.122
Lauff, C.A., Kotys-Schwartz, D., Rentschler, M.E., 2018. What is a Prototype? What are the Roles of Prototypes in Companies? J. Mech. Des. 140. https://doi.org/10.1115/1.4039340
Mora, N., Cocconcelli, F., Matrella, G., Ciampolini, P., 2020. Accurate Heartbeat Detection on Ballistocardiogram Accelerometric Traces. IEEE Trans. Instrum. Meas. 69, 9000–9009. https://doi.org/10.1109/TIM.2020.2998644
Pinheiro, E., Postolache, O., Girão, P., 2010. Theory and Developments in an Unobtrusive Cardiovascular System Representation: Ballistocardiography. Open Biomed. Eng. J. 4, 201–216. https://doi.org/10.2174/1874120701004010201
Pinheiro, E., Postolache, O., Girao, P., 2009. Pulse arrival time and ballistocardiogram application to blood pressure variability estimation, in: 2009 IEEE International Workshop on Medical Measurements and Applications. Presented at the 2009 IEEE International Workshop on Medical Measurements and Applications, pp. 132–136. https://doi.org/10.1109/MEMEA.2009.5167970
Schrage, M., 1996. Cultures of prototyping, in: Winograd, T. (Ed.), Bringing Design to Software. ACM, New York, NY, USA, pp. 191–213. https://doi.org/10.1145/229868.230045
Steinert, M., Leifer, L., 2012. "Finding One's Way": Re-Discovering a Hunter-Gatherer Model based on Wayfaring. Int. J. Eng. Educ. 28, 251–252.
Sutcliffe, A., Sawyer, P., 2013. Requirements elicitation: Towards the unknown unknowns, in: 2013 21st IEEE International Requirements Engineering Conference (RE). Presented at the 2013 21st IEEE International Requirements Engineering Conference (RE), pp. 92–104. https://doi.org/10.1109/RE.2013.6636709

Appendix

Appendix A: Arduino code for data capture in final setup

```
#include "DFRobot_BMP388_I2C.h"

#include "DFRobot_BMP388.h"

#include "Wire.h"

#include "bmp3_defs.h"


//Create a BMP object for each sensor in use

DFRobot_BMP388_I2C bmp1;

DFRobot_BMP388_I2C bmp2;


int Signal;                // holds the incoming data from the pulsesensor

int PulseSensorPurplePin = A0;        //the purple sensor of the
pulsesensor is wired to A0


void setup(){

  //Start up serial communication

  Serial.begin(115200);

  //Set adresses for the two sensors. 0x77 and 0x76 is the two possible I2C
adresses

  bmp1.set_iic_addr(BMP3_I2C_ADDR_SEC);

  bmp2.set_iic_addr(BMP3_I2C_ADDR_PRIM);

  while(bmp1.begin()){ //Initialize sensor 1

    Serial.println("Initialize error, sensor 1!");

    delay(1000);

  }

  while(bmp2.begin()){ //Initialize sensor 2

    Serial.println("Initialize error, sensor 2!");

    delay(1000);

  }

}


void loop(){
```

```
  //Read the pressure from sensor 1 and 2
  float Pressure1 = bmp1.readPressure();
  float Pressure2 = bmp2.readPressure();
  //Print the time, the pulssensor value and the two pressure values in a
CSV format
  Serial.print(millis());
  Serial.print(",");
  Serial.print(analogRead(PulseSensorPurplePin));
  Serial.print(",");
  Serial.print(Pressure1);
  Serial.print(",");
  Serial.println(Pressure2);
  //delay(1);
}
```

Appendix B: Matlab code for result analysis

Appendix B contains matlab code for analyzing the results from the proof of concept setup. The Matlab was developed by my co-supervisor Torjus Lines Steffensen. It contains two files. File 1 does the filtering, segmentation and computing of the mean signal. File 2 is only used to allow concatenation between vectors with different length.

File 1:

```
bcg plot
load a file containing bcg and ppg signal into a matlab timetable
plot the raw signals along with the power pectrum density estimate
use ppg maxima to annotate individual beats
filter the data
use ppg maxima locations to separate cycles
calculate ensemble average signal.
improvements / todos:
use a real algorithm to segment ppg signal (we are very naive and not
robust currently)
set some criteria for good/bad segment


% load a csv file
% expected structure:
% Time,PPG,Pressure
clear all
data = readtable('C:\Users\simon\Documents\Prosjektoppgave -
lokal\Målinger\Pressure_And_PPG_P7_18_11_2021.csv');
```

```
% store the data in a matlab "timetable" structure.
% this is optional, but timetables have many high level features that make
% dealing with time series simpler (for example, selecting all data between
% two specific time points without knowing what the sample indexes would
% be)
% note this means the default timetable time index, "timetable.Time", is
% not double type but one of matlab's proprietary "duration" or "datetime"
% datatypes. can switch back and forth between duration and double with the
functions
% seconds() or milliseconds() for some other functions that don't accept
% one of these types.

data.Time = milliseconds(data.Time - data.Time(1));
tt = timetable(data.Time,data.PPG,data.Pressure);
tt.Properties.VariableNames = {'ppg','Pressure'};

% calculate the mean sample rate, we'll need it later
fs = 1 / seconds(mean(diff(tt.Time)));

% plot the signal we just loaded alongside a ~10 second slice to see what
we're
% dealing with

figure
subplot(4,1,1)
plot(tt.Time,detrend(tt.Pressure))
title('raw pressure')
subplot(4,1,2)
plot(tt.Time,detrend(tt.ppg))
title('ppg')
subplot(4,1,3)
plot(tt.Time,tt.Pressure)
xlim([seconds(15) seconds(25)])
title('pressure section, close up')
subplot(4,1,4)
plot(tt.Time,detrend(tt.ppg))
```

```
xlim([seconds(15) seconds(25)])
title('ppg section, close up')
```

power spectrum

```
% plot the power spectrum of the signal in the range of interest
% (up to 20hz)

figure
pspectrum(tt.Pressure,fs)
title('power spectrum')
xlim([0 10])

% OPTIONAL: trim the raw data
% (for example if we see the signal is only "good" between 0, 30 seconds)

tr = timerange(seconds(5),seconds(45));
tt = tt(tr,:);
```

finding ppg peaks.

```
% we start by denoising the ppg signal via wavelet reconstruction (it has
% to be smooth for differentiation).

% Logical array for selecting reconstruction elements
levelForReconstruction = [false, false, false, true, true, true, true,
true, false];
% Perform the decomposition using modwt
wt = modwt(detrend(tt.ppg), 'sym4', 8);
% Construct MRA matrix using modwtmra
mra = modwtmra(wt, 'sym4');
% Sum along selected multiresolution signals
tt.bppg = sum(mra(levelForReconstruction,:),1).';

% calculate the first and second derivatives (we might use them later on if
% ppg signal is not "clean")
dppg = diff(tt.bppg);
```

```
dppg = [dppg;dppg(end)];
ddppg = diff(dppg);
ddppg = [ddppg;ddppg(end)];

% plot the smoothed ppg signal and its derivatives
figure
subplot(3,1,1)
plot(tt.Time,tt.bppg)
title('ppg and derivatives')
ylabel('ppg')
xlim([seconds(15) seconds(20)])
subplot(3,1,2)
plot(tt.Time,dppg)
ylabel('d/dt ppg')
xlim([seconds(15) seconds(20)])
subplot(3,1,3)
plot(tt.Time,ddppg)
ylabel('(d/dt)^2 ppg')
xlim([seconds(15) seconds(20)])



% in case of a pretty clean ppg, we can just use naive peak finding.
% the function findpeaks below finds local maxima, with some requirements
% like minimum prominence and minimum distance between peak candidates.
% pplocs is the index of the identified peaks


% find peaks, plot the filtered ppg signal alongside the identified peak
% locations (see if there are many missing beats or false positives, in
which case
% we either have to tweak our function call, or move on to use the
derivatives
% which takes a little more thought)


[pppks,pplocs]=findpeaks(tt.bppg,'MinPeakProminence',10,'MinPeakDistance',2
20);
figure
plot(tt.Time,tt.bppg)
```

```
xline(tt.Time(pplocs),'--r')
xlim([seconds(15) seconds(20)])
title('identified ppg peaks')

% we can now get the heart rate by looking at time between peaks in the
% part of the signal we're interested in

dim = [.2 .6 .3 .3];
str = ["HR estimate from PPG peak distance: " +
seconds(60)/mean(diff(tt.Time(pplocs)))];
t=annotation('textbox',dim,'String',str,'FitBoxToText','on');
t.BackgroundColor=[1 1 1];

filter the pressure data
% we will do two things: a classic FIR bandpass filter, and a wavelet
% decomposition. we can compare the results later.

% start by creating the digital bandpass filter. this fill attenuate signal
% components below 1hz, and above 20hz.

d=designfilt('bandpassfir','StopbandFrequency1',0.7,'PassbandFrequency1',1.
5,...
    'PassbandFrequency2',15,'StopbandFrequency2',22,...
    'StopbandAttenuation1',60,'PassbandRipple',1, ...
    'StopbandAttenuation2',60,'SampleRate',fs, ...
    'DesignMethod','kaiserwin');



% we apply the filter twice: "forwards" and "backwards". this is because
% this type of filter can affect the phase of our signal. using filtfilt()
% we avoid this (this is called zero phase filtering)

tt.bp = filtfilt(d,tt.Pressure);

% next we also do a wavelet decomposition using discrete wavelet transform
% (modwt)
```

```
levelForReconstruction = [false, false, false, true, true, true, false,
false, false];
% Perform the decomposition using modwt
wt = modwt(tt.Pressure, 'db9', 8);
% Construct MRA matrix using modwtmra
mra = modwtmra(wt, 'db9');
% Sum along selected multiresolution signals
tt.wmra = sum(mra(levelForReconstruction,:),1).';



% if we want, we can plot the original signal alongside the filtered signal
% to see the result of what we just did

f1 = figure;
plot(tt.Time,detrend(tt.Pressure),'--')
hold on
plot(tt.Time,tt.bp)
plot(tt.Time,tt.wmra)
xlim([seconds(15) seconds(25)])
legend([{'original'},{'bandpassed'},{'wmra'}])
title('filtered signal')

segment pressure data

% this code is adapted from another script, so it might look weird.

% which index do we use to segment?
% in this case we only have ppg, so we use pplocs
segmentIndex = pplocs;
segments=[];
% segmentBuffer can be used to "move" the segments uniformly backwards or
% forwards in time by the same number of samples. can be useful for
% plotting
segmentBuffer = -50;

% which signal to segment?
% bp: FIR bandpassed signal, wmra: wavelet filtered
```

```
use = 'wmra';


% using the reference points in segmentIndex, collect all data between
% index i and index i+1 into its own structure in the struct segments.
% scale all the segments using z-scoring.


try
for i = 1 : length(segmentIndex) - 1


    tr=timerange(tt.Time(segmentIndex(i) -
segmentBuffer),tt.Time(segmentIndex(i+1) - segmentBuffer));


    % watch out: some sanity checks for too long / too short segments have
been
    % added, but dangerously, are in #samples rather than actual time
    % (danger if sample rate changes)
    %Which it has done -


    %if length(tt.Time(segmentIndex(i)-segmentBuffer : segmentIndex(i+1)-
segmentBuffer)) > 80 && ...
    %        length(tt.Time(segmentIndex(i)-segmentBuffer :
segmentIndex(i+1)-segmentBuffer)) < 150


        segments(i).Time = tt.Time(segmentIndex(i)-segmentBuffer :
segmentIndex(i+1)-segmentBuffer);
        segments(i).Timefz = segments(i).Time - segments(i).Time(1);


        tmptable = tt(tr,:).ppg;
        segments(i).ppg = (tmptable - mean(tt.ppg)) / std (tt.ppg);
        segments(i).ppg = [segments(i).ppg;segments(i).ppg(end)];


        tmptable = tt(tr,use).(1);
        segments(i).pressure = (tmptable - mean(tt(:,use).(1))) / std
(tt(:,use).(1));
        segments(i).pressure =
[segments(i).pressure;segments(i).pressure(end)];
    %end
end
end
```

```
% which segments to use to plot / average?
startseg = 1;
endseg = numel(segments);
sigs = [startseg endseg];

% plot selected segments
f2 = figure;
figure(f2)
hold on
for i = startseg : endseg
    plot(milliseconds(segments(i).Timefz),segments(i).pressure,'--
','LineWidth',1);
end
title(["stacked traces, n = " + sum([diff(sigs),1])])

% calculate mean signals
% this requires the padcat external function. padcat concatenates matrices
% with incompatible sizes by padding with NaN

tr = timerange(milliseconds(0),milliseconds(1000));
means = [];
means.pressure = padcat(segments(sigs(1):sigs(2)).pressure);
means.pressure(isnan(means.pressure)) = 0;
means.pressure = mean(means.pressure,2);
means.pressure = timetable(means.pressure,'SampleRate',fs);
means.pressure = means.pressure(tr,:);
means.pressure.Properties.VariableNames={'data'};

means.ppg = padcat(segments(sigs(1):sigs(2)).ppg);
means.ppg(isnan(means.ppg)) = 0;
means.ppg = mean(means.ppg,2);
means.ppg = timetable(means.ppg,'SampleRate',fs);
means.ppg = means.ppg(tr,:);
means.ppg.Properties.VariableNames={'data'};

figure(f2)
```

```
plot(milliseconds(means.pressure.Time),means.pressure.data,'-
','LineWidth',4)

figure

plot(milliseconds(means.pressure.Time),means.pressure.data,'-
','LineWidth',2)

title(["ensemble average, n = " + sum([diff(sigs),1])])




figure

subplot(2,1,1)

plot([means.pressure.Time
means.pressure.Time+means.pressure.Time(end)],[means.ppg.data
means.ppg.data],'-b','LineWidth',2)

ylim([1.5*min(means.ppg.data) 1.5*max(means.ppg.data)])

subplot(2,1,2)

plot([means.pressure.Time
means.pressure.Time+means.pressure.Time(end)],[means.pressure.data
means.pressure.data],'-b','LineWidth',2)

ylim([1.5*min(means.pressure.data) 1.5*max(means.pressure.data)])
```

File 2:

```
function [M, TF] = padcat(varargin)

% PADCAT - concatenate vectors with different lengths by padding with NaN

%

%   M = PADCAT(V1, V2, V3, ..., VN) concatenates the vectors V1 through VN

%   into one large matrix. All vectors should have the same orientation,

%   that is, they are all row or column vectors. The vectors do not need to

%   have the same lengths, and shorter vectors are padded with NaNs.

%   The size of M is determined by the length of the longest vector. For

%   row vectors, M will be a N-by-MaxL matrix and for column vectors, M

%   will be a MaxL-by-N matrix, where MaxL is the length of the longest

%   vector.

%

%   Examples:

%      a = 1:5 ; b = 1:3 ; c = [] ; d = 1:4 ;

%      padcat(a,b,c,d) % row vectors

%         % ->   1     2     3     4     5

%         %      1     2     3   NaN   NaN

%         %    NaN   NaN   NaN   NaN   NaN
```

```
%         %      1     2     3     4   NaN
%      CC = {d.' a.' c.' b.' d.'} ;
%      padcat(CC{:}) % column vectors
%         %      1     1   NaN     1     1
%         %      2     2   NaN     2     2
%         %      3     3   NaN     3     3
%         %      4     4   NaN   NaN     4
%         %    NaN     5   NaN   NaN   NaN
%
%   [M, TF] = PADCAT(..) will also return a logical matrix TF with the same
%   size as R having true values for those positions that originate from an
%   input vector. This may be useful if any of the vectors contain NaNs.
%
%   Example:
%       a = 1:3 ; b = [] ; c = [1 NaN] ;
%       [M,tf] = padcat(a,b,c)
%       % find the original NaN
%       [Vev,Pos] = find(tf & isnan(M))
%       % -> Vec = 3 , Pos = 2
%
%   This second output can also be used to change the padding value into
%   something else than NaN.
%
%       [M, tf] = padcat(1:3,1,1:4)
%       M(~tf) = 99 % change the padding value into 99
%
%   Scalars will be concatenated into a single column vector.
%
%   See also CAT, RESHAPE, STRVCAT, CHAR, HORZCAT, VERTCAT, ISEMPTY
%            NONES, GROUP2CELL (Matlab File Exchange)

% Example figure created using:
%   C = arrayfun(@(x) ones(1,randi([10 100],1,1)),1:40,'un',0) ;
%   pcolor(padcat(C{:}))

% for Matlab 2008 and up (last tested in R2018a)
```

```
% version 1.4 (dec 2018)
% (c) Jos van der Geest
% email: samelinoa@gmail.com

% History
% 1.0 (feb 2009) created
% 1.1 (feb 2011) improved comments
% 1.2 (oct 2011) added help on changing the padding value into something
%     else than NaN
% 1.3 (feb 2016) updated contact info
% 1.4 (dec 2018) fixed minor code warnings

% Acknowledgements:
% Inspired by padadd.m (feb 2000) Fex ID 209 by Dave Johnson

narginchk(1,Inf) ;

% check the inputs
SZ = cellfun(@size,varargin,'UniformOutput',false) ; % sizes
Ndim = cellfun(@ndims,varargin) ; %

if ~all(Ndim==2)
    error([mfilename ':WrongInputDimension'], ...
        'Input should be vectors.') ;
end

TF = [] ; % default second output so we do not have to check all the time

% for 2D matrices (including vectors) the size is a 1-by-2 vector
SZ = cat(1,SZ{:}) ;
maxSZ = max(SZ) ;    % probable size of the longest vector
% maxSZ equals :
%  - [1 1] for all scalars input
%  - [X 1] for column vectors
%  - [1 X] for all row vectors
%  - [X Y] otherwise (so padcat will not work!)
```

```
if ~any(maxSZ == 1)  % hmm, not all elements are 1-by-N or N-by-1
    % 2 options ...
    if any(maxSZ==0)
        % 1) all inputs are empty
        M  = [] ;
        return
    else
        % 2) wrong input
        % Either not all vectors have the same orientation (row and column
        % vectors are being mixed) or an input is a matrix.
        error([mfilename ':WrongInputSize'], ...
            'Inputs should be all row vectors or all column vectors.') ;
    end
end

if nargin == 1
    % single input, nothing to concatenate ..
    M = varargin{1} ;
else
    % Concatenate row vectors in a row, and column vectors in a column.
    dim = (maxSZ(1)==1) + 1 ;      % Find out the dimension to work on
    X = cat(dim, varargin{:}) ;    % make one big list

    % we will use linear indexing, which operates along columns. We apply a
    % transpose at the end if the input were row vectors.

    if maxSZ(dim) == 1
        % if all inputs are scalars, ...
        M = X ;   % copy the list
    elseif all(SZ(:,dim)==SZ(1,dim))
        % all vectors have the same length
        M = reshape(X,SZ(1,dim),[]) ;% copy the list and reshape
    else
        % We do have vectors of different lengths.
        % Pre-allocate the final output array as a column oriented array.
We
```

```
        % make it one larger to accommodate the largest vector as well.
        M = zeros([maxSZ(dim)+1 nargin]) ;
        % where do the fillers begin in each column
        M(sub2ind(size(M), SZ(:,dim).'+1, 1:nargin)) = 1 ;
        % Fillers should be put in after that position as well, so applying
        % cumsum on the columns
        % Note that we remove the last row; the largest vector will fill an
        % entire column.
        M = cumsum(M(1:end-1,:),1) ; % remove last row

        % If we need to return position of the non-fillers we will get them
        % now. We cannot do it afterwards, since NaNs may be present in the
        % inputs.
        if nargout > 1
            TF = ~M ;
            % and make use of this logical array
            M(~TF) = NaN ; % put the fillers in
            M(TF)  = X ;   % put the values in
        else
            M(M==1) = NaN ; % put the fillers in
            M(M==0) = X ;   % put the values in
        end
    end

    if dim == 2
        % the inputs were row vectors, so transpose
        M = M.' ;
        TF = TF.' ; % was initialized as empty if not requested
    end
end % nargin == 1

if nargout > 1 && isempty(TF)
    % in this case, the inputs were all empty, all scalars, or all had the
    % same size.
    TF = true(size(M)) ;
end
```

Appendix C

Risk assessment form

RISIKOANALYSE

Enhet/institutt: Institutt for maskinteknikk og produksjon — **Dato opprettet:** 06.09.2021

Ansvarlig ingeniør (navn): Martin Steinert — **Sist revidert:** 14.09.2021

Ansvarlig for aktiviteten som risikovurderes (navn): Simon Gåseby Gjerde

Deltakere (navn): Simon Gåseby Gjerde

Beskrivelse av den aktuelle aktiviteten, området mv.:

Risikoanalysen omfatter testing og laging av kretser med sensorer samt tilvirkning av diverse festemekanismer for å realisere prosjekt og masteroppgave. Dette vil i hovedsak skje på Trollabs, men Mekatronikklaben på Verkstedteknisk vil også bli brukt. I tillegg vil også Realiseringslabben på Verkstedteknisk kunne bli brukt noe. Arbeidet på Mekatronikklaben vil i hovedsak bestå av lodding og kretsing av enkle lavspenningskretser, mens arbeider på Realiseringslabben vil kunne bestå av blant annet fresing, dreiing, sveising og båndsag for å lage enkle festemekanismer og lignende. Enkle håndverktøy vil også bli benyttet for montering og tilpasning. All bruk av maskiner er med forbehold om at maskinkort og risikovurdering for maskiner eksisterer samt at nødvendig opplæring på den spesifikke maskinen er gjennomført.

Aktivitet/arbeidsoppgave	Mulig uønsket hendelse	Eksisterende risikoreduserende tiltak	Vurdering av sannsynlighet (S)	Vurdering av konsekvens (K)				Risikoverdi (S x K)	Forslag til forebyggende og/eller korrigerende tiltak	Restrisiko etter tiltak (S x K)
				Vurder en konsevenskategori om gangen. Menneske skal alltid vurderes.						
			(1-5)	Menneske (1-5)	Øk/materiell (1-5)	Ytre miljø (1-5)	Omdømme (1-5)		*Prioriter tiltak som kan forhindre at hendelsen inntreffer (sannsynlighetsreduserende tiltak) foran skjerpet beredskap (konsekvensreduserende tiltak)*	
Alle	Smitte	NTNUs, Fakultetets og instituttets retningslinjer mht til smittevern							Kun bruk av egne vernebriller, vernesko og bekledning.	
Arbeid med kretser	Støt fra strømførende kretser		3	3				9	det arbeides med kretsen	3
3D-printer	Brannskader fra varme komponenter		3	1				3	Gi printeren tid til å kjøle ned etter ferdig print.	2
Montering av elementer med fjærkomponenter	Klemming av fingre eller andre kroppsdeler og komponenter som blir "kastet" i øynene		3	2				6	Ta tiltak med avstand basert på styrke til fjær. Bruke tang eller annet spesialisert utstyr. Bruke vernebriller for å skjerme øyne.	3
Lodding	Brannskader	Tidsur på loddebolt	3	1				3	Skru av loddebolt når den ikke er i bruk og sette den tilbake på egnet plass. Fjerne brannfarlig materiell når loddebolt er i bruk	2
Bruk av laserkutter	Brannskader	Slukkeutstyr tilgjengelig	2	3				6	Minimere mengde støv på plater. Alltid holde laserkutteren	4
Bruk av roterende håndverktøy	Hektling i roterende utstyr	Godt grep, oppsamming	2	2				4	Ikke ha på løse klær og stroppet. Sette opp langt hår.	2
Bruk av roterende håndverktøy	Metallspon som sprutter		2	2				4	Bruk av vernebriller	2
Bruk av manuelle skjæreverktøy	Kuttskader		3	1				3		3
Boring	Hektling i roterende utstyr	Bruke påkrevd (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	2	4				8	Ikke ha på løse klær og stroppet. Sette opp langt hår. Unngå eller brette opp lange ermer.	4
Bøying av metalldeler	Klemming av fingre eller andre kroppsdeler		3	1				3	Bruke egnet utstyr samt verneutstyr.	3
Sveising	Brannskader	Verneutstyr, maskinkort	3	2				6	Påse at deler er kjølt ned før man tar på de. Bruke ir-termometer.	3
Dreiing	Sprutrust	Verneutstyr, maskinkort	3	1				3		3
	Hektling i roterende utstyr	Bruke påkrevd (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	2	4				8	Ikke ha på løse klær og stroppet. Sette opp langt hår. Unngå eller brette opp lange ermer.	4
Dreiing	Sponsprut	Verneutstyr, maskinkort	3	1				3		3
Fresing	Sponsprut	Verneutstyr, maskinkort	3	1				3		3
	Hektling, deler som knekker og sprutter mot ansikt, metallspon som sprutter	Bruke påkrevd (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	2	4				8	Ikke ha på løse klær og stroppet. Sette opp langt hår. Unngå eller brette opp lange ermer.	4
Fresing	Avkapping av kroppdeler	Platelholder som holder platen	1	5				5	Holde begge hender på håndtaket når man kutter	5
Platekutter	Splintsprut	Vernebriller	2	3				6	Unngå å kutte sprø eller skadede materialstykker	3

Båndsliper	Hekting av klær eller annet materiell	Bruke påkrevet (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	3	3				9	Unngå løse klær og hansker. Holde objekt med tang om mulig.	4
Båndsliper	Sprut i øye	Bruke vernebriller	3	1				3		3
Bruk av båndsag	Kuttskader	Bruke påkrevet (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	3	2				6	Opplæring ved bruk, samt forsiktighet. Ikke kappe små deler.	4
Bruk av lim og sammenføyningsmiddler	Eksponering på øye	Lese sikkerhetsdatablad før bruk	3	2				6	Bruk vernebriller. Ha datablad tilgjengelig. Ha øyeskylleflaske tilgjengelig.	4
Bruk av lim og sammenføyningsmiddler	Eksponering på hud	Lese sikkerhetsdatablad før bruk	2	2				4	Bruke nitril-hansker	4
Bruk av lim og sammenføyningsmiddler	Eksponering åndedrett	Lese sikkerhetsdatablad før bruk	4	2				8	Bruk av avtrekk og/eller maske.	4
Bruk av løsemiddler for vask og desinfisering	Unødig eksponering hud og åndedrett, ekponering på hud	Lese sikkerhetsdatablad før bruk	4	2				8	Unngå unødig store mengder, bruke vernebriller og hansker om nødvendig. Bruk av avtrekk om nødvendig.	4
Vakumtrekking	Klem-skader	Bruke påkrevd (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	2	2				4	1 operatør, 2 hender på panel	2
Bruk av "CNC-trefres"	Eksponering av spon på øye	Bruke påkrevd (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	3	1				3	Bruk av vernebriller	2
Bruk av hydraulisk presse	Klemmskader	Bruke påkrevd (personlig) verneutstyr som beskrevet under opplæring og på maskinkort	3	3				9	Kun 1 operatør. Alltid holde to hender på spaken.	3
Annet eksperimentell arbeid		Egen SJA der nødvendig								

Risikovurderingen er lest og godkjent av: Veileder: _____ Dato: _____

Appendix D

Participation forms

Vil du delta i forskningsprosjektet

«Såle 2.0 – hvor foten trykker»?

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å teste en smart-sko. Skoen et utstyrt med sensorer som kan måle flere variabler mens du bruker skoen. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål

Vi ønsker å teste hvordan smartskoen vår fungerer på et lite antall deltagere. Vi vil vite om det er mulig å måle hjerterytme (puls), pulstransitt-tid og aktivitetsmønster gjennom skoen.

Dette prosjektet er del av en masteroppgave ved Institutt for Maskinteknikk og Produksjon ved NTNU.

Opplysninger vi samler inn fra deg kan brukes, i anonymisert form, i en vitenskapelig publikasjon. Du vil ikke kunne identifiseres.

Hvem er ansvarlig for forskningsprosjektet?

Institutt for Maskinteknikk og Produksjon (MTP), Fakultet for Ingeniørvitenskap ved NTNU er ansvarlig for prosjektet. Prosjektleder er professor Martin Steinert.

Hvorfor får du spørsmål om å delta?

Du får forespørsel om å delta fordi du har svart på etterspørselen vår ved MTP om å delta.

Hva innebærer det for deg å delta?

Hvis du velger å delta i prosjektet, vil vi be deg om å oppgi følgende personopplysninger gjennom intervju:

- Vekt
- Høyde
- Alder
- Kjønn
- Skostørrelse.

Deretter måler vi trykk og akselerometri gjennom smartskoen. Denne informasjonen kan brukes til å regne ut hjerterytmen din (puls) mens du har på skoen. VI vil også måle blodtrykket ditt før og under testen. Grunnen til at vi ber om å lagre personopplysningene om deg nevnt over, er at denne informasjonen kan fortelle oss hvordan for eksempel kroppsmasseindeks (BMI) påvirker metoden for å regne ut hjerterytme. Skostørrelsen kan påvirke målingene. Blodtrykk kan påvirke noen av målingene.

All informasjon vil lagres elektronisk. Informasjonen vil lagres anonymt etter datainnsamlingen.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

- Følgende personer vil ha tilgang til informasjonen om deg:
  - Simon Gåseby Gjerde
    - Tlf. 47 83 25 41, epost: simongg@stud.ntnu.no
  - Torjus Lines Steffensen
    - Tlf. 91 19 34 66, epost: torjus.l.steffensen@ntnu.no

- Ditt navn vil erstattes med en kode. En liste over koblinger mellom navn og kode vil lagres på et kryptert medium, adskilt fra øvrig data.

Vi vil kunne gjengi anonymisert informasjon i en eventuell publikasjon.
Du vil ikke kunne identifiseres.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?
Koblingsnøkkel slettes når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er 31. 12. 2022. Dette betyr at da vil all informasjonen være permanent anonymisert.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra NTNU har Personverntjenester vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:
- innsyn i hvilke opplysninger vi behandler om deg, og å få utlevert en kopi av opplysningene
- å få rettet opplysninger om deg som er feil eller misvisende
- å få slettet personopplysninger om deg
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger

Hvis du har spørsmål til studien, eller ønsker å vite mer om eller benytte deg av dine rettigheter, ta kontakt med:
- Hovedutprøver, Simon Gåseby Gjerde
  - Tlf. 47 83 25 41, simongg@stud.ntnu.no
- NTNU ved prosjektansvarlig, professor Martin Steinert (martin.steinert@ntnu.no).
- Prosjektmedarbeider Torjus Lines Steffensen
  - Tlf. 91 19 34 66, epost: torjus.l.steffensen@ntnu.no
- Vårt personvernombud: Thomas Helgesen, thomas.helgesen@ntnu.no

Hvis du har spørsmål knyttet til Personverntjenester sin vurdering av prosjektet, kan du ta kontakt med:
- Personverntjenester på epost (personverntjenester@sikt.no) eller på telefon: 53 21 15 00.

Med vennlig hilsen

Martin Steinert (Forsker/veileder)

Simon Gåseby Gjerde

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet *Såle 2.0 – hvor foten trykker*, og har fått anledning til å stille spørsmål. Jeg samtykker til:

- ☐ å delta i intervju
- ☐ å delta i datainnsamling gjennom smartsko

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

---

(Signert av prosjektdeltaker, dato)

Interview form

Spørreskjema til bruk i studien «Såle 2.0 – Hvor foten trykker»

Deltaker_ID: _____

Vennligst svar på disse spørsmålene:

1. Omtrentlig vekt:

2. Omtrentlig høyde:

3. Alder:

4. Kjønn:

5. Skostørrelse:

Appendix E

Code for evaluating *test results*

```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
import scipy.interpolate as interpolate
from scipy.signal import filtfilt, butter, detrend, savgol_filter
import pywt
from sklearn import preprocessing
import scipy
import data_processing_functions as dpf
import datetime
import seaborn as sns
import statistics


#Forsøk på et en objektorientert metode


#Objekt som tar inn tidspunkter, trykk, ppg, vekt, vekt_ppg, hc_systolic,
time, butter_level
#Computes filtered_pressure
class Measurement:


    T1TimeArduino = None
    CPTimeArduino = None
    T3TimeArduino = None
    T1TimeChart = None
    CPTimeChart = None
    T3TimeChart = None
    pressFilt = None
    timeLabChart = None
    fs_LabChart = None
    HCSystolic = None
    averagedDiastolic = None
```

```
    fingerPressureHC = None


    def __init__(self, timeStampT1, timeStampCP, timeStampT3,
dataArduinoPath, ChartDataAddress, nameListArduino, nameListChart):
        #Initializes the class. Also reads arduino data
        sns.set()
        SMALL_SIZE = 24
        MEDIUM_SIZE = 10
        BIGGER_SIZE = 24


        plt.rc('font', size=SMALL_SIZE)  # controls default text sizes
        plt.rc('axes', titlesize=SMALL_SIZE)  # fontsize of the axes title
        plt.rc('axes', labelsize=MEDIUM_SIZE)  # fontsize of the x and y
labels
        plt.rc('xtick', labelsize=12)  # fontsize of the tick labels
        plt.rc('ytick', labelsize=12)  # fontsize of the tick labels
        plt.rc('legend', fontsize=SMALL_SIZE)  # legend fontsize
        plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure
title
        #plt.rc('axes', facecolor='beige')



        dataArduino = pd.read_csv(dataArduinoPath, sep=",",
names=nameListArduino, skiprows=1)
        self.arduinoNameList = nameListArduino
        self.chartNameList = nameListChart


        timeArduino = (dataArduino.Time.to_numpy() / 1000)  # divide by
1000 to get seconds
        timeArduino = timeArduino - timeArduino[0]
        self.timeArduino = timeArduino + dataArduino["timestamp"][0]


        self.Pressure0 = dataArduino.Pressure0.to_numpy() #Right foot
furthest back
        self.Pressure1 = dataArduino.Pressure1.to_numpy() #Right foot
second furthest back
        self.Pressure2 = dataArduino.Pressure2.to_numpy() #Right foot
forward to the right
        self.Pressure3 = dataArduino.Pressure3.to_numpy() #Right foot
middle
```

```
        self.Pressure4 = dataArduino.Pressure4.to_numpy() #Left foot
forward right

        self.Pressure5 = dataArduino.Pressure5.to_numpy() #Right foot
forward left

        self.Pressure6 = dataArduino.Pressure6.to_numpy() #Left foot middle

        self.Pressure7 = dataArduino.Pressure7.to_numpy() #Left foot
forward left

        self.Pressure8 = dataArduino.Pressure8.to_numpy() #Left foot second
furthest back

        self.Pressure9 = dataArduino.Pressure9.to_numpy() #Left foot
furthest back


        self.PPG = dataArduino.PPG.to_numpy()

        self.Scale = dataArduino.Scale.to_numpy()

        self.PPG2 = self.PPG*self.PPG


        self.fs_Arduino = (self.timeArduino[-1] - self.timeArduino[0]) /
len(self.timeArduino)

        self.hz_Arduino = 1 / self.fs_Arduino

        self.timeArduinoScale = self.timeArduino[1::2]

        #self.Scale = self.Scale[1::2] #Decided to handle this later so i
can reuse segmentation code

        self.PPGScale = self.PPG[1::2]

        self.fs_Scale = (self.timeArduinoScale[-1] -
self.timeArduinoScale[0]) / len(self.timeArduinoScale) #Que?

        self.dataArduinoPath = dataArduinoPath

        self.chartAddress = ChartDataAddress

        self.T1TimeString = timeStampT1

        self.CPTimeString = timeStampCP

        self.T3TimeString = timeStampT3


        self.pressSegT1 = None

        self.normPressSegT1 = None

        self.avgPressSegT1 = None

        self.pressSegCP = None

        self.normPressSegCP = None

        self.avgPressSegCP = None

        self.pressSegT3 = None

        self.normPressSegT3 = None
```

```
        self.avgPressSegT3 = None


        self.PPGSegT1 = None
        self.normPPGSegT1 = None
        self.avgPPGSegT1 = None
        self.PPGSegCP = None
        self.normPPGSegCP = None
        self.avgPPGSegCP = None
        self.PPGSegT3 = None
        self.normPPGSegT3 = None
        self.avgPPGSegT3 = None


        #computing ddPPG
        #Just testing what filtering the ppg will do - no good is the
conclusion
        #sos = butter(N=12 , Wn=[1,10], btype='bandpass',
fs=self.hz_Arduino, output='sos')
        #self.PPG = scipy.signal.sosfilt(sos, self.PPG)


        polyOrder = 3
        windowSizeSeconds = 0.15 #Change this back to 0.15
        windowSizeArduino = int(windowSizeSeconds / self.fs_Arduino)
        if windowSizeArduino % 2 == 0:
            windowSizeArduino = windowSizeArduino + 1
        self.PPGFilt = savgol_filter(self.PPG, windowSizeArduino,
polyOrder)
        self.dPPGFilt = savgol_filter(self.PPG, windowSizeArduino,
polyOrder, deriv=1)
        self.ddPPGFilt = savgol_filter(self.PPG, windowSizeArduino,
polyOrder, deriv=2)
        #dppg = np.gradient(self.PPGFilt)
        #polyOrder = 3
        #windowSizeSeconds = 0.15 #Change this back to 0.15
        #windowSizeArduino = int(windowSizeSeconds / self.fs_Arduino)
        #if windowSizeArduino % 2 == 0:
        #    windowSizeArduino = windowSizeArduino + 1
        #self.dPPGFilt = savgol_filter(dppg, windowSizeArduino, polyOrder)
        #ddppg = np.gradient(self.dPPGFilt)
```

```
        # Filtering double derivative to identify peaks
        #polyOrder = 3
        #windowSizeSeconds = 0.15 #Change this back to 0.15
        #windowSizeArduino = int(windowSizeSeconds / self.fs_Arduino)
        #if windowSizeArduino % 2 == 0:
        #    windowSizeArduino = windowSizeArduino + 1
        #self.ddPPGFilt = savgol_filter(ddppg, windowSizeArduino,
polyOrder)

        # Need the derivative for the scale measurement as well
        polyOrder = 3
        windowSizeSeconds = 0.15
        windowSizeScale = int(windowSizeSeconds / self.fs_Scale)
        if windowSizeScale % 2 == 0:
            windowSizeScale = windowSizeScale + 1
        self.PPGFiltScale = savgol_filter(self.PPGScale, windowSizeScale,
polyOrder)
        dppgScale = np.gradient(self.PPGFiltScale)

        polyOrder = 3
        windowSizeSeconds = 0.15
        windowSizeScale = int(windowSizeSeconds / self.fs_Scale)
        if windowSizeScale % 2 == 0:
            windowSizeScale = windowSizeScale + 1
        self.dPPGFiltScale = savgol_filter(dppgScale, windowSizeScale,
polyOrder)
        ddppgScale = np.gradient(self.dPPGFiltScale)

        polyOrder = 3
        windowSizeSeconds = 0.15
        windowSizeScale = int(windowSizeSeconds / self.fs_Scale)
        if windowSizeScale % 2 == 0:
            windowSizeScale = windowSizeScale + 1
        self.ddPPGFiltScale = savgol_filter(ddppgScale, windowSizeScale,
polyOrder)

    def filterPressure(self, butterLevel, passBand):
```

```
        sos = butter(N=butterLevel, Wn=passBand, btype='bandpass',
fs=self.hz_Arduino, output='sos')
        self.pressFilt0 = scipy.signal.sosfiltfilt(sos, self.Pressure0)
        self.pressFilt1 = scipy.signal.sosfiltfilt(sos, self.Pressure1)
        self.pressFilt2 = scipy.signal.sosfiltfilt(sos, self.Pressure2)
        self.pressFilt3 = scipy.signal.sosfiltfilt(sos, self.Pressure3)
        self.pressFilt4 = scipy.signal.sosfiltfilt(sos, self.Pressure4)
        self.pressFilt5 = scipy.signal.sosfiltfilt(sos, self.Pressure5)
        self.pressFilt6 = scipy.signal.sosfiltfilt(sos, self.Pressure6)
        self.pressFilt7 = scipy.signal.sosfiltfilt(sos, self.Pressure7)
        self.pressFilt8 = scipy.signal.sosfiltfilt(sos, self.Pressure8)
        self.pressFilt9 = scipy.signal.sosfiltfilt(sos, self.Pressure9)




    def loadChartData(self):
        #reads the labchartdata and extracts the interesting data into a
readable format
        data = pd.read_csv(self.chartAddress, sep="\t",
names=self.chartNameList, header=None)
        startLocations = data.loc[data["Time"] == "Range="]
        indexFinalStart = startLocations.index[-1]
        #data.iloc[indexFinalStart - 5]
        startTimeString = data["Finger_Pressure"][indexFinalStart - 4]
        startTimeList = startTimeString.split()[0].split(".") +
startTimeString.split()[1].split(":")
        startTimeList = startTimeList[:-1] + startTimeList[-1].split(",")
        for i in range(len(startTimeList)):
            startTimeList[i] = int(startTimeList[i])
        epochLabChart = datetime.datetime(startTimeList[2],
startTimeList[1], startTimeList[0], startTimeList[3],
                                          minute=startTimeList[4],
second=startTimeList[5],
                                          microsecond=startTimeList[6] *
1000).timestamp()
        data.drop(list(range(indexFinalStart + 1)), inplace=True)
        numpyArrayList = []
        interestingDataList = ["Time", "HC_Systolic", "Active_Cuff",
"Finger_Pressure_HC"]
```

```
        for i in interestingDataList:
            data[i] = data[i].str.replace(',', '.')
            data[i] = data[i].astype(float)
            data[i] = data[i].fillna(0)
            numpyArrayList.append(data[i].to_numpy())
        timeLabChart = numpyArrayList[0]
        self.timeLabChart = timeLabChart + epochLabChart
        self.fs_LabChart = (self.timeLabChart[-1] - self.timeLabChart[0]) /
len(self.timeLabChart)
        self.HCSystolic = numpyArrayList[1]
        self.averagedDiastolic = numpyArrayList[2]
        self.fingerPressureHC = numpyArrayList[3]


    def findMatchingTime(self, timeArray, timeStamp):
        #Should probably be a private function
        for i in range(len(timeArray)):
            if abs(timeArray[i] - timeStamp) < 0.01:
                return i


    def computeTimeIndice(self, timeSet, timeArray):
        # Timeset should be a list [start, stop] where start and stop
should be written as dd:mm:yy:hh:mm:ss:ms
        epochTimeSet = []
        for i in timeSet:
            timeList = i.split(":")
            for i in range(len(timeList)):
                timeList[i] = int(timeList[i])
            epochTimeStamp = datetime.datetime(timeList[2], timeList[1],
timeList[0], timeList[3],
                                               minute=timeList[4],
second=timeList[5],
                                               microsecond=timeList[6] *
1000).timestamp()
            epochTimeSet.append(self.findMatchingTime(timeArray,
epochTimeStamp))
        timeIndice = epochTimeSet
        return timeIndice
        #convert string time stamps into epoch time stamps
```

```
    def SetTimeIndices(self):

        timeIndice = self.computeTimeIndice(self.T1TimeString,
self.timeArduino)

        self.T1TimeArduino = timeIndice

        timeIndice = self.computeTimeIndice(self.CPTimeString,
self.timeArduino)

        self.CPTimeArduino = timeIndice

        timeIndice = self.computeTimeIndice(self.T3TimeString,
self.timeArduino)

        self.T3TimeArduino = timeIndice

        timeIndice = self.computeTimeIndice(self.T1TimeString,
self.timeLabChart)

        self.T1TimeChart = timeIndice

        timeIndice = self.computeTimeIndice(self.CPTimeString,
self.timeLabChart)

        self.CPTimeChart = timeIndice

        timeIndice = self.computeTimeIndice(self.T3TimeString,
self.timeLabChart)

        self.T3TimeChart = timeIndice


    def averageFingerPressure(self):

        #Computes an average finger pressure by using savitzky golay filter

        polyOrder = 3

        windowSizeSeconds = 35

        windowSize = int(windowSizeSeconds / self.fs_LabChart)

        if windowSize % 2 == 0:

            windowSize = windowSize + 1

        self.HCSystolicFiltered = savgol_filter(self.HCSystolic,
windowSize, polyOrder)


    def plotAvgFingPress(self):

        #Plot the averaged finger pressure

        timeIndexesLabchart = self.T1TimeChart + self.CPTimeChart +
self.T3TimeChart

        plt.figure(figsize=(32, 16), dpi=150)

        plt.ylim(70, 180)

        # plt.plot(HC_Systolic[0:-1])

        # plt.plot(timeLabChart[0:-1], HC_Systolic[0:-1])

        plt.plot(self.timeLabChart[0:-1], self.HCSystolicFiltered[0:-1])
```

```
        for i in timeIndexesLabchart:
            plt.axvline(x=self.timeLabChart[i])
        plt.show()


    def findPPGPeaks(self, threshold, timeBetween):
        #Finds the peaks of the ppg for segmentation
        #Another idea is to find the PPG squared peaks. Should be easier
(maybe)
        samplesBetweenEach = timeBetween/self.fs_Arduino
        self.peaksPPG, properties = scipy.signal.find_peaks(self.PPGFilt,
height=threshold, distance=samplesBetweenEach)
        self.peaksPPG2, properties = scipy.signal.find_peaks(self.PPG2,
height=threshold, distance=samplesBetweenEach)
        samplesBetweenEach = timeBetween/self.fs_Scale
        self.peaksPPGScale, properties =
scipy.signal.find_peaks(self.PPGFiltScale, height=threshold,
distance=samplesBetweenEach)



    def findddPPGPeaks(self, threshold, timeBetween):
        #Finds the peaks of the ddPPG for segmentation
        samplesBetweenEach = timeBetween/self.fs_Arduino
        self.peaksddPPG, properties =
scipy.signal.find_peaks(self.ddPPGFilt, height=threshold,
distance=samplesBetweenEach)
        samplesBetweenEach = timeBetween/self.fs_Scale
        self.peaksddPPGScale, properties =
scipy.signal.find_peaks(self.ddPPGFiltScale, height=threshold,
distance=samplesBetweenEach)


    def organizePeaks(self):
        #I think it is better to do the sanity check here on the peaks.
        #Compute median distance between peaks, then i look through the
gaps between the peaks. If a gap is too long one of the peaks is wrong.
        #But which one?
        #gapLengths = np.zeros(len(self.peaksPPG) - 1)
        #for i in range(len(self.peaksPPG)):
        #    gapLengths[i] = self.peaksPPG[i + 1] - self.peaksPPG[i]
        #medianLength = gapLengths[int(len(gapLengths)/2)]
        #wrongGapIndex = []
```

```
        #for i in range(len(gapLengths)):
        #    if abs(medianLength - gapLengths[i]) > 0.05*medianLength:
        #        #Something is wrong and i should remove one of the peaks.
But im not sure which one.
        #        wrongGapIndex.append(i)
        #Checking which indices are possibly wrong. Whats interesting to
check is wether

        T1Peaks = []
        T3Peaks = []
        CPPeaks = []

        for i in self.peaksPPG:
            if i > self.T1TimeArduino[0] and i < self.T1TimeArduino[1]:
                T1Peaks.append(i)
            elif i > self.CPTimeArduino[0] and i < self.CPTimeArduino[1]:
                CPPeaks.append(i)
            elif i > self.T3TimeArduino[0] and i < self.T3TimeArduino[1]:
                T3Peaks.append(i)

        self.T1Peaks = T1Peaks
        self.CPPeaks = CPPeaks
        self.T3Peaks = T3Peaks

        T1Peaks2 = []
        T3Peaks2 = []
        CPPeaks2 = []

        for i in self.peaksPPG2:
            if i > self.T1TimeArduino[0] and i < self.T1TimeArduino[1]:
                T1Peaks2.append(i)
            elif i > self.CPTimeArduino[0] and i < self.CPTimeArduino[1]:
                CPPeaks2.append(i)
            elif i > self.T3TimeArduino[0] and i < self.T3TimeArduino[1]:
                T3Peaks2.append(i)

        self.T1Peaks2 = T1Peaks2
```

```
        self.CPPeaks2 = CPPeaks2
        self.T3Peaks2 = T3Peaks2


    def sanityCheckSeg(self, segmentList):
        minLength = 100000
        # Identifying the shortest period and making all the segments that
length by cutting off the end
        # Should add some sanity check to remove wrongly identified
segments. For instance by length
        avgLength = 0
        lengthList = []
        for i in range(len(segmentList)):
            lengthList.append(len(segmentList[i]))
        #median = lengthList[int(len(lengthList)/2)]
        median = statistics.median(lengthList)

        for j in range(len(segmentList)):
        #    avgLength = avgLength + len(segmentList[j])
            if minLength > len(segmentList[j]):
                minLength = len(segmentList[j])
        avgLength = avgLength / len(segmentList)
        avgLength = avgLength + 0.1 * avgLength  # Not in use atm
        popList = []
        for k in range(len(segmentList)):
            if abs(len(segmentList[k]) - median) > 0.05*median:
                # pressureSegmentList.pop(k)
                popList.append(k) #List of the indices im gonna pop
(remove)
            else:
                segmentList[k] = segmentList[k][0:minLength] #Setting all
segments to the same length for easier plotting
        for l in range(len(popList)):
            segmentList.pop(popList[l] - l)
        #print(popList)
        return segmentList, popList


    def segmentUsingPPGT1(self, peaks, fs, press=False, PPG=False):
```

```
        #Segments the data by using the identified PPG peaks
        #How do i choose which peaks i want to use?
        #Should change this to just use self.peaksT1. Doesnt need me to
give it peaks
        if press:
            pressureSegmentList0 = []
            pressureSegmentList1 = []
            pressureSegmentList2 = []
            pressureSegmentList3 = []
            pressureSegmentList4 = []
            pressureSegmentList5 = []
            pressureSegmentList6 = []
            pressureSegmentList7 = []
            pressureSegmentList8 = []
            pressureSegmentList9 = []
            for i in range(len(peaks) - 1):
                # Takes a subset from data from peak1 to peak2 and pad
about 0.15 seconds on each side
                segment0 = self.pressFilt0[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment1 = self.pressFilt1[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment2 = self.pressFilt2[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment3 = self.pressFilt3[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment4 = self.pressFilt4[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment5 = self.pressFilt5[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment6 = self.pressFilt6[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment7 = self.pressFilt7[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment8 = self.pressFilt8[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment9 = self.pressFilt9[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]

                pressureSegmentList0.append(segment0)
                pressureSegmentList1.append(segment1)
```

```
                pressureSegmentList2.append(segment2)
                pressureSegmentList3.append(segment3)
                pressureSegmentList4.append(segment4)
                pressureSegmentList5.append(segment5)
                pressureSegmentList6.append(segment6)
                pressureSegmentList7.append(segment7)
                pressureSegmentList8.append(segment8)
                pressureSegmentList9.append(segment9)

                #Doing sanity checks

            self.pressureSegmentList0T1, popList =
self.sanityCheckSeg(pressureSegmentList0)
            self.pressureSegmentList1T1, popList =
self.sanityCheckSeg(pressureSegmentList1)
            self.pressureSegmentList2T1, popList =
self.sanityCheckSeg(pressureSegmentList2)
            self.pressureSegmentList3T1, popList =
self.sanityCheckSeg(pressureSegmentList3)
            self.pressureSegmentList4T1, popList =
self.sanityCheckSeg(pressureSegmentList4)
            self.pressureSegmentList5T1, popList =
self.sanityCheckSeg(pressureSegmentList5)
            self.pressureSegmentList6T1, popList =
self.sanityCheckSeg(pressureSegmentList6)
            self.pressureSegmentList7T1, popList =
self.sanityCheckSeg(pressureSegmentList7)
            self.pressureSegmentList8T1, popList =
self.sanityCheckSeg(pressureSegmentList8)
            self.pressureSegmentList9T1, popList =
self.sanityCheckSeg(pressureSegmentList9)
            self.popListT1 = popList
            #print("PopListT1: ", self.popListT1)
        elif PPG:
            PPGSegmentList = []
            for i in range(len(peaks) - 1):
                # Takes a subset from data from peak1 to peak2 and pad
about 0.15 seconds on each side
                segment = self.PPGFilt[peaks[i] - int(0.5 / fs): peaks[i +
1] + int(0.5 / fs)]
                PPGSegmentList.append(segment)
```

```
            PPGSegmentList, popList = self.sanityCheckSeg(PPGSegmentList)
            self.PPGSegmentListT1 = PPGSegmentList
            self.popListT1 = popList
            #print("PopListT1: ", self.popListT1)

    def segmentUsingPPGCP(self, peaks, fs, press=False, PPG=False):
        #Segments the data by using the identified PPG peaks
        #How do i choose which peaks i want to use?

        if press:
            pressureSegmentList0 = []
            pressureSegmentList1 = []
            pressureSegmentList2 = []
            pressureSegmentList3 = []
            pressureSegmentList4 = []
            pressureSegmentList5 = []
            pressureSegmentList6 = []
            pressureSegmentList7 = []
            pressureSegmentList8 = []
            pressureSegmentList9 = []
            for i in range(len(peaks) - 1):
                # Takes a subset from data from peak1 to peak2 and pad
about 0.15 seconds on each side
                segment0 = self.pressFilt0[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment1 = self.pressFilt1[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment2 = self.pressFilt2[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment3 = self.pressFilt3[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment4 = self.pressFilt4[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment5 = self.pressFilt5[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment6 = self.pressFilt6[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment7 = self.pressFilt7[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
```

```
                segment8 = self.pressFilt8[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]

                segment9 = self.pressFilt9[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]


                pressureSegmentList0.append(segment0)

                pressureSegmentList1.append(segment1)

                pressureSegmentList2.append(segment2)

                pressureSegmentList3.append(segment3)

                pressureSegmentList4.append(segment4)

                pressureSegmentList5.append(segment5)

                pressureSegmentList6.append(segment6)

                pressureSegmentList7.append(segment7)

                pressureSegmentList8.append(segment8)

                pressureSegmentList9.append(segment9)


                #Doing sanity checks

            self.pressureSegmentList0CP, popList =
self.sanityCheckSeg(pressureSegmentList0)

            self.pressureSegmentList1CP, popList =
self.sanityCheckSeg(pressureSegmentList1)

            self.pressureSegmentList2CP, popList =
self.sanityCheckSeg(pressureSegmentList2)

            self.pressureSegmentList3CP, popList =
self.sanityCheckSeg(pressureSegmentList3)

            self.pressureSegmentList4CP, popList =
self.sanityCheckSeg(pressureSegmentList4)

            self.pressureSegmentList5CP, popList =
self.sanityCheckSeg(pressureSegmentList5)

            self.pressureSegmentList6CP, popList =
self.sanityCheckSeg(pressureSegmentList6)

            self.pressureSegmentList7CP, popList =
self.sanityCheckSeg(pressureSegmentList7)

            self.pressureSegmentList8CP, popList =
self.sanityCheckSeg(pressureSegmentList8)

            self.pressureSegmentList9CP, popList =
self.sanityCheckSeg(pressureSegmentList9)

            self.popListCP = popList

        elif PPG:

            PPGSegmentList = []

            for i in range(len(peaks) - 1):
```

```
                # Takes a subset from data from peak1 to peak2 and pad
about 0.15 seconds on each side
                segment = self.PPGFilt[peaks[i] - int(0.5 / fs): peaks[i +
1] + int(
                0.5 / fs)]
                PPGSegmentList.append(segment)
            PPGSegmentList, popList = self.sanityCheckSeg(PPGSegmentList)
            self.PPGSegmentListCP = PPGSegmentList
            self.popListCP = popList
    def segmentUsingPPGT3(self, peaks, fs, press=False, PPG=False):
        #Segments the data by using the identified PPG peaks
        #How do i choose which peaks i want to use?

        if press:
            pressureSegmentList0 = []
            pressureSegmentList1 = []
            pressureSegmentList2 = []
            pressureSegmentList3 = []
            pressureSegmentList4 = []
            pressureSegmentList5 = []
            pressureSegmentList6 = []
            pressureSegmentList7 = []
            pressureSegmentList8 = []
            pressureSegmentList9 = []
            for i in range(len(peaks) - 1):
                # Takes a subset from data from peak1 to peak2 and pad
about 0.15 seconds on each side
                segment0 = self.pressFilt0[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment1 = self.pressFilt1[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment2 = self.pressFilt2[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment3 = self.pressFilt3[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment4 = self.pressFilt4[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
                segment5 = self.pressFilt5[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]
```

```
                segment6 = self.pressFilt6[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]

                segment7 = self.pressFilt7[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]

                segment8 = self.pressFilt8[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]

                segment9 = self.pressFilt9[peaks[i] - int(0.5 / fs):
peaks[i + 1] + int(0.5 / fs)]


                pressureSegmentList0.append(segment0)

                pressureSegmentList1.append(segment1)

                pressureSegmentList2.append(segment2)

                pressureSegmentList3.append(segment3)

                pressureSegmentList4.append(segment4)

                pressureSegmentList5.append(segment5)

                pressureSegmentList6.append(segment6)

                pressureSegmentList7.append(segment7)

                pressureSegmentList8.append(segment8)

                pressureSegmentList9.append(segment9)


                #Doing sanity checks

            self.pressureSegmentList0T3, popList =
self.sanityCheckSeg(pressureSegmentList0)

            self.pressureSegmentList1T3, popList =
self.sanityCheckSeg(pressureSegmentList1)

            self.pressureSegmentList2T3, popList =
self.sanityCheckSeg(pressureSegmentList2)

            self.pressureSegmentList3T3, popList =
self.sanityCheckSeg(pressureSegmentList3)

            self.pressureSegmentList4T3, popList =
self.sanityCheckSeg(pressureSegmentList4)

            self.pressureSegmentList5T3, popList =
self.sanityCheckSeg(pressureSegmentList5)

            self.pressureSegmentList6T3, popList =
self.sanityCheckSeg(pressureSegmentList6)

            self.pressureSegmentList7T3, popList =
self.sanityCheckSeg(pressureSegmentList7)

            self.pressureSegmentList8T3, popList =
self.sanityCheckSeg(pressureSegmentList8)

            self.pressureSegmentList9T3, popList =
self.sanityCheckSeg(pressureSegmentList9)

            self.popListT3 = popList
```

```
        elif PPG:
            PPGSegmentList = []
            for i in range(len(peaks) - 1):
                # Takes a subset from data from peak1 to peak2 and pad
about 0.15 seconds on each side
                segment = self.PPGFilt[peaks[i] - int(0.5 / fs): peaks[i +
1] + int(
                0.5 / fs)]  # This will probably make wrongly segmented
segments worse.
                PPGSegmentList.append(segment)
            PPGSegmentList, popList = self.sanityCheckSeg(PPGSegmentList)
            self.PPGSegmentListT3 = PPGSegmentList
            self.popListT3 = popList


    def normAndAvgPressSegT1(self):
        #Shit - This does not consider T1, CP and T3 - Start here next time
(22.05.2022) and then fix plotting.
        normSeg0 = []
        for i in self.pressureSegmentList0T1:
            normSeg0.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg0T1 = normSeg0
        self.avgSeg0T1 = dpf.averageSegments(segments=normSeg0)


        normSeg1 = []
        for i in self.pressureSegmentList1T1:
            normSeg1.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg1T1 = normSeg1
        self.avgSeg1T1 = dpf.averageSegments(segments=normSeg1)


        normSeg2 = []
        for i in self.pressureSegmentList2T1:
            normSeg2.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg2T1 = normSeg2
        self.avgSeg2T1 = dpf.averageSegments(segments=normSeg2)
```

```
        normSeg3 = []
        for i in self.pressureSegmentList3T1:
            normSeg3.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg3T1 = normSeg3
        self.avgSeg3T1 = dpf.averageSegments(segments=normSeg3)


        normSeg4 = []
        for i in self.pressureSegmentList4T1:
            normSeg4.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg4T1 = normSeg4
        self.avgSeg4T1 = dpf.averageSegments(segments=normSeg4)


        normSeg5 = []
        for i in self.pressureSegmentList5T1:
            normSeg5.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg5T1 = normSeg5
        self.avgSeg5T1 = dpf.averageSegments(segments=normSeg5)


        normSeg6 = []
        for i in self.pressureSegmentList6T1:
            normSeg6.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg6T1 = normSeg6
        self.avgSeg6T1 = dpf.averageSegments(segments=normSeg6)


        normSeg7 = []
        for i in self.pressureSegmentList7T1:
            normSeg7.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg7T1 = normSeg7
        self.avgSeg7T1 = dpf.averageSegments(segments=normSeg7)


        normSeg8 = []
        for i in self.pressureSegmentList8T1:
            normSeg8.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
```

```
        self.normSeg8T1 = normSeg8
        self.avgSeg8T1 = dpf.averageSegments(segments=normSeg8)


        normSeg9 = []
        for i in self.pressureSegmentList9T1:
            normSeg9.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg9T1 = normSeg9
        self.avgSeg9T1 = dpf.averageSegments(segments=normSeg9)


    def normAndAvgPressSegCP(self):
        #Shit - This does not consider T1, CP and T3 - Start here next time
(22.05.2022) and then fix plotting.
        normSeg0 = []
        for i in self.pressureSegmentList0CP:
            normSeg0.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg0CP = normSeg0
        self.avgSeg0CP = dpf.averageSegments(segments=normSeg0)


        normSeg1 = []
        for i in self.pressureSegmentList1CP:
            normSeg1.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg1CP = normSeg1
        self.avgSeg1CP = dpf.averageSegments(segments=normSeg1)


        normSeg2 = []
        for i in self.pressureSegmentList2CP:
            normSeg2.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg2CP = normSeg2
        self.avgSeg2CP = dpf.averageSegments(segments=normSeg2)


        normSeg3 = []
        for i in self.pressureSegmentList3CP:
            normSeg3.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg3CP = normSeg3
```

```
        self.avgSeg3CP = dpf.averageSegments(segments=normSeg3)


        normSeg4 = []
        for i in self.pressureSegmentList4CP:
            normSeg4.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg4CP = normSeg4
        self.avgSeg4CP = dpf.averageSegments(segments=normSeg4)


        normSeg5 = []
        for i in self.pressureSegmentList5CP:
            normSeg5.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg5CP = normSeg5
        self.avgSeg5CP = dpf.averageSegments(segments=normSeg5)


        normSeg6 = []
        for i in self.pressureSegmentList6CP:
            normSeg6.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg6CP = normSeg6
        self.avgSeg6CP = dpf.averageSegments(segments=normSeg6)


        normSeg7 = []
        for i in self.pressureSegmentList7CP:
            normSeg7.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg7CP = normSeg7
        self.avgSeg7CP = dpf.averageSegments(segments=normSeg7)


        normSeg8 = []
        for i in self.pressureSegmentList8CP:
            normSeg8.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg8CP = normSeg8
        self.avgSeg8CP = dpf.averageSegments(segments=normSeg8)


        normSeg9 = []
```

```
        for i in self.pressureSegmentList9CP:
            normSeg9.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg9CP = normSeg9
        self.avgSeg9CP = dpf.averageSegments(segments=normSeg9)


    def normAndAvgPressSegT3(self):
        #Shit - This does not consider T1, CP and T3 - Start here next time
(22.05.2022) and then fix plotting.
        normSeg0 = []
        for i in self.pressureSegmentList0T3:
            normSeg0.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg0T3 = normSeg0
        self.avgSeg0T3 = dpf.averageSegments(segments=normSeg0)


        normSeg1 = []
        for i in self.pressureSegmentList1T3:
            normSeg1.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg1T3 = normSeg1
        self.avgSeg1T3 = dpf.averageSegments(segments=normSeg1)


        normSeg2 = []
        for i in self.pressureSegmentList2T3:
            normSeg2.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg2T3 = normSeg2
        self.avgSeg2T3 = dpf.averageSegments(segments=normSeg2)


        normSeg3 = []
        for i in self.pressureSegmentList3T3:
            normSeg3.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg3T3 = normSeg3
        self.avgSeg3T3 = dpf.averageSegments(segments=normSeg3)


        normSeg4 = []
        for i in self.pressureSegmentList4T3:
```

```
            normSeg4.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg4T3 = normSeg4
        self.avgSeg4T3 = dpf.averageSegments(segments=normSeg4)


        normSeg5 = []
        for i in self.pressureSegmentList5T3:
            normSeg5.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg5T3 = normSeg5
        self.avgSeg5T3 = dpf.averageSegments(segments=normSeg5)


        normSeg6 = []
        for i in self.pressureSegmentList6T3:
            normSeg6.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg6T3 = normSeg6
        self.avgSeg6T3 = dpf.averageSegments(segments=normSeg6)


        normSeg7 = []
        for i in self.pressureSegmentList7T3:
            normSeg7.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg7T3 = normSeg7
        self.avgSeg7T3 = dpf.averageSegments(segments=normSeg7)


        normSeg8 = []
        for i in self.pressureSegmentList8T3:
            normSeg8.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg8T3 = normSeg8
        self.avgSeg8T3 = dpf.averageSegments(segments=normSeg8)


        normSeg9 = []
        for i in self.pressureSegmentList9T3:
            normSeg9.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSeg9T3 = normSeg9
        self.avgSeg9T3 = dpf.averageSegments(segments=normSeg9)
```

```
    def normAndAvgPPGSegT1(self):
        normSeg = []
        for i in self.PPGSegmentListT1:
            normSeg.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSegPPGT1 = normSeg
        self.avgSegPPGT1 = dpf.averageSegments(segments=normSeg)


    def normAndAvgPPGSegCP(self):
        normSeg = []
        for i in self.PPGSegmentListCP:
            normSeg.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSegPPGCP = normSeg
        self.avgSegPPGCP = dpf.averageSegments(segments=normSeg)


    def normAndAvgPPGSegT3(self):
        normSeg = []
        for i in self.PPGSegmentListT3:
            normSeg.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.normSegPPGT3 = normSeg
        self.avgSegPPGT3 = dpf.averageSegments(segments=normSeg)


    def segmentNormalizeAndAverage(self, fs, segPress, segPPG, PPG2 =
False, ddPPG = False, segScale = False, T1=False, CP=False, T3=False):
        #ref is the reference we use to segments. Called peaks later
        #ddPPG is not fixed


        if T1:
            if ddPPG:
                ref =
self.ddPPGFilt[self.T1TimeArduino[0]:self.T1TimeArduino[1]]
            elif PPG2:
                ref = self.T1Peaks2
            else:
                #ref =
self.PPG[self.T1TimeArduino[0]:self.T1TimeArduino[1]]
```

```
                ref = self.T1Peaks
            self.segmentUsingPPGT1(ref, fs, press=segPress, PPG=segPPG)
            if segPress:
                self.normAndAvgPressSegT1()
            elif segPPG:
                self.normAndAvgPPGSegT1()


        elif CP:
            if ddPPG:
                ref =
self.ddPPGFilt[self.CPTimeArduino[0]:self.CPTimeArduino[1]]
            elif PPG2:
                ref = self.CPPeaks2
            else:
                #ref =
self.PPG[self.CPTimeArduino[0]:self.CPTimeArduino[1]]
                ref = self.CPPeaks
            self.segmentUsingPPGCP(ref, fs, press=segPress, PPG=segPPG)
            if segPress:
                self.normAndAvgPressSegCP()
            if segPPG:
                self.normAndAvgPPGSegCP()
        elif T3:
            if ddPPG:
                ref =
self.ddPPGFilt[self.T3TimeArduino[0]:self.T3TimeArduino[1]]
            elif PPG2:
                ref = self.T1Peaks2
            else:
                #ref =
self.PPG[self.T3TimeArduino[0]:self.T3TimeArduino[1]]
                ref = self.T3Peaks
            self.segmentUsingPPGT3(ref, fs, press=segPress, PPG=segPPG)
            if segPress:
                self.normAndAvgPressSegT3()
            if segPPG:
                self.normAndAvgPPGSegT3()
```

```
    def segmentPressure(self, ddPPG = False, PPG2 = False): #Should change
the name of the variable for wether im segmenting ppg

        self.segmentNormalizeAndAverage(self.fs_Arduino, segPress=True,
segPPG=False, PPG2=PPG2, ddPPG = ddPPG, T1=True)

        self.segmentNormalizeAndAverage(self.fs_Arduino, segPress=True,
segPPG=False, PPG2=PPG2, ddPPG=ddPPG, CP=True)

        self.segmentNormalizeAndAverage(self.fs_Arduino, segPress=True,
segPPG=False, PPG2=PPG2, ddPPG=ddPPG, T3=True)


    def segmentScalePressure(self, threshold, timeBetween): #Needs to be
updated

        self.segmentNormalizeAndAverage(self.Scale, self.fs_Scale,
threshold, timeBetween, T1 = True)

        self.segmentNormalizeAndAverage(self.Scale, self.fs_Scale,
threshold, timeBetween, CP = True)

        self.segmentNormalizeAndAverage(self.Scale, self.fs_Scale,
threshold, timeBetween, T3 = True)


    def segmentPPG(self, ddPPG = False, PPG2 = False):

        self.segmentNormalizeAndAverage(self.fs_Arduino, segPress=False,
segPPG=True, PPG2=PPG2, ddPPG=ddPPG, T1=True)

        self.segmentNormalizeAndAverage(self.fs_Arduino, segPress=False,
segPPG=True, PPG2=PPG2, ddPPG=ddPPG, CP=True)

        self.segmentNormalizeAndAverage(self.fs_Arduino, segPress=False,
segPPG=True, PPG2=PPG2, ddPPG=ddPPG, T3=True)


    def plotSegPress(self, pressSelect, T1 = False, CP = False, T3 = False,
savePlot = False, ID = ""):

        #Might be better to just make this 9 functions. One for each sensor

        if pressSelect == 0:

            if T1:

                avgData = self.avgSeg0T1

                normData = self.normSeg0T1

            elif CP:

                avgData = self.avgSeg0CP

                normData = self.normSeg0CP

            elif T3:

                avgData = self.avgSeg0T3

                normData = self.normSeg0T3

        elif pressSelect == 1:
```

```
    if T1:
        avgData = self.avgSeg1T1
        normData = self.normSeg1T1
    elif CP:
        avgData = self.avgSeg1CP
        normData = self.normSeg1CP
    elif T3:
        avgData = self.avgSeg1T3
        normData = self.normSeg1T3
elif pressSelect == 2:
    if T1:
        avgData = self.avgSeg2T1
        normData = self.normSeg2T1
    elif CP:
        avgData = self.avgSeg2CP
        normData = self.normSeg2CP
    elif T3:
        avgData = self.avgSeg2T3
        normData = self.normSeg2T3
elif pressSelect == 3:
    if T1:
        avgData = self.avgSeg3T1
        normData = self.normSeg3T1
    elif CP:
        avgData = self.avgSeg3CP
        normData = self.normSeg3CP
    elif T3:
        avgData = self.avgSeg3T3
        normData = self.normSeg3T3
elif pressSelect == 4:
    if T1:
        avgData = self.avgSeg4T1
        normData = self.normSeg4T1
    elif CP:
        avgData = self.avgSeg4CP
        normData = self.normSeg4CP
```

```
    elif T3:
        avgData = self.avgSeg4T3
        normData = self.normSeg4T3
elif pressSelect == 5:
    if T1:
        avgData = self.avgSeg5T1
        normData = self.avgSeg5T3
    elif CP:
        avgData = self.avgSeg5CP
        normData = self.normSeg5CP
    elif T3:
        avgData = self.avgSeg5T3
        normData = self.normSeg5T3
elif pressSelect == 6:
    if T1:
        avgData = self.avgSeg6T1
        normData = self.normSeg6T1
    elif CP:
        avgData = self.avgSeg6CP
        normData = self.normSeg6CP
    elif T3:
        avgData = self.avgSeg6T3
        normData = self.normSeg6T3
elif pressSelect == 7:
    if T1:
        avgData = self.avgSeg7T3
        normData = self.normSeg7T1
    elif CP:
        avgData = self.avgSeg7CP
        normData = self.normSeg7CP
    elif T3:
        avgData = self.avgSeg7T3
        normData = self.avgSeg2T3
elif pressSelect == 8:
    if T1:
        avgData = self.avgSeg8T1
```

```
                normData = self.normSeg8T1
            elif CP:
                avgData = self.avgSeg8CP
                normData = self.normSeg8CP
            elif T3:
                avgData = self.avgSeg8T3
                normData = self.normSeg8T3
        elif pressSelect == 9:
            if T1:
                avgData = self.avgSeg9T1
                normData = self.normSeg9T1
            elif CP:
                avgData = self.avgSeg9CP
                normData = self.normSeg9CP
            elif T3:
                avgData = self.avgSeg9T3
                normData = self.normSeg9T3



        plt.figure(figsize=(32, 16), dpi=150)
        plt.plot(
            np.linspace(start=0, stop=((len(avgData)) * self.fs_Arduino),
num=(len(avgData))),
            avgData[:], "r", linewidth=4)
        for i in range(len(normData)):
            plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,
                                 num=(len(normData[i]))), normData[i], "b--
")
        plt.grid(axis='x')
        plt.title("Pressure segmented and averaged")
        plt.xticks(np.arange(start=0, stop=((len(avgData)) *
self.fs_Arduino), step=0.05))
        if savePlot:
            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\SegPress_" + ID
            plt.savefig(saveName)
        plt.show()
```

```
    def plotSegPressV2(self, pressSelect, T1=False, CP=False, T3=False,
savePlot = False, ID = ""):

        # Might be better to just make this 9 functions. One for each
sensor

        if pressSelect == 0:

            if T1:

                avgData = self.avgSeg0T1

                normData = self.normSeg0T1

            elif CP:

                avgData = self.avgSeg0CP

                normData = self.normSeg0CP

            elif T3:

                avgData = self.avgSeg0T3

                normData = self.normSeg0T3

        elif pressSelect == 1:

            if T1:

                avgData = self.avgSeg1T1

                normData = self.normSeg1T1

            elif CP:

                avgData = self.avgSeg1CP

                normData = self.normSeg1CP

            elif T3:

                avgData = self.avgSeg1T3

                normData = self.normSeg1T3

        elif pressSelect == 2:

            if T1:

                avgData = self.avgSeg2T1

                normData = self.normSeg2T1

            elif CP:

                avgData = self.avgSeg2CP

                normData = self.normSeg2CP

            elif T3:

                avgData = self.avgSeg2T3

                normData = self.normSeg2T3

        elif pressSelect == 3:

            if T1:
```

```
            avgData = self.avgSeg3T1
            normData = self.normSeg3T1
        elif CP:
            avgData = self.avgSeg3CP
            normData = self.normSeg3CP
        elif T3:
            avgData = self.avgSeg3T3
            normData = self.normSeg3T3
    elif pressSelect == 4:
        if T1:
            avgData = self.avgSeg4T1
            normData = self.normSeg4T1
        elif CP:
            avgData = self.avgSeg4CP
            normData = self.normSeg4CP
        elif T3:
            avgData = self.avgSeg4T3
            normData = self.normSeg4T3
    elif pressSelect == 5:
        if T1:
            avgData = self.avgSeg5T1
            normData = self.avgSeg5T3
        elif CP:
            avgData = self.avgSeg5CP
            normData = self.normSeg5CP
        elif T3:
            avgData = self.avgSeg5T3
            normData = self.normSeg5T3
    elif pressSelect == 6:
        if T1:
            avgData = self.avgSeg6T1
            normData = self.normSeg6T1
        elif CP:
            avgData = self.avgSeg6CP
            normData = self.normSeg6CP
        elif T3:
```

```
            avgData = self.avgSeg6T3
            normData = self.normSeg6T3
        elif pressSelect == 7:
            if T1:
                avgData = self.avgSeg7T3
                normData = self.normSeg7T1
            elif CP:
                avgData = self.avgSeg7CP
                normData = self.normSeg7CP
            elif T3:
                avgData = self.avgSeg7T3
                normData = self.avgSeg2T3
        elif pressSelect == 8:
            if T1:
                avgData = self.avgSeg8T1
                normData = self.normSeg8T1
            elif CP:
                avgData = self.avgSeg8CP
                normData = self.normSeg8CP
            elif T3:
                avgData = self.avgSeg8T3
                normData = self.normSeg8T3
        elif pressSelect == 9:
            if T1:
                avgData = self.avgSeg9T1
                normData = self.normSeg9T1
            elif CP:
                avgData = self.avgSeg9CP
                normData = self.normSeg9CP
            elif T3:
                avgData = self.avgSeg9T3
                normData = self.normSeg9T3
        fig, ax1 = plt.subplots(figsize=(32, 16), dpi = 150)
        ax1.plot(np.linspace(start=0, stop=((len(avgData)) *
self.fs_Arduino), num=(len(avgData))),
                 avgData[:], "r", linewidth=4)
        #plt.plot(
```

```
        #    np.linspace(start=0, stop=((len(avgData)) * self.fs_Arduino),
num=(len(avgData))),

        #    avgData[:], "r", linewidth=4)

        axisList = []

        for i in range(len(normData)):

            axisList.append(ax1.twinx())

            axisList[i].plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino, num=(len(normData[i]))),

                             normData[i], "b--")

            fig.axes[i].get_yaxis().set_visible(False)

            #plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,

            #                     num=(len(normData[i]))), normData[i], "b-
-")

        plt.grid(axis='x')

        plt.title("Pressure segmented and averaged plotV2")

        plt.xticks(np.arange(start=0, stop=((len(avgData)) *
self.fs_Arduino), step=0.05))

        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\SegPressV2_" + ID

            plt.savefig(saveName)

        plt.show()


    def plotAllPressureSegAndAvg(self, ID, T1 = False, CP = False, T3 =
False, savePlot = False):

        if T1:

            pressSegs = [self.normSeg0T1, self.normSeg1T1, self.normSeg2T1,
self.normSeg3T1, self.normSeg4T1, self.normSeg5T1,

                         self.normSeg6T1, self.normSeg7T1, self.normSeg8T1,
self.normSeg9T1]

            pressAvgs = [self.avgSeg0T1, self.avgSeg1T1, self.avgSeg2T1,
self.avgSeg3T1, self.avgSeg4T1, self.avgSeg5T1,

                         self.avgSeg6T1, self.avgSeg7T1, self.avgSeg8T1,
self.avgSeg9T1]

        elif CP:

            pressSegs = [self.normSeg0CP, self.normSeg1CP, self.normSeg2CP,
self.normSeg3CP, self.normSeg4CP, self.normSeg5CP,

                         self.normSeg6CP, self.normSeg7CP, self.normSeg8CP,
self.normSeg9CP]

            pressAvgs = [self.avgSeg0CP, self.avgSeg1CP, self.avgSeg2CP,
self.avgSeg3CP, self.avgSeg4CP, self.avgSeg5CP,
```

```
                    self.avgSeg6CP, self.avgSeg7CP, self.avgSeg8CP,
self.avgSeg9CP]
        elif T3:
            pressSegs = [self.normSeg0T3, self.normSeg1T3, self.normSeg2T3,
self.normSeg3T3, self.normSeg4T3, self.normSeg5T3,
                    self.normSeg6T3, self.normSeg7T3, self.normSeg8T3,
self.normSeg9T3]
            pressAvgs = [self.avgSeg0T3, self.avgSeg1T3, self.avgSeg2T3,
self.avgSeg3T3, self.avgSeg4T3, self.avgSeg5T3,
                    self.avgSeg6T3, self.avgSeg7T3, self.avgSeg8T3,
self.avgSeg9T3]
        posList = ["right 1", "right 2", "right 5", "right 3", "left 5",
"right 4", "left 3", "left 4", "left 2", "left 1"]
        #self.Pressure0 = dataArduino.Pressure0.to_numpy() #Right foot
furthest back
        #self.Pressure1 = dataArduino.Pressure1.to_numpy() #Right foot
second furthest back
        #self.Pressure2 = dataArduino.Pressure2.to_numpy() #Right foot
forward to the right
        #self.Pressure3 = dataArduino.Pressure3.to_numpy() #Right foot
middle
        #self.Pressure4 = dataArduino.Pressure4.to_numpy() #Left foot
forward right
        #self.Pressure5 = dataArduino.Pressure5.to_numpy() #Right foot
forward left
        #self.Pressure6 = dataArduino.Pressure6.to_numpy() #Left foot
middle
        #self.Pressure7 = dataArduino.Pressure7.to_numpy() #Left foot
forward left
        #self.Pressure8 = dataArduino.Pressure8.to_numpy() #Left foot
second furthest back
        #self.Pressure9 = dataArduino.Pressure9.to_numpy() #Left foot
furthest back
        #Testing something here
        for i in range(len(pressSegs)):
            fig, ax1 = plt.subplots(figsize=(32, 16), dpi=150)
            ax1.plot(np.linspace(start=0, stop=((len(pressAvgs[i])) *
self.fs_Arduino), num=(len(pressAvgs[i]))),
                     pressAvgs[i][:], "r", linewidth=4)
        # plt.plot(
        #    np.linspace(start=0, stop=((len(avgData)) * self.fs_Arduino),
num=(len(avgData))),
        #    avgData[:], "r", linewidth=4)
            axisList = []
```

```
            ax2 = ax1.twinx()

            for j in range(len(pressSegs[i])):

                #axisList.append(ax1.twinx())

                #axisList[j].plot(np.linspace(start=0,
stop=(len(pressSegs[i][j])) * self.fs_Arduino, num=(len(pressSegs[i][j]))),

                #                pressSegs[i][j], "b--")

                #fig.axes[j].get_yaxis().set_visible(False)

                ax2.plot(np.linspace(start=0, stop=(len(pressSegs[i][j])) *
self.fs_Arduino, num=(len(pressSegs[i][j]))),

                         pressSegs[i][j], "b--")

            # plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,

            #                      num=(len(normData[i]))), normData[i], "b-
-")

            plt.grid(axis='x')

            plt.title("Pressure segmented and averaged " + "position " +
posList[i] + " - " + ID)

            plt.xticks(np.arange(start=0, stop=((len(pressAvgs[i])) *
self.fs_Arduino), step=0.05))

            if savePlot:

                saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots2\\SegPressV2_Press" +
str(i) + ID

                plt.savefig(saveName)

            #plt.show()


    def segmentScale(self):

        #alright so i have the peaks for the normal PPG but not the PPG
scale. I can probably take their position use that to find the scale
segments then

        #shorten the scale segments. Lets actually try that

        peaks = self.T1Peaks

        scaleSegmentList = []

        for i in range(len(peaks) - 1):

            # Takes a subset from data from peak1 to peak2 and pad about
0.15 seconds on each side

            segment = self.Scale[peaks[i] - int(0.5 / self.fs_Arduino):
peaks[i + 1] + int(

                0.5 / self.fs_Arduino)]  # This will probably make wrongly
segmented segments worse.

            scaleSegmentList.append(segment)

        scaleSegmentList, popList = self.sanityCheckSeg(scaleSegmentList)
```

```
        for i in range(len(scaleSegmentList)):
            scaleSegmentList[i] = scaleSegmentList[i][1::2]
#         scaleSegmentList = scaleSegmentList[1::2]
        self.scaleSegmentListT1 = scaleSegmentList
        self.popListScaleT1 = popList
        normScale=[]
        for i in self.scaleSegmentListT1:
            normScale.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.scaleSegmentListT1 = normScale
        self.avgScaleSegT1 =
dpf.averageSegments(segments=self.scaleSegmentListT1)



        peaks = self.CPPeaks
        scaleSegmentList = []
        for i in range(len(peaks) - 1):
            # Takes a subset from data from peak1 to peak2 and pad about
0.15 seconds on each side
            segment = self.Scale[peaks[i] - int(0.5 / self.fs_Arduino):
peaks[i + 1] + int(
                0.5 / self.fs_Arduino)]  # This will probably make wrongly
segmented segments worse.
            scaleSegmentList.append(segment)
        scaleSegmentList, popList = self.sanityCheckSeg(scaleSegmentList)
        for i in range(len(scaleSegmentList)):
            scaleSegmentList[i] = scaleSegmentList[i][1::2]
        self.scaleSegmentListCP = scaleSegmentList
        self.popListScaleCP = popList
        normScale=[]
        for i in self.scaleSegmentListCP:
            normScale.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())
        self.scaleSegmentListCP = normScale
        self.avgScaleSegCP =
dpf.averageSegments(segments=self.scaleSegmentListCP)



        peaks = self.T3Peaks
```

```
        scaleSegmentList = []

        for i in range(len(peaks) - 1):

            # Takes a subset from data from peak1 to peak2 and pad about
0.15 seconds on each side

            segment = self.Scale[peaks[i] - int(0.5 / self.fs_Arduino):
peaks[i + 1] + int(

                0.5 / self.fs_Arduino)]  # This will probably make wrongly
segmented segments worse.

            scaleSegmentList.append(segment)

        scaleSegmentList, popList = self.sanityCheckSeg(scaleSegmentList)

        for i in range(len(scaleSegmentList)):

            scaleSegmentList[i] = scaleSegmentList[i][1::2]

#         scaleSegmentList = scaleSegmentList[1::2]

        self.scaleSegmentListT3 = scaleSegmentList

        self.popListScaleT3 = popList

        normScale=[]

        for i in self.scaleSegmentListT3:

            normScale.append(dpf.normZScore(i.reshape(-1, 1)).reshape(1, -
1).flatten())

        self.scaleSegmentListT3 = normScale

        self.avgScaleSegT3 =
dpf.averageSegments(segments=self.scaleSegmentListT3)




    def plotSegScaleT1(self):

        plt.figure(figsize=(32, 16), dpi=150)

        plt.plot(np.linspace(start=0, stop=((len(self.avgScaleSegT1)) *
self.fs_Scale), num=(len(self.avgScaleSegT1))),

                 self.avgScaleSegT1[:], "r", linewidth=4)

        for i in range(len(self.scaleSegmentListT1)):

            plt.plot(np.linspace(start=0,
stop=(len(self.scaleSegmentListT1[i])) * self.fs_Scale,

                                 num=(len(self.scaleSegmentListT1[i]))),
self.scaleSegmentListT1[i], "b--")

        plt.grid(axis='x')

        plt.title("Scale segmented and averaged")

        plt.xticks(np.arange(start=0,
stop=((len(self.scaleSegmentListT1[1])) * self.fs_Scale), step=0.1))

        #
plt.savefig("C:\\Users\\simon\\Documents\\Master\\Bilder\\filtered_segmente
d_with_PPG_Scale_back_from_shoes_2_0_with_ppg__scale_03_04_2022_test1.png")
```

```
        plt.show()


    def plotSegScaleCP(self):

        plt.figure(figsize=(32, 16), dpi=150)

        plt.plot(np.linspace(start=0, stop=((len(self.avgScaleSegCP)) *
self.fs_Scale), num=(len(self.avgScaleSegCP))),

                 self.avgScaleSegCP[:], "r", linewidth=4)

        for i in range(len(self.scaleSegmentListCP)):

            plt.plot(np.linspace(start=0,
stop=(len(self.scaleSegmentListCP[i])) * self.fs_Scale,

                                 num=(len(self.scaleSegmentListCP[i]))),
self.scaleSegmentListCP[i], "b--")

        plt.grid(axis='x')

        plt.title("Scale segmented and averaged")

        plt.xticks(np.arange(start=0,
stop=((len(self.scaleSegmentListCP[1])) * self.fs_Scale), step=0.1))

        #
plt.savefig("C:\\Users\\simon\\Documents\\Master\\Bilder\\filtered_segmente
d_with_PPG_Scale_back_from_shoes_2_0_with_ppg__scale_03_04_2022_test1.png")

        plt.show()


    def plotSegScaleT3(self):

        plt.figure(figsize=(32, 16), dpi=150)

        plt.plot(np.linspace(start=0, stop=((len(self.avgScaleSegT3)) *
self.fs_Scale), num=(len(self.avgScaleSegT3))),

                 self.avgScaleSegT3[:], "r", linewidth=4)

        for i in range(len(self.scaleSegmentListT3)):

            plt.plot(np.linspace(start=0,
stop=(len(self.scaleSegmentListT3[i])) * self.fs_Scale,

                                 num=(len(self.scaleSegmentListT3[i]))),
self.scaleSegmentListT3[i], "b--")

        plt.grid(axis='x')

        plt.title("Scale segmented and averaged")

        plt.xticks(np.arange(start=0,
stop=((len(self.scaleSegmentListT3[1])) * self.fs_Scale), step=0.1))

        #
plt.savefig("C:\\Users\\simon\\Documents\\Master\\Bilder\\filtered_segmente
d_with_PPG_Scale_back_from_shoes_2_0_with_ppg__scale_03_04_2022_test1.png")

        plt.show()


    def plotPPGAvgVsScaleAvgT1(self):
```

```
        #x1 = np.linspace(start=0, stop=((len(pressAvg)) *
self.fs_Arduino), num=(len(pressAvg)))


        fig, ax1 = plt.subplots(figsize=(16, 8), dpi=250)

        l1, = ax1.plot(np.linspace(start=0, stop=((len(self.avgScaleSegT1))
* self.fs_Scale), num=(len(self.avgScaleSegT1))),

                 self.avgScaleSegT1[:], "r", linewidth=2)

        ax2 = ax1.twinx()


        l2, = ax2.plot(np.linspace(start=0, stop=(len(self.avgSeg0T1)) *
self.fs_Arduino, num=(len(self.avgSeg0T1))),

                 self.avgSeg0T1, "b", linewidth=2)

        #plt.axvline(x=x1[ppgPeak[0][0]])

        #plt.axvline(x=x1[bcgPeak[0][0]])

        #print("PPG peak:", x1[ppgPeak[0][0]], " and bcg peak: ",
x1[bcgPeak[0][0]])

        #fig.axes[1].get_yaxis().set_visible(False)

        #fig.axes[0].get_yaxis().set_visible(False)

        # plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,

        #                     num=(len(normData[i]))), normData[i], "b--")

        plt.grid(axis='x')

        plt.legend([l1, l2], ["Pressure averaged", "PPG average"],
loc="upper left",

                   fontsize=14)

        #ax1.legend(loc='upper left', fontsize=10)

        ax1.set_xlabel("Time [seconds]", fontsize=16)

        ax1.set_ylabel("Scale normalized", fontsize=16)

        ax2.set_ylabel("Pressure normalized", fontsize=16)

        plt.title("Scale segments averaged vs pressure average - T1")

        plt.tight_layout()

        plt.xticks(np.arange(start=0, stop=((len(self.avgScaleSegT1)) *
self.fs_Scale), step=0.1))

        plt.show()


    def plotPPGAvgVsScaleAvgCP(self):

        #x1 = np.linspace(start=0, stop=((len(pressAvg)) *
self.fs_Arduino), num=(len(pressAvg)))
```

```
        fig, ax1 = plt.subplots(figsize=(16, 8), dpi=250)

        l1, = ax1.plot(np.linspace(start=0, stop=((len(self.avgScaleSegCP))
* self.fs_Scale), num=(len(self.avgScaleSegCP))),

                 self.avgScaleSegCP[:], "r", linewidth=2)

        ax2 = ax1.twinx()


        l2, = ax2.plot(np.linspace(start=0, stop=(len(self.avgSeg0CP)) *
self.fs_Arduino, num=(len(self.avgSeg0CP))),

                 self.avgSeg0CP, "b", linewidth=2)

        #plt.axvline(x=x1[ppgPeak[0][0]])

        #plt.axvline(x=x1[bcgPeak[0][0]])

        #print("PPG peak:", x1[ppgPeak[0][0]], " and bcg peak: ",
x1[bcgPeak[0][0]])

        #fig.axes[1].get_yaxis().set_visible(False)

        #fig.axes[0].get_yaxis().set_visible(False)

        # plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,

        #                     num=(len(normData[i]))), normData[i], "b--")

        plt.grid(axis='x')

        plt.legend([l1, l2], ["Pressure averaged", "PPG average"],
loc="upper left",

                   fontsize=14)

        #ax1.legend(loc='upper left', fontsize=10)

        ax1.set_xlabel("Time [seconds]", fontsize=16)

        ax1.set_ylabel("Scale normalized", fontsize=16)

        ax2.set_ylabel("Pressure normalized", fontsize=16)

        plt.title("Scale segments averaged vs pressure average - T1")

        plt.tight_layout()

        plt.title("Scale segments averaged vs pressure segments averaged -
T2")

        plt.xticks(np.arange(start=0, stop=((len(self.avgScaleSegCP)) *
self.fs_Scale), step=0.05))

        plt.show()


    def plotPPGAvgVsScaleAvgT3(self):

        #x1 = np.linspace(start=0, stop=((len(pressAvg)) *
self.fs_Arduino), num=(len(pressAvg)))


        fig, ax1 = plt.subplots(figsize=(16, 8), dpi=250)
```

```
        l1, = ax1.plot(np.linspace(start=0, stop=((len(self.avgScaleSegT3))
* self.fs_Scale), num=(len(self.avgScaleSegT3))),
                 self.avgScaleSegT3[:], "r", linewidth=2)
        ax2 = ax1.twinx()


        l2, = ax2.plot(np.linspace(start=0, stop=(len(self.avgSeg0T3)) *
self.fs_Arduino, num=(len(self.avgSeg0T3))),
                 self.avgSeg0T3, "b", linewidth=2)
        #plt.axvline(x=x1[ppgPeak[0][0]])
        #plt.axvline(x=x1[bcgPeak[0][0]])
        #print("PPG peak:", x1[ppgPeak[0][0]], " and bcg peak: ",
x1[bcgPeak[0][0]])
        #fig.axes[1].get_yaxis().set_visible(False)
        #fig.axes[0].get_yaxis().set_visible(False)
        # plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,
        #                     num=(len(normData[i]))), normData[i], "b--")
        plt.grid(axis='x')
        plt.legend([l1, l2], ["Pressure averaged", "PPG average"],
loc="upper left",
                   fontsize=14)
        #ax1.legend(loc='upper left', fontsize=10)
        ax1.set_xlabel("Time [seconds]", fontsize=16)
        ax1.set_ylabel("Scale normalized", fontsize=16)
        ax2.set_ylabel("Pressure normalized", fontsize=16)
        plt.title("Scale segments averaged vs pressure average - T1")
        plt.tight_layout()
        plt.title("Scale segments averaged vs pressure segments averaged -
T3")
        plt.xticks(np.arange(start=0, stop=((len(self.avgScaleSegT3)) *
self.fs_Scale), step=0.05))
        plt.show()



    def plotSegPPG(self, T1 = False, CP = False, T3 = False, savePlot =
False, ID = ""):
        #TODO:
        if T1:
            avgData = self.avgSegPPGT1
```

```
            normData = self.normSegPPGT1
        elif CP:
            avgData = self.avgSegPPGCP
            normData = self.normSegPPGCP
        elif T3:
            avgData = self.avgSegPPGT3
            normData = self.normSegPPGT3



        plt.figure(figsize=(32, 16), dpi=150)
        plt.plot(
            np.linspace(start=0, stop=((len(avgData)) * self.fs_Arduino),
num=(len(avgData))),
            avgData[:], "r", linewidth=4)
        for i in range(len(normData)):
            plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,
                                 num=(len(normData[i]))), normData[i], "b--
")
        plt.grid(axis='x')
        plt.title("PPG segmented and averaged")
        plt.xticks(np.arange(start=0, stop=((len(avgData)) *
self.fs_Arduino), step=0.05))
        if savePlot:
            if T1:
                timeStr = "T1"
            elif CP:
                timeStr = "CP"
            elif T3:
                timeStr = "T3"
            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\SegPPG_" + ID + "_" +
timeStr
            plt.savefig(saveName)
        #
plt.savefig("C:\\Users\\simon\\Documents\\Master\\Bilder\\filtered_segmente
d_with_PPG_right_foot_second_furthest_back_interpolated_from_shoes_2_0_with
_ppg__scale_03_04_2022_test1.png")
        plt.show()
```

```
    def plotPeaksVSPPGT1(self, savePlot = False, ID = ""):
        plt.figure(figsize=(32, 16), dpi=150)
plt.plot(self.timeArduino[self.T1TimeArduino[0]:self.T1TimeArduino[1]],
self.PPGFilt[self.T1TimeArduino[0]:self.T1TimeArduino[1]])
        for i in self.T1Peaks:
            plt.axvline(x=self.timeArduino[i])
        #Poplist contains a list of indices of wrong segments. Each segment
has atleast one peak wrongly identified next to it
        #The indice of a wrong segment corresponds to the indice of the
peak in front of the segment.
        #I dont know which peak is wrong....
        lastIndice = 0
        for j in range(len(self.popListT1)):
            try:
                if abs(self.popListT1[j] - self.popListT1[j + 1]) == 1:
plt.axvline(x=self.timeArduino[self.T1Peaks[self.popListT1[j+1]]],
color="r")
                else:
plt.axvline(x=self.timeArduino[self.T1Peaks[self.popListT1[j]]], color="r")
            except IndexError:
plt.axvline(x=self.timeArduino[self.T1Peaks[self.popListT1[j]]], color="r")
        plt.title("PPG and identified PPG peaks")
        if savePlot:
            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\SegPPG" + "_T1_" + ID
            plt.savefig(saveName)
        plt.show()

    def plotPeaksVSPPGCP(self, savePlot = False, ID = ""):
        plt.figure(figsize=(32, 16), dpi=150)
plt.plot(self.timeArduino[self.CPTimeArduino[0]:self.CPTimeArduino[1]],
                 self.PPGFilt[self.CPTimeArduino[0]:self.CPTimeArduino[1]])
        for i in self.CPPeaks:
            plt.axvline(x=self.timeArduino[i])
```

```
        for j in range(len(self.popListCP)):
            try:
                if abs(self.popListCP[j] - self.popListCP[j + 1]) == 1:

plt.axvline(x=self.timeArduino[self.CPPeaks[self.popListCP[j+1]]],
color="r")
                else:

plt.axvline(x=self.timeArduino[self.CPPeaks[self.popListCP[j]]], color="r")
            except IndexError:

plt.axvline(x=self.timeArduino[self.CPPeaks[self.popListCP[j]]], color="r")
        plt.title("PPG and identified PPG peaks")
        if savePlot:
            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\SegPPG" + "_CP_" + ID
            plt.savefig(saveName)
        plt.show()


    def plotPeaksVSPPGT3(self, savePlot = False, ID = ""):
        plt.figure(figsize=(32, 16), dpi=150)

plt.plot(self.timeArduino[self.T3TimeArduino[0]:self.T3TimeArduino[1]],

self.PPGFilt[self.T3TimeArduino[0]:self.T3TimeArduino[1]])
        for i in self.T3Peaks:
            plt.axvline(x=self.timeArduino[i])


        for j in range(len(self.popListT3)):
            try:
                if abs(self.popListT3[j] - self.popListT3[j + 1]) == 1:

plt.axvline(x=self.timeArduino[self.T3Peaks[self.popListT3[j+1]]],
color="r")
                else:

plt.axvline(x=self.timeArduino[self.T3Peaks[self.popListT3[j]]], color="r")
            except IndexError:

plt.axvline(x=self.timeArduino[self.T3Peaks[self.popListT3[j]]], color="r")
```

```
        plt.title("PPG and identified PPG peaks")

        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\SegPPG" + "_CP_" + ID

            plt.savefig(saveName)

        plt.show()


    def plotRightFootT1(self, savePlot = False, ID = ""):

        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(16,16),
dpi=150)

        fig.suptitle('Right foot sensors - T1', fontsize=24)


        ax1.plot(np.linspace(start=0, stop=((len(self.avgSeg0T1)) *
self.fs_Arduino), num=(len(self.avgSeg0T1))), self.avgSeg0T1)

        ax1.set_title("Furthest back", fontsize=16)

        ax2.plot(np.linspace(start=0, stop=((len(self.avgSeg1T1)) *
self.fs_Arduino), num=(len(self.avgSeg1T1))), self.avgSeg1T1)

        ax2.set_title("Second furthest back", fontsize=16)

        ax3.plot(np.linspace(start=0, stop=((len(self.avgSeg3T1)) *
self.fs_Arduino), num=(len(self.avgSeg3T1))), self.avgSeg3T1)

        ax3.set_title("Middle", fontsize=16)

        ax4.plot(np.linspace(start=0, stop=((len(self.avgSeg5T1)) *
self.fs_Arduino), num=(len(self.avgSeg5T1))), self.avgSeg5T1)

        ax4.set_title("Left forward", fontsize=16)

        ax5.plot(np.linspace(start=0, stop=((len(self.avgSeg2T1)) *
self.fs_Arduino), num=(len(self.avgSeg2T1))), self.avgSeg2T1)

        ax5.set_title("Right forward", fontsize=16)

        fig.tight_layout()


        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\RightFootT1_" + ID

            plt.savefig(saveName)


        plt.show()


    def plotLeftFootT1(self, savePlot=False, ID =""):

        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(16,16),
dpi=150)

        fig.suptitle('Left foot sensors - T1', fontsize=24)
```

```
        ax1.plot(np.linspace(start=0, stop=((len(self.avgSeg9T1)) *
self.fs_Arduino), num=(len(self.avgSeg9T1))), self.avgSeg9T1)

        ax1.set_title("Furthest back", fontsize=16)

        ax2.plot(np.linspace(start=0, stop=((len(self.avgSeg8T1)) *
self.fs_Arduino), num=(len(self.avgSeg8T1))), self.avgSeg8T1)

        ax2.set_title("Second furthest back", fontsize=16)

        ax3.plot(np.linspace(start=0, stop=((len(self.avgSeg6T1)) *
self.fs_Arduino), num=(len(self.avgSeg6T1))), self.avgSeg6T1)

        ax3.set_title("Middle", fontsize=16)

        ax4.plot(np.linspace(start=0, stop=((len(self.avgSeg7T1)) *
self.fs_Arduino), num=(len(self.avgSeg7T1))), self.avgSeg7T1)

        ax4.set_title("Left forward", fontsize=16)

        ax5.plot(np.linspace(start=0, stop=((len(self.avgSeg4T1)) *
self.fs_Arduino), num=(len(self.avgSeg4T1))), self.avgSeg4T1)

        ax5.set_title("Right forward", fontsize=16)

        fig.tight_layout()


        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\LeftFootT1_" + ID

            plt.savefig(saveName)


        plt.show()


    def plotRightFootCP(self, savePlot = False, ID=""):

        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(16,16),
dpi=150)

        fig.suptitle('Right foot sensors - Cold pressor', fontsize=24)


        ax1.plot(np.linspace(start=0, stop=((len(self.avgSeg0CP)) *
self.fs_Arduino), num=(len(self.avgSeg0CP))), self.avgSeg0CP)

        ax1.set_title("Furthest back", fontsize=16)

        ax2.plot(np.linspace(start=0, stop=((len(self.avgSeg1CP)) *
self.fs_Arduino), num=(len(self.avgSeg1CP))), self.avgSeg1CP)

        ax2.set_title("Second furthest back", fontsize=16)

        ax3.plot(np.linspace(start=0, stop=((len(self.avgSeg3CP)) *
self.fs_Arduino), num=(len(self.avgSeg3CP))), self.avgSeg3CP)

        ax3.set_title("Middle", fontsize=16)

        ax4.plot(np.linspace(start=0, stop=((len(self.avgSeg5CP)) *
self.fs_Arduino), num=(len(self.avgSeg5CP))), self.avgSeg5CP)
```

```
        ax4.set_title("Left forward", fontsize=16)

        ax5.plot(np.linspace(start=0, stop=((len(self.avgSeg2CP)) *
self.fs_Arduino), num=(len(self.avgSeg2CP))), self.avgSeg2CP)

        ax5.set_title("Right forward", fontsize=16)

        fig.tight_layout()


        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\RightFootCP_" + ID

            plt.savefig(saveName)


        plt.show()


    def plotLeftFootCP(self, savePlot=False, ID = ""):

        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(16,16),
dpi=150)

        fig.suptitle('Left foot sensors - Cold pressor', fontsize=24)


        ax1.plot(np.linspace(start=0, stop=((len(self.avgSeg9CP)) *
self.fs_Arduino), num=(len(self.avgSeg9CP))), self.avgSeg9CP)

        ax1.set_title("Furthest back", fontsize=16)

        ax2.plot(np.linspace(start=0, stop=((len(self.avgSeg8CP)) *
self.fs_Arduino), num=(len(self.avgSeg8CP))), self.avgSeg8CP)

        ax2.set_title("Second furthest back", fontsize=16)

        ax3.plot(np.linspace(start=0, stop=((len(self.avgSeg6CP)) *
self.fs_Arduino), num=(len(self.avgSeg6CP))), self.avgSeg6CP)

        ax3.set_title("Middle", fontsize=16)

        ax4.plot(np.linspace(start=0, stop=((len(self.avgSeg7CP)) *
self.fs_Arduino), num=(len(self.avgSeg7CP))), self.avgSeg7CP)

        ax4.set_title("Left forward", fontsize=16)

        ax5.plot(np.linspace(start=0, stop=((len(self.avgSeg4CP)) *
self.fs_Arduino), num=(len(self.avgSeg4CP))), self.avgSeg4CP)

        ax5.set_title("Right forward", fontsize=16)

        fig.tight_layout()


        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\LeftFootCP_" + ID

            plt.savefig(saveName)
```

```
        plt.show()


    def plotRightFootT3(self, savePlot=False, ID=""):
        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(16,16),
dpi=150)
        fig.suptitle('Right foot sensors - T3', fontsize=24)


        ax1.plot(np.linspace(start=0, stop=((len(self.avgSeg0T3)) *
self.fs_Arduino), num=(len(self.avgSeg0T3))), self.avgSeg0T3)
        ax1.set_title("Furthest back", fontsize=16)
        ax2.plot(np.linspace(start=0, stop=((len(self.avgSeg1T3)) *
self.fs_Arduino), num=(len(self.avgSeg1T3))), self.avgSeg1T3)
        ax2.set_title("Second furthest back", fontsize=16)
        ax3.plot(np.linspace(start=0, stop=((len(self.avgSeg3T3)) *
self.fs_Arduino), num=(len(self.avgSeg3T3))), self.avgSeg3T3)
        ax3.set_title("Middle", fontsize=16)
        ax4.plot(np.linspace(start=0, stop=((len(self.avgSeg5T3)) *
self.fs_Arduino), num=(len(self.avgSeg5T3))), self.avgSeg5T3)
        ax4.set_title("Left forward", fontsize=16)
        ax5.plot(np.linspace(start=0, stop=((len(self.avgSeg2T3)) *
self.fs_Arduino), num=(len(self.avgSeg2T3))), self.avgSeg2T3)
        ax5.set_title("Right forward", fontsize=16)
        fig.tight_layout()


        if savePlot:
            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\RightFootT3_" + ID
            plt.savefig(saveName)


        plt.show()


    def plotLeftFootT3(self, savePlot=False, ID=""):
        fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, figsize=(16,16),
dpi=150)
        fig.suptitle('Left foot sensors', fontsize=24)


        ax1.plot(np.linspace(start=0, stop=((len(self.avgSeg9T3)) *
self.fs_Arduino), num=(len(self.avgSeg9T3))), self.avgSeg9T3)
        ax1.set_title("Furthest back", fontsize=16)
```

```
        ax2.plot(np.linspace(start=0, stop=((len(self.avgSeg8T3)) *
self.fs_Arduino), num=(len(self.avgSeg8T3))), self.avgSeg8T3)

        ax2.set_title("Second furthest back", fontsize=16)

        ax3.plot(np.linspace(start=0, stop=((len(self.avgSeg6T3)) *
self.fs_Arduino), num=(len(self.avgSeg6T3))), self.avgSeg6T3)

        ax3.set_title("Middle", fontsize=16)

        ax4.plot(np.linspace(start=0, stop=((len(self.avgSeg7T3)) *
self.fs_Arduino), num=(len(self.avgSeg7T3))), self.avgSeg7T3)

        ax4.set_title("Left forward", fontsize=16)

        ax5.plot(np.linspace(start=0, stop=((len(self.avgSeg4T3)) *
self.fs_Arduino), num=(len(self.avgSeg4T3))), self.avgSeg4T3)

        ax5.set_title("Right forward", fontsize=16)

        fig.tight_layout()


        if savePlot:

            saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots\\LeftFootT3_" + ID

            plt.savefig(saveName)


        plt.show()


    def plotLeftAndRightFootT1(self): # Not finished.

        fig, axs = plt.sublpots(5, 2, figsize=(16, 32), dpi = 150)

        fig.suptitle('Left and right foot pressure', fontsize=24)

        #Right foot sensors

        axs[0, 0].plot(np.linspace(start=0, stop=((len(self.avgSeg0T1)) *
self.fs_Arduino), num=(len(self.avgSeg0T1))), self.avgSeg0T1)

        axs[0, 0].set_title("Furthest back", fontsize=16)

        axs[1, 0].plot(np.linspace(start=0, stop=((len(self.avgSeg1T3)) *
self.fs_Arduino), num=(len(self.avgSeg1T3))),

                  self.avgSeg1T3)

        axs[1, 0].set_title("Second furthest back", fontsize=16)

        axs[2, 0].plot(np.linspace(start=0, stop=((len(self.avgSeg3T3)) *
self.fs_Arduino), num=(len(self.avgSeg3T3))),

                  self.avgSeg3T3)

        axs[2, 0].set_title("Middle", fontsize=16)

        axs[3, 0].plot(np.linspace(start=0, stop=((len(self.avgSeg5T3)) *
self.fs_Arduino), num=(len(self.avgSeg5T3))),

                  self.avgSeg5T3)

        axs[3, 0].set_title("Left forward", fontsize=16)
```

```
        axs[4, 0].plot(np.linspace(start=0, stop=((len(self.avgSeg2T3)) *
self.fs_Arduino), num=(len(self.avgSeg2T3))),
                  self.avgSeg2T3)
        axs[4, 0].set_title("Right forward", fontsize=16)
        fig.tight_layout()



    def plotPPGavgVsPressavgT1(self, pressSelect, savePlot = False): #Not
finished
        if pressSelect == 0:
            pressAvg = self.avgSeg0T1
        if pressSelect == 1:
            pressAvg = self.avgSeg1T1
        if pressSelect == 2:
            pressAvg = self.avgSeg2T1
        if pressSelect == 3:
            pressAvg = self.avgSeg3T1
        if pressSelect == 4:
            pressAvg = self.avgSeg4T1
        if pressSelect == 5:
            pressAvg = self.avgSeg5T1
        if pressSelect == 6:
            pressAvg = self.avgSeg6T1
        if pressSelect == 7:
            pressAvg = self.avgSeg7T1
        if pressSelect == 8:
            pressAvg = self.avgSeg8T1
        if pressSelect == 9:
            pressAvg = self.avgSeg9T1
        ppgPeak = scipy.signal.find_peaks(self.avgSegPPGT1, height=0.5)#,
distance=1/self.fs_Arduino) #Find the peak within 1 second
        bcgPeak = scipy.signal.find_peaks(pressAvg, height=0.6)
        x1 = np.linspace(start=0, stop=((len(pressAvg)) * self.fs_Arduino),
num=(len(pressAvg)))



        fig, ax1 = plt.subplots(figsize=(16, 8), dpi = 250)
```

```
        l1, = ax1.plot(np.linspace(start=0, stop=((len(pressAvg)) *
self.fs_Arduino), num=(len(pressAvg))),
                 pressAvg[:], "r", linewidth=2)
        ax2 = ax1.twinx()

        l2, = ax2.plot(np.linspace(start=0, stop=(len(self.avgSegPPGT1)) *
self.fs_Arduino, num=(len(self.avgSegPPGT1))),
                       self.avgSegPPGT1, "b", linewidth=2)
        plt.axvline(x=x1[ppgPeak[0][0]])
        plt.axvline(x=x1[bcgPeak[0][1]], color="r")
        fig.axes[1].get_yaxis().set_visible(False)
        fig.axes[0].get_yaxis().set_visible(False)
        ax1.set_ylabel("Pressure normalized", fontsize=18)
        ax2.set_ylabel("PPG normalized", fontsize=18)
        ax1.set_xlabel("Time [seconds]", fontsize=18)
        plt.legend([l1, l2], ["Pressure segments averaged", "PPG segments
averaged"], loc="upper right",
                   fontsize=14)
        print("PPG peak:", x1[ppgPeak[0][1]], " and bcg peak: ",
x1[bcgPeak[0][1]])
        #fig.axes[1].get_yaxis().set_visible(False)
        #fig.axes[0].get_yaxis().set_visible(False)
            #plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,
            #                     num=(len(normData[i]))), normData[i], "b-
-")
        #plt.legend([l1, l2], ["Pressure averaged", "PPG average"],
loc="upper left",
        #           fontsize=10)
        #ax1.legend(loc='upper left', fontsize=10)
        #ax1.set_xlabel("Time [seconds]", fontsize=16)
        plt.grid(axis='x')
        plt.title("Pressure segments averaged vs PPG segments averaged -
T1")
        plt.xticks(np.arange(start=0, stop=((len(self.avgSeg0T1)) *
self.fs_Arduino), step=0.05))
        plt.show()

    def plotPPGavgVsPressavgCP(self, pressSelect, savePlot = False): #Not
finished
```

```
        if pressSelect == 0:
            pressAvg = self.avgSeg0CP
        if pressSelect == 1:
            pressAvg = self.avgSeg1CP
        if pressSelect == 2:
            pressAvg = self.avgSeg2CP
        if pressSelect == 3:
            pressAvg = self.avgSeg3CP
        if pressSelect == 4:
            pressAvg = self.avgSeg4CP
        if pressSelect == 5:
            pressAvg = self.avgSeg5CP
        if pressSelect == 6:
            pressAvg = self.avgSeg6CP
        if pressSelect == 7:
            pressAvg = self.avgSeg7CP
        if pressSelect == 8:
            pressAvg = self.avgSeg8CP
        if pressSelect == 9:
            pressAvg = self.avgSeg9CP


        ppgPeak = scipy.signal.find_peaks(self.avgSegPPGCP, height=0.5)#,
distance=1/self.fs_Arduino) #Find the peak within 1 second
        bcgPeak = scipy.signal.find_peaks(pressAvg, height=0.6)
        x1 = np.linspace(start=0, stop=((len(pressAvg)) * self.fs_Arduino),
num=(len(pressAvg)))



        fig, ax1 = plt.subplots(figsize=(16, 8), dpi = 250)
        l1, = ax1.plot(np.linspace(start=0, stop=((len(pressAvg)) *
self.fs_Arduino), num=(len(pressAvg))),
                  pressAvg[:], "r", linewidth=2)
        ax2 = ax1.twinx()


        l2, = ax2.plot(np.linspace(start=0, stop=(len(self.avgSegPPGCP)) *
self.fs_Arduino, num=(len(self.avgSegPPGCP))),
                       self.avgSegPPGCP, "b", linewidth=2)
        fig.axes[1].get_yaxis().set_visible(False)
```

```
        fig.axes[0].get_yaxis().set_visible(False)



        ax1.set_xlabel("Time [seconds]", fontsize=18)

        plt.legend([l1, l2], ["Pressure segments averaged", "PPG segments
averaged"], loc="upper right",

                   fontsize=14)

        plt.axvline(x=x1[ppgPeak[0][0]])

        plt.axvline(x=x1[bcgPeak[0][1]], color = "r")

        ax1.set_ylabel("Pressure normalized", fontsize=18)

        ax2.set_ylabel("PPG normalized", fontsize=18)

        print("PPG peak:", x1[ppgPeak[0][0]], " and bcg peak: ",
x1[bcgPeak[0][1]])



        #fig.axes[1].get_yaxis().set_visible(False)

            #plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,

            #                     num=(len(normData[i]))), normData[i], "b-
-")

        plt.grid(axis='x')

        plt.title("Pressure segments averaged vs PPG segments averaged -
T2")

        plt.xticks(np.arange(start=0, stop=((len(self.avgSeg0CP)) *
self.fs_Arduino), step=0.05))

        plt.show()


    def plotPPGavgVsPressavgT3(self, pressSelect, savePlot = False): #Not
finished

        if pressSelect == 0:

            pressAvg = self.avgSeg0T3

        if pressSelect == 1:

            pressAvg = self.avgSeg1T3

        if pressSelect == 2:

            pressAvg = self.avgSeg2T3

        if pressSelect == 3:

            pressAvg = self.avgSeg3T3

        if pressSelect == 4:

            pressAvg = self.avgSeg4T3

        if pressSelect == 5:
```

```
            pressAvg = self.avgSeg5T3
        if pressSelect == 6:
            pressAvg = self.avgSeg6T3
        if pressSelect == 7:
            pressAvg = self.avgSeg7T3
        if pressSelect == 8:
            pressAvg = self.avgSeg8T3
        if pressSelect == 9:
            pressAvg = self.avgSeg9T3


        ppgPeak = scipy.signal.find_peaks(self.avgSegPPGT3, height=0.5)#,
distance=1/self.fs_Arduino) #Find the peak within 1 second
        bcgPeak = scipy.signal.find_peaks(pressAvg, height=0.6)
        x1 = np.linspace(start=0, stop=((len(pressAvg)) * self.fs_Arduino),
num=(len(pressAvg)))


        fig, ax1 = plt.subplots(figsize=(16, 8), dpi = 250)
        l1, = ax1.plot(np.linspace(start=0, stop=((len(pressAvg)) *
self.fs_Arduino), num=(len(pressAvg))),
                  pressAvg[:], "r", linewidth=2)
        ax2 = ax1.twinx()
        l2, = ax2.plot(np.linspace(start=0, stop=(len(self.avgSegPPGT3)) *
self.fs_Arduino, num=(len(self.avgSegPPGT3))),
                       self.avgSegPPGT3, "b", linewidth=2)
        fig.axes[1].get_yaxis().set_visible(False)
        fig.axes[0].get_yaxis().set_visible(False)
        plt.axvline(x=x1[ppgPeak[0][0]])
        plt.axvline(x=x1[bcgPeak[0][1]], color = "r")
        ax1.set_ylabel("Pressure normalized", fontsize=18)
        ax2.set_ylabel("PPG normalized", fontsize=18)
        ax1.set_xlabel("Time [seconds]", fontsize=18)
        plt.legend([l1, l2], ["Pressure segments averaged", "PPG segments
averaged"], loc="upper right",
                   fontsize=14)
        print("PPG peak:", x1[ppgPeak[0][1]], " and bcg peak: ",
x1[bcgPeak[0][1]])
        #fig.axes[1].get_yaxis().set_visible(False)
            #plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,
```

```
            #                       num=(len(normData[i]))), normData[i], "b-
--")

        plt.grid(axis='x')

        plt.title("Pressure segments averaged vs PPG segments averaged -
T3")

        plt.xticks(np.arange(start=0, stop=((len(self.avgSeg0T3)) *
self.fs_Arduino), step=0.05))

        plt.show()


    def plotPressureWithPPG(self, ID, T1 = False, CP = False, T3 = False,
savePlot = False):

        if T1:

            pressSegs = [self.normSeg0T1, self.normSeg1T1, self.normSeg2T1,
self.normSeg3T1, self.normSeg4T1, self.normSeg5T1,

                         self.normSeg6T1, self.normSeg7T1, self.normSeg8T1,
self.normSeg9T1]

            pressAvgs = [self.avgSeg0T1, self.avgSeg1T1, self.avgSeg2T1,
self.avgSeg3T1, self.avgSeg4T1, self.avgSeg5T1,

                         self.avgSeg6T1, self.avgSeg7T1, self.avgSeg8T1,
self.avgSeg9T1]

            ppgAvg = self.avgSegPPGT1

            timeStr = "T1"

        elif CP:

            pressSegs = [self.normSeg0CP, self.normSeg1CP, self.normSeg2CP,
self.normSeg3CP, self.normSeg4CP, self.normSeg5CP,

                         self.normSeg6CP, self.normSeg7CP, self.normSeg8CP,
self.normSeg9CP]

            pressAvgs = [self.avgSeg0CP, self.avgSeg1CP, self.avgSeg2CP,
self.avgSeg3CP, self.avgSeg4CP, self.avgSeg5CP,

                         self.avgSeg6CP, self.avgSeg7CP, self.avgSeg8CP,
self.avgSeg9CP]

            ppgAvg = self.avgSegPPGCP

            timeStr = "T2"

        elif T3:

            pressSegs = [self.normSeg0T3, self.normSeg1T3, self.normSeg2T3,
self.normSeg3T3, self.normSeg4T3, self.normSeg5T3,

                         self.normSeg6T3, self.normSeg7T3, self.normSeg8T3,
self.normSeg9T3]

            pressAvgs = [self.avgSeg0T3, self.avgSeg1T3, self.avgSeg2T3,
self.avgSeg3T3, self.avgSeg4T3, self.avgSeg5T3,

                         self.avgSeg6T3, self.avgSeg7T3, self.avgSeg8T3,
self.avgSeg9T3]

            ppgAvg = self.avgSegPPGT3
```

```
            timeStr = "T3"

        posList = ["right 1", "right 2", "right 5", "right 3", "left 5",
"right 4", "left 3", "left 4", "left 2", "left 1"]

        #self.Pressure0 = dataArduino.Pressure0.to_numpy() #Right foot
furthest back

        #self.Pressure1 = dataArduino.Pressure1.to_numpy() #Right foot
second furthest back

        #self.Pressure2 = dataArduino.Pressure2.to_numpy() #Right foot
forward to the right

        #self.Pressure3 = dataArduino.Pressure3.to_numpy() #Right foot
middle

        #self.Pressure4 = dataArduino.Pressure4.to_numpy() #Left foot
forward right

        #self.Pressure5 = dataArduino.Pressure5.to_numpy() #Right foot
forward left

        #self.Pressure6 = dataArduino.Pressure6.to_numpy() #Left foot
middle

        #self.Pressure7 = dataArduino.Pressure7.to_numpy() #Left foot
forward left

        #self.Pressure8 = dataArduino.Pressure8.to_numpy() #Left foot
second furthest back

        #self.Pressure9 = dataArduino.Pressure9.to_numpy() #Left foot
furthest back

        #Testing something here

        for i in range(len(pressSegs)):

            fig, (ax1, ax2) = plt.subplots(2, figsize=(12, 8), dpi=250)


            l1, = ax1.plot(np.linspace(start=0, stop=((len(ppgAvg)) *
self.fs_Arduino), num=(len(ppgAvg))),

                 ppgAvg[:], "r", linewidth=4, label = "PPG Average")

            ax1.set_title("PPG average based on segmentation", fontsize=24)


            l2, = ax2.plot(np.linspace(start=0, stop=((len(pressAvgs[i])) *
self.fs_Arduino), num=(len(pressAvgs[i]))),

                 pressAvgs[i][:], "r", linewidth=4)


        # plt.plot(

        #    np.linspace(start=0, stop=((len(avgData)) * self.fs_Arduino),
num=(len(avgData))),

        #    avgData[:], "r", linewidth=4)

            axisList = []
```

```
            #ax3 = ax2.twinx() # If i dont twin i force everything onto the
same axes

            for j in range(len(pressSegs[i])):

                #axisList.append(ax1.twinx())

                #axisList[j].plot(np.linspace(start=0,
stop=(len(pressSegs[i][j])) * self.fs_Arduino, num=(len(pressSegs[i][j]))),

                #                pressSegs[i][j], "b--")

                #fig.axes[j].get_yaxis().set_visible(False)

                l3, = ax2.plot(np.linspace(start=0,
stop=(len(pressSegs[i][j])) * self.fs_Arduino, num=(len(pressSegs[i][j]))),

                           pressSegs[i][j], "b--", linewidth=1)

            # plt.plot(np.linspace(start=0, stop=(len(normData[i])) *
self.fs_Arduino,

            #                     num=(len(normData[i]))), normData[i], "b-
-")

            plt.grid(axis='x')

            ax2.set_title("Pressure segmented and averaged - " + "position
" + posList[i], fontsize=24)

            plt.xticks(np.arange(start=0, stop=((len(pressAvgs[i])) *
self.fs_Arduino), step=0.1))

            plt.legend([l2, l3], ["Average pressure normalized", "Pressure
segments normalized"], loc="upper right", fontsize=14)

            ax1.legend(loc='upper right', fontsize = 14)

            ax1.set_xlabel("Time [seconds]", fontsize = 18)

            ax2.set_xlabel("Time [seconds]", fontsize = 18)

            fig.tight_layout()


            if T1:

                tp = "T1"

            elif CP:

                tp = "CP"

            elif T3:

                tp = "T3"

            if savePlot:

                saveName =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\Plots2\\PressAndPPG_Press" +
str(i) + "_" + tp + "_" + ID

                plt.savefig(saveName)


            #plt.show()
```

```
#Testing functions:

#timeStampsT1_2170 = ["04:05:2022:14:26:00:0", "04:05:2022:14:26:15:0"]

#timeStampsCP_2170 = ["04:05:2022:14:27:40:0", "04:05:2022:14:27:55:0"]

#timeStampsT3_2170 = ["04:05:2022:14:33:00:0", "04:05:2022:14:33:15:0"]

#dataLabChartPath_2170 =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\2170\\2170_Labchart_Raw_Text.
txt"

#dataArduinoPath_2170 =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\2170\\2170_Arduino_Raw_4.csv"

#interestingDataList = ["Time", "HC_Systolic", "Active_Cuff",
"Finger_Pressure_HC"]

#nameList = ["Time", "Date", "Finger_Pressure", "Finger_Pressure_HC",
"HCU_Pressure", "Systolic", "HC_Systolic",

#            "Mean_Arterial", "HC_Mean_Arterial", "Diastolic",
"HC_Diastolic", "HR", "Interbeat_Interval", "Active_Cuff",

#           "Cuff_Countdown", "Autocal_Quality", "Autocal_Countdown",
"Comments"]

#nameListArduino = ["timestamp", "Time", "PPG", "Scale", "Pressure0",
"Pressure1", "Pressure2", "Pressure3", "Pressure4"

#                   , "Pressure5", "Pressure6", "Pressure7", "Pressure8",
"Pressure9"]


#Test2170 = Measurement(timeStampsT1_2170, timeStampsCP_2170,
timeStampsT3_2170, dataArduinoPath_2170, dataLabChartPath_2170,
nameListArduino, nameList)

#Test2170.loadChartData()

#Test2170.SetTimeIndices()

#Test2170.averageFingerPressure()

#Test2170.plotAvgFingPress()

#Test2170.findPPGPeaks(threshold=500, timeBetween=0.5)

#Test2170.organizePeaks()

#Test2170.filterPressure(butterLevel=12, passBand=[3,10])

#Test2170.segmentPressure(ddPPG=False)

#Test2170.segmentPPG(ddPPG=False)

#Test2170.plotSegPress(pressSelect=1, T1 = True)

#Test2170.plotSegPPG(T1 = True)

#Test2170.plotPeaksVSPPGT1()

#Test2170.plotRightFoot()

#print(Test2170.hz_Arduino)


import numpy as np
```

```
from matplotlib import pyplot as plt

import pandas as pd

import scipy.interpolate as interpolate

from scipy.signal import filtfilt, butter, detrend, savgol_filter

import pywt

from sklearn import preprocessing

import scipy

import data_processing_functions as dpf

import Processing_Test_Results_V3 as ptr

import datetime

import seaborn as sns


interestingDataList = ["Time", "HC_Systolic", "Active_Cuff",
"Finger_Pressure_HC"]

nameList = ["Time", "Date", "Finger_Pressure", "Finger_Pressure_HC",
"HCU_Pressure", "Systolic", "HC_Systolic",

            "Mean_Arterial", "HC_Mean_Arterial", "Diastolic",
"HC_Diastolic", "HR", "Interbeat_Interval", "Active_Cuff",

           "Cuff_Countdown", "Autocal_Quality", "Autocal_Countdown",
"Comments"]

nameListArduino = ["timestamp", "Time", "PPG", "Scale", "Pressure0",
"Pressure1", "Pressure2", "Pressure3", "Pressure4"

                   , "Pressure5", "Pressure6", "Pressure7", "Pressure8",
"Pressure9"]


dataLabChartPath_2832 =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\2832\\2832_Labchart_Raw_Text.
txt"

dataArduinoPath_2832 =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\2832\\2832_Arduino_Raw_2.csv"

timeStampsT1_2832 = ["04:05:2022:13:23:30:0", "04:05:2022:13:24:05:0"]

timeStampsCP_2832 = ["04:05:2022:13:25:45:0", "04:05:2022:13:26:20:0"]

timeStampsT3_2832 = ["04:05:2022:13:30:00:0", "04:05:2022:13:30:35:0"]


Test2832 = ptr.Measurement(timeStampsT1_2832, timeStampsCP_2832,
timeStampsT3_2832, dataArduinoPath_2832,

                           dataLabChartPath_2832, nameListArduino,
nameList)


Test2832.loadChartData()

Test2832.SetTimeIndices()
```

```
Test2832.averageFingerPressure()

Test2832.findPPGPeaks(threshold=500, timeBetween=0.5)

Test2832.organizePeaks()

Test2832.filterPressure(butterLevel=12, passBand=[3,10])

Test2832.segmentPressure(ddPPG=False)

Test2832.segmentPPG(ddPPG=False)


Test2832.plotAvgFingPress()


#Test2832.plotSegPressV2(pressSelect=3, T1 = True, savePlot=True,
ID="T1_2832")

#Test2832.plotAllPressureSegAndAvg(ID="_T1_2832", T1 = True, CP = False, T3
= False, savePlot = True)

Test2832.plotPressureWithPPG(ID="2832", T1 = True, CP = False, T3 = False,
savePlot = True)

Test2832.plotSegPPG(T1 = True)

Test2832.plotPeaksVSPPGT1()


#Test2832.plotSegPressV2(pressSelect=1, CP = True, savePlot=True,
ID="CP_2832")

#Test2832.plotAllPressureSegAndAvg(ID="_CP_2832", T1 = False, CP = True, T3
= False, savePlot = True)

Test2832.plotPressureWithPPG(ID="2832", T1 = False, CP = True, T3 = False,
savePlot = True)

Test2832.plotSegPPG(CP = True)

Test2832.plotPeaksVSPPGCP()


#Test2832.plotSegPressV2(pressSelect=1, T3 = True, savePlot=True,
ID="T3_2832")

#Test2832.plotAllPressureSegAndAvg(ID="_T3_2832", T1 = False, CP = False,
T3 = True, savePlot = True)

Test2832.plotPressureWithPPG(ID="2832", T1 = False, CP = False, T3 = True,
savePlot = True)

Test2832.plotSegPPG(T3 = True)

Test2832.plotPeaksVSPPGT3()


#Test2832.plotRightFootT1(savePlot=True, ID="2832")

#Test2832.plotLeftFootT1(savePlot=True, ID="2832")


Test2832.plotPPGavgVsPressavgT1(pressSelect = 8 , savePlot = False)
```

```
Test2832.plotPPGavgVsPressavgCP(pressSelect = 8 , savePlot = False)

Test2832.plotPPGavgVsPressavgT3(pressSelect = 8 , savePlot = False)


#Computing the average pressure for each section

T1Avg =
np.average(Test2832.HCSystolicFiltered[Test2832.T1TimeChart[0]:Test2832.T1T
imeChart[1]])

T2Avg =
np.average(Test2832.HCSystolicFiltered[Test2832.CPTimeChart[0]:Test2832.CPT
imeChart[1]])

T3Avg =
np.average(Test2832.HCSystolicFiltered[Test2832.T3TimeChart[0]:Test2832.T3T
imeChart[1]])

print("T1 avgpress: ", T1Avg, "\n", "T2 avgpress: ", T2Avg, "\n", "T3
avgpress: ", T3Avg  )


Test2832.segmentScale()

Test2832.plotSegScaleT1()

Test2832.plotSegScaleCP()

Test2832.plotSegScaleT3()

Test2832.plotPPGAvgVsScaleAvgT1()

Test2832.plotPPGAvgVsScaleAvgCP()

Test2832.plotPPGAvgVsScaleAvgT3()


#Plotting scale vs PPG

pressAvg = Test2832.avgSeg8T1

fig, ax1 = plt.subplots(figsize=(16, 8), dpi = 250)

l1, =ax1.plot(np.linspace(start=0, stop=((len(pressAvg)) *
Test2832.fs_Arduino), num=(len(pressAvg))),

         pressAvg[:], "r", linewidth=2)

ax2 = ax1.twinx()


l2, = ax2.plot(np.linspace(start=0, stop=(len(Test2832.avgScaleSegT1)) *
Test2832.fs_Scale, num=(len(Test2832.avgScaleSegT1))),

                         Test2832.avgScaleSegT1, "b", linewidth=2)

ax3 = ax1.twinx()

l3, = ax2.plot(np.linspace(start=0, stop=(len(Test2832.avgSegPPGT1)) *
Test2832.fs_Arduino, num=(len(Test2832.avgSegPPGT1))),

                         Test2832.avgSegPPGT1, "g", linewidth=2)

plt.grid(axis='x')
```

```
plt.legend([l1, l2, l3],["Trykksensor", "Vekt", "PPG"], loc="upper left",
fontsize=16)

plt.show()


## Participant 4116


dataLabChartPath_4116 =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\4116\\4116_Labchart_Raw_Text.
txt"

dataArduinoPath_4116 =
"C:\\Users\\simon\\Documents\\Master\\Forsøk\\4116\\4116_Arduino_Raw_2.csv"

timeStampsT1_4116 = ["04:05:2022:10:20:00:0", "04:05:2022:10:20:25:0"]

timeStampsCP_4116 = ["04:05:2022:10:22:00:0", "04:05:2022:10:22:25:0"]

timeStampsT3_4116 = ["04:05:2022:10:26:15:0", "04:05:2022:10:26:40:0"]


Test4116 = ptr.Measurement(timeStampsT1_4116, timeStampsCP_4116,
timeStampsT3_4116, dataArduinoPath_4116,

                           dataLabChartPath_4116, nameListArduino,
nameList)


Test4116.loadChartData()

Test4116.SetTimeIndices()

Test4116.averageFingerPressure()

Test4116.findPPGPeaks(threshold=500, timeBetween=0.5)

Test4116.organizePeaks()

Test4116.filterPressure(butterLevel=12, passBand=[3,10])

Test4116.segmentPressure(ddPPG=False)

Test4116.segmentPPG(ddPPG=False)


Test4116.plotAvgFingPress()


#Test4116.plotSegPressV2(pressSelect=3, T1 = True, savePlot=True,
ID="T1_4116")

#Test4116.plotAllPressureSegAndAvg(ID="_T1_4116", T1 = True, CP = False, T3
= False, savePlot = True)

Test4116.plotPressureWithPPG(ID="4116", T1 = True, CP = False, T3 = False,
savePlot = True)

Test4116.plotSegPPG(T1 = True)

Test4116.plotPeaksVSPPGT1()
```

```
#Test4116.plotSegPressV2(pressSelect=1, CP = True, savePlot=True,
ID="CP_4116")

#Test4116.plotAllPressureSegAndAvg(ID="_CP_4116", T1 = False, CP = True, T3
= False, savePlot = True)

Test4116.plotPressureWithPPG(ID="4116", T1 = False, CP = True, T3 = False,
savePlot = True)

Test4116.plotSegPPG(CP = True)

Test4116.plotPeaksVSPPGCP()


#Test4116.plotSegPressV2(pressSelect=1, T3 = True, savePlot=True,
ID="T3_4116")

#Test4116.plotAllPressureSegAndAvg(ID="_T3_4116", T1 = False, CP = False,
T3 = True, savePlot = True)

Test4116.plotPressureWithPPG(ID="4116", T1 = False, CP = False, T3 = True,
savePlot = True)

Test4116.plotSegPPG(T3 = True)

Test4116.plotPeaksVSPPGT3()


#Test4116.plotRightFootT1(savePlot=True, ID="4116")

#Test4116.plotLeftFootT1(savePlot=True, ID="4116")


#the best and consistent for T1, T2, and T3 was R2 probably? L1 and L2 is
also a good choice

Test4116.plotPPGavgVsPressavgT1(pressSelect = 0 , savePlot = False)

Test4116.plotPPGavgVsPressavgCP(pressSelect = 0 , savePlot = False)

Test4116.plotPPGavgVsPressavgT3(pressSelect = 0 , savePlot = False)




#Computing the average pressure for each section

T1Avg =
np.average(Test4116.HCSystolicFiltered[Test4116.T1TimeChart[0]:Test4116.T1T
imeChart[1]])

T2Avg =
np.average(Test4116.HCSystolicFiltered[Test4116.CPTimeChart[0]:Test4116.CPT
imeChart[1]])

T3Avg =
np.average(Test4116.HCSystolicFiltered[Test4116.T3TimeChart[0]:Test4116.T3T
imeChart[1]])

print("T1 avgpress: ", T1Avg, "\n", "T2 avgpress: ", T2Avg, "\n", "T3
avgpress: ", T3Avg  )


Test4116.segmentScale()
```

```
Test4116.plotSegScaleT1()
Test4116.plotSegScaleCP()
Test4116.plotSegScaleT3()
Test4116.plotPPGAvgVsScaleAvgT1()
Test4116.plotPPGAvgVsScaleAvgCP()
Test4116.plotPPGAvgVsScaleAvgT3()



# Other stuff

strList = ["T1", "T2", "T3"]

pttT1_2832 = 0.437 - 0.498
pttT2_2832 = 0.403 - 0.498
pttT3_2832 = 0.504 - 0.498
ptt_2832 = [pttT1_2832, pttT2_2832, pttT3_2832]
HC_2832 = [149, 166, 134]
sns.set()
pttT1_4116 = 0.481 - 0.297
pttT2_4116 = 0.456 - 0.297
pttT3_4116 = 0.487 - 0.297
ptt_4116 = [pttT1_4116, pttT2_4116, pttT3_4116]
HC_4116 = [100, 151, 101]
fig, ax1 = plt.subplots(figsize= (4, 3), dpi=250)
l1, = ax1.plot(strList, ptt_2832, "--rx")
ax2 = ax1.twinx()
l2, = ax2.plot(strList, HC_2832, "--bx")

ax1.set_ylabel("Pulse Travel Time", fontsize=10)
ax2.set_ylabel("Blood pressure", fontsize=10)
ax1.set_xlabel("Time [seconds]", fontsize=10)
plt.legend([l1, l2],["Pulse travel time", "Systolic blood pressure"],
loc="upper left", fontsize=5)
plt.show()

fig, ax1 = plt.subplots(figsize= (4, 3), dpi=250)
```

```
l1, = ax1.plot(strList, ptt_4116, "--rx")

ax2 = ax1.twinx()

l2, = ax2.plot(strList, HC_4116, "--bx")


ax1.set_ylabel("Pulse Travel Time", fontsize=10)

ax2.set_ylabel("Blood pressure", fontsize=10)

ax1.set_xlabel("Time [seconds]", fontsize=10)

plt.legend([l1, l2],["Pulse travel time", "Systolic blood pressure"],
loc="upper left", fontsize=5)

plt.show()




dataArduinoPath =
"C:\\Users\\simon\\Documents\\Master\\water_filled_bag_test1_01_03_2022.csv
"

dataArduino = pd.read_csv(dataArduinoPath, sep=",")

dataArduino


Time = (dataArduino["Time"].to_numpy())/10**3

Pressure = dataArduino["Pressure"].to_numpy()

fs = (Time[-1] - Time[0])/len(Time)

hz = 1/fs

xlim = [1000, 3000]


plt.figure(figsize=(32, 16), dpi=150)

plt.xticks(np.arange(Time[xlim[0]], Time[xlim[1]], step=1))

plt.grid(axis='x')

plt.plot(Time[xlim[0]:xlim[1]], Pressure[xlim[0]:xlim[1]])

plt.show()


plt.figure(figsize=(32, 16), dpi=150)

plt.xlim([1, 30])

plt.psd(Pressure, Fs=hz)

plt.show()


sos = butter(N=12, Wn=[1, 10], btype='bandpass', fs=hz, output='sos')
```

```
PressFilt = scipy.signal.sosfilt(sos, Pressure)


plt.figure(figsize=(32, 16), dpi=150)
plt.xticks(np.arange(Time[xlim[0]], Time[xlim[1]], step=1))
plt.grid(axis='x')
plt.plot(Time[xlim[0]:xlim[1]], PressFilt[xlim[0]:xlim[1]])
plt.show()


#Initial placement test plots
#3bmp388_in_shoe_08_03_2022_test2
dataArduinoPath1 =
"C:\\Users\\simon\\Documents\\Master\\3bmp388_in_shoe_08_03_2022_test2.csv"
dataArduino1 = pd.read_csv(dataArduinoPath1, sep=",")
dataArduinoPath2 =
"C:\\Users\\simon\\Documents\\Master\\3bmp388_in_shoe_08_03_2022_test3.csv"
dataArduino2 = pd.read_csv(dataArduinoPath2, sep=",")
#Dataarduino1:
#Press0 = Pos 2
#Press1 = Pos 4
#Press2 = Pos 5
#Dataarduino 2:
#Press0 = Pos 1
#Press1 = Pos 3
#Press2 = Pos 6
pos1 = dataArduino2["Pressure0"].to_numpy()
pos2 = dataArduino1["Pressure0"].to_numpy()
pos3 = dataArduino2["Pressure1"].to_numpy()
pos4 = dataArduino1["Pressure1"].to_numpy()
pos5 = dataArduino1["Pressure2"].to_numpy()
pos6 = dataArduino2["Pressure2"].to_numpy()


time1 = dataArduino1["Time"].to_numpy()/10**6
time2 = dataArduino2["Time"].to_numpy()/10**6


pos1 = pos1[1::2]
pos2 = pos2[1::2]
pos3 = pos3[1::2]
```

```
pos4 = pos4[1::2]
pos5 = pos5[1::2]
pos6 = pos6[1::2]
time1 = time1[1::2]
time2 = time2[1::2]

hz1 = 1/((time1[-1]-time1[0])/len(time1))
hz2 = 1/((time2[-1]-time2[0])/len(time2))

sos = butter(N=12, Wn=[2, 10], btype='bandpass', fs=hz1, output='sos')
PosFilt2 = scipy.signal.sosfilt(sos, pos2)
sos = butter(N=12, Wn=[2, 10], btype='bandpass', fs=hz1, output='sos')
PosFilt4 = scipy.signal.sosfilt(sos, pos4)
sos = butter(N=12, Wn=[2, 10], btype='bandpass', fs=hz1, output='sos')
PosFilt5 = scipy.signal.sosfilt(sos, pos5)

sos = butter(N=12, Wn=[2, 10], btype='bandpass', fs=hz2, output='sos')
PosFilt1 = scipy.signal.sosfilt(sos, pos1)
sos = butter(N=12, Wn=[2, 10], btype='bandpass', fs=hz2, output='sos')
PosFilt3 = scipy.signal.sosfilt(sos, pos3)
sos = butter(N=12, Wn=[2, 10], btype='bandpass', fs=hz2, output='sos')
PosFilt6 = scipy.signal.sosfilt(sos, pos6)
len(PosFilt6)
hz1

plt.figure(figsize=(8,4), dpi=250)
plt.plot(time2[1000:1100], pos1[1000:1100])
plt.show()

plt.figure(figsize=(8,4), dpi=250)
plt.plot(time2[4000:7000], PosFilt1[4000:7000], linewidth=1)
plt.title("Filtered pressure in placement 1", fontsize=16)
plt.xlabel("Time [seconds]", fontsize=16)
plt.grid(axis='x')
plt.xticks(np.arange(time2[4000], time2[7000], step=1), fontsize=8)
plt.tight_layout()
```

```
plt.show()

plt.figure(figsize=(10,6), dpi=250)
plt.plot(time1[4000:7000], PosFilt2[4000:7000], linewidth=1)
plt.title("Filtered pressure in placement 2", fontsize=16)
plt.xlabel("Time [seconds]", fontsize=16)
plt.xticks(np.arange(time1[4000], time1[7000], step=1), fontsize=8)
plt.grid(axis='x')
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,4), dpi=250)
plt.plot(time2[4000:7000], PosFilt3[4000:7000], linewidth=1)
plt.title("Filtered pressure in placement 3", fontsize=16)
plt.xlabel("Time [seconds]", fontsize=16)
plt.xticks(np.arange(time2[4000], time2[7000], step=1), fontsize=8)
plt.grid(axis='x')
plt.tight_layout()
plt.show()

plt.figure(figsize=(8,4), dpi=250)
plt.plot(time1[4000:7000], PosFilt4[4000:7000], linewidth=1)
plt.title("Filtered pressure in placement 4", fontsize=16)
plt.xlabel("Time [seconds]", fontsize=16)
plt.xticks(np.arange(time1[4000], time1[7000], step=1), fontsize=8)
plt.tight_layout()
plt.grid(axis='x')
plt.show()

plt.figure(figsize=(8,4), dpi=250)
plt.plot(time1[4000:7000], PosFilt5[4000:7000], linewidth=1)
plt.title("Filtered pressure in placement 5", fontsize=16)
plt.xlabel("Time [seconds]", fontsize=16)
plt.xticks(np.arange(time1[4000], time1[7000], step=1), fontsize=8)
plt.grid(axis='x')
plt.tight_layout()
```

```
plt.show()

plt.figure(figsize=(8,4), dpi=250)
plt.plot(time2[4000:7000], PosFilt6[4000:7000], linewidth=1)
plt.title("Filtered pressure in placement 6", fontsize=16)
plt.xlabel("Time [seconds]", fontsize=16)
plt.xticks()
plt.xticks(np.arange(time2[4000], time2[7000], step=1), fontsize=8)
plt.grid(axis='x')
plt.tight_layout()
plt.show()

plt.figure(figsize=(32, 16), dpi=250)
plt.xlim([1, 30])
plt.psd(PosFilt1, Fs=hz2, linewidth = 1)
plt.show()

plt.figure(figsize=(8, 4), dpi=250)
plt.xlim([1, 30])
plt.psd(PosFilt2, Fs=hz1, linewidth = 1)
plt.show()

plt.figure(figsize=(8, 4), dpi=250)
plt.xlim([1, 30])
plt.psd(PosFilt3, Fs=hz2, linewidth = 1)
plt.show()

plt.figure(figsize=(8, 4), dpi=250)
plt.xlim([1, 30])
plt.psd(PosFilt4, Fs=hz1, linewidth = 1)
plt.show()

plt.figure(figsize=(8, 4), dpi=250)
plt.xlim([1, 30])
plt.psd(PosFilt5, Fs=hz1, linewidth = 1)
plt.show()
```

```
plt.figure(figsize=(8, 32), dpi=250)
plt.xlim([1, 30])
#plt.ylim([0,80])
plt.psd(PosFilt6, Fs=hz2, linewidth = 1)
plt.show()
```

Appendix F

Arduino code for sole 2.0 prototype

```
#include <Wire.h>

#include "HX711.h"


// OBS: I do not have any protection if the slave sends less data than
expected. - fixed ish


#define TCAADDR 0x70

int PPGPin = A0;

#define LOADCELL_DOUT_PIN A1

#define LOADCELL_SCK_PIN A2


const uint32_t BMP_ADDR1 = 0x76;

const uint32_t BMP_ADDR2 = 0x77;


uint32_t pressure;

// int clockFrequency = 400000;

int clockFrequency = 3400000;


//------------- Setting up functions i need ------------------------------
----

//For selecting which multiplexer adress is used

void tcaselect(uint8_t i)

{

  if (i > 7)

    return;


  Wire.beginTransmission(TCAADDR);

  Wire.write(1 << i);

  Wire.endTransmission();

}

//Borrowed from DFrobot library

void I2C_WriteOneByte(uint8_t DevAddr, uint8_t RegAddr, uint8_t value){
```

```
  Wire.beginTransmission(DevAddr);
  Wire.write(RegAddr);
  Wire.write(value);
  Wire.endTransmission();
}
//Borrowed from DFrobot library
int8_t readMultipleBytes(uint8_t devId ,uint8_t regAddr, uint8_t *dataVec,
int dataLen){
  int i = 0;

  Wire.beginTransmission(devId);
  Wire.write(regAddr);
  Wire.endTransmission();

  Wire.requestFrom(devId, dataLen);
  while (Wire.available()){
    dataVec[i++] = Wire.read();
  }
  return 0;
}

uint8_t I2C_ReadOneByte(uint8_t DevAddr, uint8_t RegAddr)
{
  uint8_t value;

  Wire.beginTransmission(DevAddr);
  Wire.write((byte)RegAddr);
  Wire.endTransmission();

  Wire.requestFrom(DevAddr, (byte)1);
  value = Wire.read();

  return value;
}

struct COMPCOEFFS
{
```

```
  double PAR_T1;
  double PAR_T2;
  double PAR_T3;
  double PAR_P1;
  double PAR_P2;
  double PAR_P3;
  double PAR_P4;
  double PAR_P5;
  double PAR_P6;
  double PAR_P7;
  double PAR_P8;
  double PAR_P9;
  double PAR_P10;
  double PAR_P11;
  double t_lin;
};


class BMPSENSOR
{
public:
  uint8_t Address;
  double Pressure;
  uint32_t uncompPressure;
  uint32_t uncompTemperature;
  double Temperature;
  COMPCOEFFS PressureCoeffs;

  // Kinda an unnecessary
  void setAddress(uint32_t bmpAddress){
    Address = bmpAddress;
  }

  void setPressureCoeffs(){
    /*
      TODO:
```

```
  Get coefficients (21 registers (21 bytes), 17 values) (0x31 - 9x45)
  Parse coefficients
*/
uint8_t regAddress = 0x31;
int dataLen = 21;
uint8_t dataVec[21] = {0};

readMultipleBytes(Address , regAddress, dataVec, dataLen);

//Need some temporary variables
uint32_t data1;
uint32_t data2;

//The types of my NVM_PAR variables might be wrong

data1 = (uint16_t)dataVec[0];
data2 = (uint16_t)dataVec[1] << 8;
uint16_t NVM_PAR_T1 = data2 | data1;

data1 = (uint16_t)dataVec[2];
data2 = (uint16_t)dataVec[3] << 8;
uint16_t NVM_PAR_T2 = data2 | data1;

data1 = (int8_t)dataVec[4];
int8_t NVM_PAR_T3 = data1;

data1 = (int16_t)dataVec[5];
data2 = (int16_t)dataVec[6] << 8;
int16_t NVM_PAR_P1 = data2 | data1;

data1 = (int16_t)dataVec[7];
data2 = (int16_t)dataVec[8] << 8;
int16_t NVM_PAR_P2 = data2 | data1;

data1 = (int8_t)dataVec[9];
int8_t NVM_PAR_P3 = data1;
```

```
data1 = (int8_t)dataVec[10];
int8_t NVM_PAR_P4 = data1;

data1 = (uint16_t)dataVec[11];
data2 = (uint16_t)dataVec[12] << 8;
uint16_t NVM_PAR_P5 = data2 | data1;

data1 = (uint16_t)dataVec[13];
data2 = (uint16_t)dataVec[14] << 8;
uint16_t NVM_PAR_P6 = data2 | data1;

data1 = (int8_t)dataVec[15];
int8_t NVM_PAR_P7 = data1;

data1 = (int8_t)dataVec[16];
int8_t NVM_PAR_P8 = data1;

data1 = (int16_t)dataVec[17];
data2 = (int16_t)dataVec[18] << 8;
int16_t NVM_PAR_P9 = data2 | data1;

data1 = (int8_t)dataVec[19];
int8_t NVM_PAR_P10 = data1;

data1 = (int8_t)dataVec[20];
int8_t NVM_PAR_P11 = data1;



//This should be correct!
//2^-8 = 0.00390625
PressureCoeffs.PAR_T1 = ((double)NVM_PAR_T1 / 0.00390625);
//2^30 = 1073741824
PressureCoeffs.PAR_T2 = ((double)NVM_PAR_T2 / 1073741824.0);
//2^48 = 281474976710656
PressureCoeffs.PAR_T3 = (((double)NVM_PAR_T3) / 281474976710656.0);
```

```
    //2^20 = 1048576, 2^14 = 16384
    PressureCoeffs.PAR_P1 = (((double)(NVM_PAR_P1 - (16384))) /
(1048576.0));
    //2^29 = 536870912, 2^14 = 16384
    PressureCoeffs.PAR_P2 = (((double)(NVM_PAR_P2 - (16384))) /
(536870912.0));
    //2^32 = 4294967296
    PressureCoeffs.PAR_P3 = (((double)NVM_PAR_P3) / 4294967296.0);
    //2^37 = 137438953472
    PressureCoeffs.PAR_P4 = (((double)NVM_PAR_P4) / 137438953472.0);
    //2^-3 = 0.125
    PressureCoeffs.PAR_P5 = (((double)NVM_PAR_P5) / 0.125);
    //2^ = 64
    PressureCoeffs.PAR_P6 = (((double)NVM_PAR_P6) / 64.0);
    //2^8 = 256
    PressureCoeffs.PAR_P7 = (((double)NVM_PAR_P7) / 256.0);
    //2^15 = 32768
    PressureCoeffs.PAR_P8 = (((double)NVM_PAR_P8) / 32768.0);
    //2^48 = 281474976710656
    PressureCoeffs.PAR_P9 = (((double)NVM_PAR_P9) / 281474976710656.0);
    //2^48 = 281474976710656
    PressureCoeffs.PAR_P10 = (((double)NVM_PAR_P10) / 281474976710656.0);
    //2^65 = 36893488147419103232
    PressureCoeffs.PAR_P11 = (((double)NVM_PAR_P11) /
36893488147419103232.0);
    /*
    Serial.print("PAR_T1: ");
    Serial.println(PressureCoeffs.PAR_T1, 9);
    Serial.print("PAR_T2: ");
    Serial.println(PressureCoeffs.PAR_T2, 9);
    Serial.print("PAR_T3: ");
    Serial.println(PressureCoeffs.PAR_T3, 9);
    Serial.print("PAR_P1: ");
    Serial.println(PressureCoeffs.PAR_P1, 9);
    Serial.print("PAR_P2: ");
    Serial.println(PressureCoeffs.PAR_P2, 9);
    Serial.print("PAR_P3: ");
```

```
    Serial.println(PressureCoeffs.PAR_P3, 9);
    Serial.print("PAR_P4: ");
    Serial.println(PressureCoeffs.PAR_P4, 9);
    Serial.print("PAR_P5: ");
    Serial.println(PressureCoeffs.PAR_P5, 9);
    Serial.print("PAR_P6: ");
    Serial.println(PressureCoeffs.PAR_P6, 9);
    Serial.print("PAR_P7: ");
    Serial.println(PressureCoeffs.PAR_P7, 9);
    Serial.print("PAR_P8: ");
    Serial.println(PressureCoeffs.PAR_P8, 9);
    Serial.print("PAR_P9: ");
    Serial.println(PressureCoeffs.PAR_P9, 9);
    Serial.print("PAR_P10: ");
    Serial.println(PressureCoeffs.PAR_P10, 9);
    Serial.print("PAR_P11: ");
    Serial.println(PressureCoeffs.PAR_P11, 9);
    */



  }


  int8_t begin(){
    //TODO: Clean up
    //Implementing some stuff from the DFROBOT library to see if it fixes
things
    //readMultipleBytes(Address, )
    uint8_t chipIdAdress = 0x00;
    int DataLen = 1;
    uint8_t chipId = 0;
    uint8_t cmd_rdy_status;
    uint8_t cmd_err_status;


    int8_t result = readMultipleBytes(Address, chipIdAdress, &chipId, 1);
//reading chipId
    if (result == 0){ //Successfully read
      if (chipId == 0x50){ //devId is correct
```

```
        //Reset sensors and continue
        //Reads sensor status (page 31) to check if it ready for a new
command
        int8_t result = readMultipleBytes(Address, 0x03, &cmd_rdy_status,
1);
        //If true then sensor is ready for a new command
        if ((cmd_rdy_status & 0x10) && (result == 0)){
          I2C_WriteOneByte(Address, 0x7E, 0xB6); // Trigger a Softreset
(page 39)
          delay(20);

          //Checking for command error status
          result = readMultipleBytes(Address, 0x02, &cmd_err_status, 1);
          if ((cmd_err_status & 0x02) || (result != 0)){
            return -2;
          }
        }
        else{return -2;}
      }
    else {
      return -2;
      }
    }
    else{
      return -2;
    }
    //Finished with what i borrowed from DFRobot
    //All of this must be wrong
    I2C_WriteOneByte(Address, 0x1c, 0x00); // Disable oversampling pressure
    delay(20);
    I2C_WriteOneByte(Address, 0x1d, 0x00); // Output data rate is set to
200hz
    delay(20);
    I2C_WriteOneByte(Address, 0x1f, 0x00); // Disable filter
    delay(20);
    //I2C_WriteOneByte(Address, 0x1B, (byte)00110011);
```

```
    I2C_WriteOneByte(Address, 0x1B, 0x03 | 0x30); // Sets sensor to normal
mode - can maybe just use 00110001 (or 00110011)


    setPressureCoeffs();

    return 0;

  }


  //Is used in getTemperature

  void compensateTemperature(){

    //Algorithm for compensating is taken from page 55 and 56 of the
technical documentation

    //https://www.bosch-sensortec.com/products/environmental-
sensors/pressure-sensors/bmp388/#technical

    //Serial.println("Compensating temp: ");

    double partial_data1;

    double partial_data2;


    partial_data1 = (double)(uncompTemperature - PressureCoeffs.PAR_T1);

    partial_data2 = (double)(partial_data1 * PressureCoeffs.PAR_T2);

    /*

    Serial.print("Partial_data1: ");

    Serial.println(partial_data1);

    Serial.print("Partial data2: ");

    Serial.println(partial_data2);

    */

    PressureCoeffs.t_lin = partial_data2 + (partial_data1 *
partial_data1)*PressureCoeffs.PAR_T3;

    Temperature = PressureCoeffs.t_lin;

  }


  void updateTemperature(){

    /*

    Wire.beginTransmission(Address);

    Wire.write(0x07); // Set the pointer for where im reading

    Wire.endTransmission();

    */

    uint8_t regAddress = 0x07;

    int dataLen = 3;
```

```
    uint8_t dataVec[dataLen] = {0};

    readMultipleBytes(Address , regAddress, dataVec, dataLen);

    //Need some temporary variables
    uint32_t data1;
    uint32_t data2;
    uint32_t data3;

    data1 = (uint32_t)dataVec[0];
    data2 = (uint32_t)dataVec[1] << 8;
    data3 = (uint32_t)dataVec[2] << 16;

    uncompTemperature = data3 | data2 | data3;

  }

  void compensatePressure(){
    //Algorithm for compensating is taken from page 55 and 56 of the
technical documentation
    //https://www.bosch-sensortec.com/products/environmental-
sensors/pressure-sensors/bmp388/#technical

    double partial_data1 = 0;
    double partial_data2 = 0;
    double partial_data3 = 0;
    double partial_data4 = 0;
    double partial_out1 = 0;
    double partial_out2 = 0;

    partial_data1 = PressureCoeffs.PAR_P6 * PressureCoeffs.t_lin; //ish
10800
    partial_data2 = PressureCoeffs.PAR_P7 *
(PressureCoeffs.t_lin*PressureCoeffs.t_lin); // ish -24
    partial_data3 = PressureCoeffs.PAR_P8 *
(PressureCoeffs.t_lin*PressureCoeffs.t_lin*PressureCoeffs.t_lin); //ish -
3,24
    partial_out1 = PressureCoeffs.PAR_P5 + partial_data1 + partial_data2 +
partial_data3; // ish 211100
```

```
    partial_data1 = PressureCoeffs.PAR_P2 * PressureCoeffs.t_lin; //lite

    partial_data2 = PressureCoeffs.PAR_P3 *
(PressureCoeffs.t_lin*PressureCoeffs.t_lin); //knøttlite

    partial_data3 = PressureCoeffs.PAR_P4 *
(PressureCoeffs.t_lin*PressureCoeffs.t_lin*PressureCoeffs.t_lin);
//knøttlite

    partial_out2 = uncompPressure * (PressureCoeffs.PAR_P1 + partial_data1
+ partial_data2 + partial_data3);//ish negative -1100000


    partial_data1 = ((double)uncompPressure) * ((double)uncompPressure);
//kjempestort

    partial_data2 = PressureCoeffs.PAR_P9 + (PressureCoeffs.PAR_P10 *
PressureCoeffs.t_lin); //knøttlite

    partial_data3 = partial_data1 * partial_data2; //uuuh,
kjempestort*knøttlite

    partial_data4 = partial_data3 + (((double)uncompPressure) *
((double)uncompPressure) * ((double)uncompPressure)) *
PressureCoeffs.PAR_P11;

    /*

    Serial.println("Compensating pressure: ");

    Serial.print("Partial out1 : ");

    Serial.println(partial_out1);

    Serial.print("Partial out2 : ");

    Serial.println(partial_out2);

    Serial.print("Partial data4 : ");

    Serial.println(partial_data4);

    */

    Pressure = partial_out1 + partial_out2 + partial_data4;

  }


  void updatePressure(){

    //Remember to run temperature first to update t_lin

    /*

    Wire.beginTransmission(Address);

    Wire.write(0x04); // Set the pointer for where im reading

    Wire.endTransmission();

    */

    uint8_t regAddress = 0x04;

    int dataLen = 6;
```

```
    uint8_t dataVec[dataLen] = {0};

    readMultipleBytes(Address , regAddress, dataVec, dataLen);

    //Need some temporary variables
    uint32_t data1;
    uint32_t data2;
    uint32_t data3;

    data1 = (uint32_t)dataVec[0];
    data2 = (uint32_t)dataVec[1] << 8;
    data3 = (uint32_t)dataVec[2] << 16;

    uncompPressure = data3 | data2 | data1;

    data1 = (uint32_t)dataVec[3];
    data2 = (uint32_t)dataVec[4] << 8;
    data3 = (uint32_t)dataVec[5] << 16;

    uncompTemperature = data3 | data2 | data1;

    compensateTemperature();
    compensatePressure();

  }
};

// ----------------- Finished with function and class definitions ---------
----------------

BMPSENSOR BMP0;
BMPSENSOR BMP1;
BMPSENSOR BMP2;
BMPSENSOR BMP3;
BMPSENSOR BMP4;
BMPSENSOR BMP5;
BMPSENSOR BMP6;
```

```
BMPSENSOR BMP7;
BMPSENSOR BMP8;
BMPSENSOR BMP9;
BMPSENSOR BMP10;
//Comment in if using scale
HX711 scale;
int i = 1;
float scaleValue = 0;

void setup()
{
  delay(5000); //just for debugging
  Serial.begin(115200);
  Wire.begin();
  Wire.setClock(clockFrequency);

  //Comment in if using scale
  scale.begin(LOADCELL_DOUT_PIN, LOADCELL_SCK_PIN, 128);


  //Starting up sensors and setting addresses
  tcaselect(0);
  BMP0.Address = BMP_ADDR2;
  while(BMP0.begin() != 0){
    Serial.println("Error starting up sensor 0");
    delay(200);
  }
  BMP1.Address = BMP_ADDR1;
  while(BMP1.begin() != 0){
    Serial.println("Error starting up sensor 1");
    delay(200);
  }
  tcaselect(1);
  BMP2.Address = BMP_ADDR2;
  while(BMP2.begin() != 0){
    Serial.println("Error starting up sensor 2");
```

```
  delay(200);
}
BMP3.Address = BMP_ADDR1;
while(BMP3.begin() != 0){
  Serial.println("Error starting up sensor 3");
  delay(200);
}
tcaselect(2);
BMP4.Address = BMP_ADDR2;
while(BMP4.begin() != 0){
  Serial.println("Error starting up sensor 4");
  delay(200);
}
BMP5.Address = BMP_ADDR1;
while(BMP5.begin() != 0){
  Serial.println("Error starting up sensor 5");
  delay(200);
}
tcaselect(3);
BMP6.Address = BMP_ADDR2;
while(BMP6.begin() != 0){
  Serial.println("Error starting up sensor 6");
  delay(200);
}
BMP7.Address = BMP_ADDR1;
while(BMP7.begin() != 0){
  Serial.println("Error starting up sensor 7");
  delay(200);
}
tcaselect(4);
BMP8.Address = BMP_ADDR2;
while(BMP8.begin() != 0){
  Serial.println("Error starting up sensor 8");
  delay(200);
}
BMP9.Address = BMP_ADDR1;
```

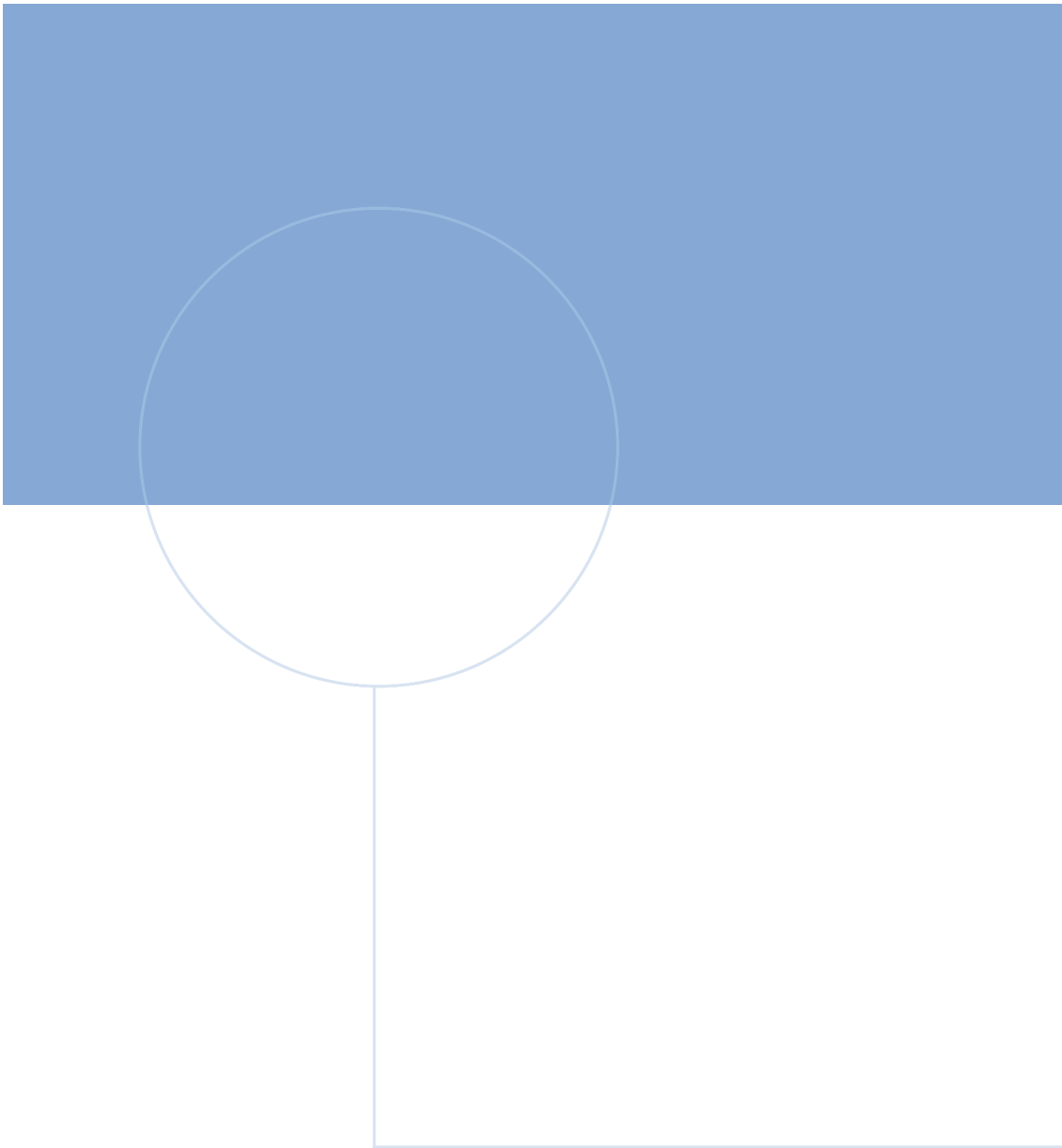
```
  while(BMP9.begin() != 0){
    Serial.println("Error starting up sensor 9");
    delay(200);
  }
  delay(200);
}

void loop()
{
  //reading pressure values from sensors
  //Serial.print(millis());
  //Serial.print(" - ");
  tcaselect(0);
  BMP0.updatePressure();
  BMP1.updatePressure();
  tcaselect(1);
  BMP2.updatePressure();
  BMP3.updatePressure();
  tcaselect(2);
  BMP4.updatePressure();
  BMP5.updatePressure();
  tcaselect(3);
  BMP6.updatePressure();
  BMP7.updatePressure();
  tcaselect(4);
  BMP8.updatePressure();
  BMP9.updatePressure();
  //Serial.println(millis());

  //printing time and pressure values
  Serial.print(millis());
  Serial.print(",");
  //Comment in if using PPG
  Serial.print(analogRead(PPGPin));
  Serial.print(",");
  //Comment in if using scale
```

```
if (i%2 == 0){
  scaleValue = scale.read();
  Serial.print(scaleValue);
  Serial.print(',');
}
else{
  Serial.print(scaleValue);
  Serial.print(",");
}
//right foot furthest back
Serial.print(BMP0.Pressure);
Serial.print(",");
//right foot second to the furthest back
Serial.print(BMP1.Pressure);
Serial.print(",");
//right foot forward to the right
Serial.print(BMP2.Pressure);
Serial.print(",");
//right foot middle
Serial.print(BMP3.Pressure);
Serial.print(",");
//left foot forward right
Serial.print(BMP4.Pressure);
Serial.print(",");
//right foot forward left
Serial.print(BMP5.Pressure);
Serial.print(",");
//left foot middle
Serial.print(BMP6.Pressure);
Serial.print(",");
//left foot forward left
Serial.print(BMP7.Pressure);
Serial.print(",");
//left foot second to furthest back
Serial.print(BMP8.Pressure);
Serial.print(",");
```

```
  //left foot furthest back
  Serial.println(BMP9.Pressure);
  i++;
  delay(4); // delay to ensure im not rereading the same value twice.
Should implement a smarter way to do this
  }
```



NTNU
Kunnskap for en bedre verden