Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Tobias Thorvaldsen

Design of sensor system for slump test of fresh concrete

Master's thesis in Electronic Systems Design
Supervisor: Dag Roar Hjelme
Co-supervisor: Dominik Osinski
June 2022

NTNU
Norwegian University of Science and Technology

Tobias Thorvaldsen

Design of sensor system for slump test of fresh concrete

Master’s thesis in Electronic Systems Design
Supervisor: Dag Roar Hjelme
Co-supervisor: Dominik Osinski
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Abstract

Getting a measure of the consistency of concrete is essential to identify whether a particular batch of fresh concrete is appropriate for construction operations. The slump test is one of many existing methods for finding the consistency of fresh concrete and is the most used method in Norway. Businesses in the construction industry now seek a digital solution to replace the standard slump test to facilitate the conduction of the test and increase its efficiency. To this end, a measurement concept using an accelerometer to measure the impact of an object falling into the fresh concrete was created, and a prototype was designed to examine whether the measurement concept works as an equivalent to the standard slump test. The sensor system implements an accelerometer, with an output data rate of 1600 Hz and a measurement range of ± 200 g, with an Arduino Nano 33 BLE. A python script that runs on a nearby laptop handles control of the sensor system and storage of the sampled data, allowing it to be appropriately analyzed. Preliminary tests were conducted, giving valuable insight into how further testing should be carried out, albeit raising more questions than giving answers. No definite conclusion could be made on whether the developed measurement concept could function as a way of measuring the consistency of fresh concrete.

Sammendrag

Å få et mål på konsistensen til fersk betong er essensielt for en bygningsarbeider for å vite om et spesifikt parti med betong kan brukes. Slumptesten er en av mange eksisterende metoder for å måle konsistensen til fersk betong og er den mest brukte metoden i Norge. Bedrifter innenfor bygg- og anleggssektoren søker nå en mulig digital løsning som kan erstatte slumptesten, med formål om å øke effektiviteten og gjøre det enklere å ta konsistensmålinger av fersk betong. Et målekonsept som bruker et akselerometer til å måle støtet som oppstår når et objekt faller ned i betongen ble laget, og en prototype ble konstruert for å teste ut om målekonseptet kan fungere som en ekvivalent til slumptesten. Sensorsystemet er bygget opp av et akselerometer, med en utdatahastighet på 1600 Hz og et måleområde på $\pm$ 200 g, og en Arduino Nano 33 BLE. Et script skrevet i Python som kjøres på en nærstående bærbar PC kontrollerer sensorsystemet og lagrer dataen fra målingene slik at de kan analyseres i etterkant. Innledende tester ble gjennomført som ga god innsikt i hvordan fremtidig testing burde gjennomføres, selv om de resulterte i at man stod igjen med flere spørsmål enn svar. Ingen definitiv konklusjon kunne tas på om det utviklede målekonseptet kan fungere til å måle konsistensen av fersk betong.

Acknowledgements

I would like to thank Dominik Osinski at NTNU for all the help he has given me throughout the project. I would also like to thank Gunrid Kjellmark and Natalia Iakymenko at SINTEF for giving me the fantastic opportunity to work on such an exciting project. I want to give thanks to Yannick Martin Anton at SINTEF for allowing me to come visit and conduct testing of the prototype at SINTEF's concrete laboratory. I am very grateful to Kari Aarstad at Unicon AS for setting up and giving Dominik and me a guided tour around one of Unicon's concrete factories. I must also thank Olav Aleksander Myrvang at NTNU for helping me with finding components for and creating the prototype. I would also like to thank the people at the electronics and prototype laboratory at NTNU for soldering the circuit. I am also extremely grateful to Erling Bakken, Christer Nesset and Morten Ekeberg at NTNU for constructing the housing for the sensor system. Making the prototype would never have been completed without their help. Lastly, I would like to thank all the concrete experts who helped provide valuable information about concrete and feedback on the measurement concept.

Table of Contents

List of Figures

List of Tables

List of abbreviations

Abbreviations and acronyms	Definition
g	Gravitational acceleration
BLE	Bluetooth low energy
IoT	Internet of things
RAM	Random access memory
SRAM	Static random access memory
I/O	Input/output
PWM	Pulse width modulation
UART	Universal asynchronous receiver-transmitter
SPI	Serial peripheral interface
I2C	Inter-integrated circuit
ADC	Analog-to-digital converter
DAC	Digital-to-analog converter
RMS	Root mean square
ODR	Output data rate
HPF	High-pass filter
LPF	Low-pass filter
CNC	Computer numerical control
CSV	Comma-separated values
FIFO	First in first out
MSB(s)	Most significant bit(s)
LSB(s)	Least significant bit(s)
LiPo	Lithium-polymer

Chapter 1

Introduction

1.1 Background

1.1.1 Digital transformation in the construction industry

Productivity in the construction industry has declined in several countries since 2000. Statistics Norway [1] did calculations back in 2018 which indicated that the productivity in the construction industry had decreased with 10%, whereas the productivity in all other market-oriented businesses in mainland Norway had increased by 30% in the same period (2000 – 2016). Productivity is defined as gross product at constant prices per hour worked. The inability of construction businesses to adopt process and technology innovations has been viewed as one of the main reasons for this. Compared to almost all other big industries, the construction industry is one of the least digitized. A digital transformation is required to meet the increasing demand of housing and infrastructure, at the same time as contractors' financial returns are often low and volatile. Digital transformation can be defined as the implementation of advanced technologies and ways of working to enhance the development and delivery of projects [2, 3, 4, 5].

Construction projects going over budget and taking longer to finish than planned has also become a common occurrence. Over the last couple of years, many businesses have made digital transformation attempts, but several of the attempts have been unsuccessful. The main problem is that businesses have worked on digital transformation separately instead of cooperating. A typical construction project involves a multitude of independent contractors and suppliers, and projects are never identical, making it difficult to implement digital solutions that all involved parties are willing to adopt, which can also be used on future projects. Cooperation across the value chain is reportedly the the only way to achieve digital transformation [3, 4, 5, 6].

Implementing sensor technology to improve the efficiency and the accuracy of existing methods and procedures is one of the many areas being analyzed to address the negative trend in productivity. The last couple of years have seen a significant increase in IoT technology being used in a vast number of applications like smart housing, manufacturing, agriculture, energy management and many more. This makes it reasonable that the construction industry also seeks possible ways to implement IoT into their value chain cost-efficiently. A prerequisite for investing in sensor technology is that it improves existing procedures and methods, meaning that in-depth studies are required to identify what can be improved with the use of sensors [3, 4, 5, 6].

1.1.2 SiteCast

SiteCast is an example of a project where several businesses have come together to find solutions to increase efficiency and reduce cost. Specifically, the SiteCast project seeks to solve challenges with site casting of concrete, which is when a concrete element is cast at the construction site. The other typical way of constructing with concrete is by using precast concrete, where the elements are cast in a factory and then transported to the construction site. Areas of investigation for the project are coordination, materials, engineering, planning, and use of sensor technology. The end goal is to reduce time consumption with 30% and cost by 15% compared to what was the level at the start of the project, in 2017. At the time of writing, the project is still ongoing [7, 8, 9].

1.1.3 Sensor system for slump test of concrete

Finding a digital solution for consistency measurements of fresh concrete is one of the things being investigated as part of the SiteCast project. The basic idea is to use sensor technology to replace an already existing method, called slump test. A sensor system must give measurements at least as accurate as the slump test, but it should be easier and faster to perform. Knowing the consistency of fresh concrete is important for construction workers to know if a specific batch of concrete delivered to the construction site is usable. Whether it is usable or not depends on the structural elements that are intended to be cast with the concrete. For example, casting a floor requires a different consistency of the concrete than casting a wall.

1.2 Client

This thesis is carried out on the request from SINTEF and Unicon AS. SINTEF is one of Europe's largest independent research organizations, with head office in Trondheim, Norway [10]. SINTEF is organised as an enterprise group consisting of six research institutes [10]: SINTEF Community, SINTEF Digital, SINTEF Energy, SINTEF Industry, SINTEF Manufacturing and SINTEF Ocean [11]. My contact person at SINTEF, and for the SiteCast project, is Gunrid Kjellmark, research manager at SINTEF Community [12]. Unicon AS is Norway's leading concrete supplier and is a wholly owned subsidiary of the Italian Cementir Group [13].

1.3 Problem description

The goal of this master's thesis is to develop a measurement concept, using sensors, which can replace the slump test. A prototype will then be designed based on the measurement concept; to test if it can work as an equivalent to the slump test. Some prerequisites for the sensor system are: It needs to deliver results of at least the same accuracy as the slump test; it needs to be easy to use; it should require less equipment to carry out than the slump test; and it needs to be faster to perform than the slump test. The thesis is a continuation of a project that has been worked on for three semesters already.

1.4 Delimitation

Creating a sensor system based on a measurement concept, of which no existing research can be found, requires extensive knowledge of what is to be measured. In this case, this is the consistency of fresh concrete. The consistency is just one of many interacting properties of concrete, which complexity means that the entire master's thesis could go into only learning about concrete. Without existing research to use as a basis for the sensor system, a prototype would have to be constructed to test whether or not the measurement concept works to measure the consistency. The time limitations of the thesis requires the work from the previous semesters to be used as the basis for creating a prototype to test the measurement concept. No time will be spent on developing other measurement concepts, even if the existing one might not be the ideal solution.

1.5 Report Structure

This report presents the work done on the master's thesis, including the most important parts of the work previously done on the project. The report is built up of four main parts. These are Chapter 2: Theory, Chapter 3: Methodology, Chapter 4: Results, and Chapter 5: Discussion. In Chapter 2: Theory, the basics of concrete and an explanation of some methods for measuring the consistency of concrete are presented to explain what the sensor system is supposed to do, and what it is supposed to replace. It mainly consists of revised work from previous semesters. In Chapter 3: Methodology, an explanation of how the problem was approached and attempted solved is presented. Section 3.1 is the part of this chapter that presents work mostly done in previous semesters. In Chapter 4 Results, the results from testing are presented. Chapter 5: Discussion shows the analysis of the results, the lessons learned, and reflections about the project, as well as ideas and thoughts relevant for possible further work on the project.

Chapter 2

Theory

2.1 Basics of concrete

Concrete is the second-most-used substance in the world after water, and is an artificial composite material made from cement, aggregates, and water. Typical aggregates are sand, crushed stone, and gravel. The larger aggregates like crushed stone and large gravel are generally referred to as coarse aggregates and the smaller aggregates like small gravel and sand are generally referred to as fine aggregates. Admixtures in the form of powder or fluids can be added to a concrete mixture to give it certain characteristics. Some examples of admixtures are: accelerators, which speeds up the setting/hardening process of the concrete; plasticizers and superplasticizers, which increase the workability of the fresh concrete, allowing it to be placed more easily; and pigments, which alter the color of the concrete, used for aesthetic purposes [14, 15, 16, 17].

Combining cement and water creates a glue-like substance called cement-paste, which binds the aggregates together. The Amount of water content in the paste is the main factor affecting the workability of a concrete mixture and its strength. A simplified explanation of the effect water content has on the workability and the strength is that high water content gives high workability but low strength, while low water content gives low workability and high strength. With the correct admixtures however, it is possible to get a concrete mixture with high strength and high workability with the use of little water, which is the perfect scenario [14, 15, 16, 17, 18, 19].

The mixing ratio between cement-paste and aggregates, and the types of aggregates used, vary depending on what will be constructed with the mixture. Regardless of the application for a concrete mixture, the degree of compaction is of great importance, especially for the strength of the concrete. Compaction of concrete is an operation with the goals of expelling voids, i.e., entrapped air, in freshly placed concrete, and packing aggregates together to increase the density of the concrete. Increased density gives increased strength. Concrete is usually compacted by the means of vibrating or ramming [14, 15, 16, 17, 18, 20, 21].

2.2 Workability and consistency of fresh concrete

Workability is best defined as the amount of work needed to achieve full compaction. Another more quantitative definition of workability is that: Workability is a property determining the effort required to manipulate a freshly mixed quantity of concrete with minimum loss of homogeneity. Concrete mixture homogeneity is a percentage according to the given composition of components, with a mixture being considered homogeneous if samples taken from different places in the mixer contain the components of the mixture in equal percentages [22]. Another term used to describe the state of fresh concrete is consistency. Consistency of concrete is best defined as the relative mobility or ability of freshly mixed concrete to flow. In a way, it describes the degree of wetness of a concrete mixture. Within limits, wet concrete mixtures are more workable than dry concrete mixtures. There exists more definitions of workability and consistency than presented above [17, 19, 21, 23, 24].

As described by A. M. Neville:

> Technical literature abounds with variations of the definitions of workability and consistency but they are all qualitative in nature and more reflections of a personal viewpoint rather than of scientific precision [21].

2.3 Tests for measuring consistency and workability of fresh concrete

2.3.1 Slump test

Slump test of fresh concrete is one of the most used on-site tests carried out to determine the consistency and the workability of a concrete mixture. The necessary equipments for conducting a slump test are: A quadratic metal plate with side lengths of 700 mm; a hollow cone with a height of 300 mm and opening widths of 100 mm and 200 mm; a metal rod with a length of 380 mm measuring 16 mm in diameter; and something to measure length/height [19, 21, 25, 26, 27, 28].

A slump test is then performed with the following procedure, according to [19, 21, 25, 26, 27, 29]:

1. The inside and base, the 200 mm opening, of the cone is moisturized to reduce any surface friction which might influence the measurement.
2. The cone is placed atop the metal plate, in the center of it, with the base facing down.
3. A sample of fresh concrete is extracted from the batch that is to be examined.
4. The concrete is poured into the cone in 3 rounds, each round filling about 1/3 of the cone's height. Between each round the concrete is tamped 25 times with the rod.

5. When the cone is filled, any concrete reaching above the height of the cone, and any concrete that has dropped down on the metal plate, is removed.

6. The cone is lifted vertically up, without sideways movement or twisting, over the course of 2 to 5 seconds.

7. The concrete will then slump. Figure 2.1 shows different examples of slump that might occur. True slump indicates the test was successful, and the measurement can be taken. Zero slump and collapsed slump indicates that the workability/consistency is outside the limits of the test. A shear slump indicates that the test went wrong and must be redone. If shear slump persists it is an indication of lack of cohesion in the mix, which means that aggregates may separate, and that the mixture is of unsatisfactory quality.

8. In the case of true slump, the decrease in the height of the slumped concrete, from the cone's height, is measured to the nearest 10 mm. This is the slump value.

9. The measured slump value is then compared with a table given in for example standard NS-EN 206 [30]. The standard used may vary from country to country. This table is shown in Table 2.1 and assigns the measured slump to a slump class ranging from S1 to S5. S1 equals low workability and consistency, and S5 equals high workability and consistency.

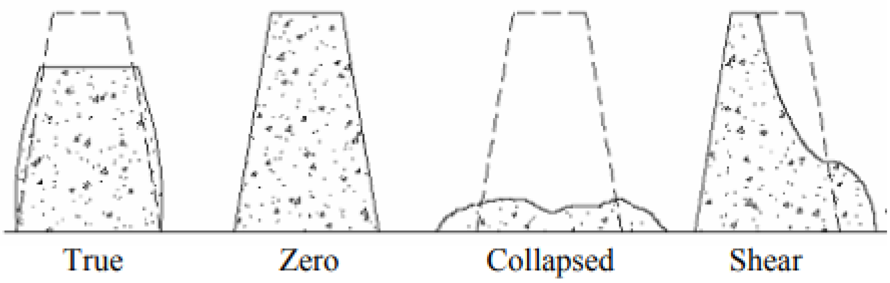


Figure 2.1: Different types of slump [19].

Table 2.1: Slump classes [29].

Slump Class	Slump (mm)
S1	10 to 40
S2	50 to 90
S3	100 to 150
S4	160 to 210
S5	≥ 220

Although the slump test is described to measure the workability of concrete, it does not measure it directly. The same slump value can be measured for concrete mixtures with different workabilities, depending on the composition of aggregates in the mixture. It is still a useful on the site check on the batch-to-batch or hour-to-hour variation in the materials being fed into the mixer [21].

2.3.2 Slump flow test

The slump flow test of fresh concrete is similar to the slump test. It is usually performed on concrete that will show collapsed slump with the slump test, i.e., on concrete with consistency/workability higher than what can be measured with the slump test. The necessary equipment for carrying out the slump flow test is: A quadratic metal plate with side lengths of 700 mm; A hollow cone with a height of 200 mm and opening widths of 130 mm and 200 mm; and something to measure length/height [31].

A slump flow test is then performed with the following procedure, according to [31]:

1. Place the cone atop the metal plate, in the center of it, with the base facing down.
2. A sample of fresh concrete is extracted from the batch that is to be examined.
3. The concrete is poured into the cone.
4. When the cone is filled, any concrete reaching above the height of the cone, and any concrete that has dropped down on the metal plate, is removed.
5. The cone is lifted vertically up, without sideways movement or twisting.
6. The concrete will flow out on the plate as in Figure 2.2.
7. The largest diameter, and the diameter perpendicular to that, are measured and the average of them is calculated. This is the flow value.
8. The flow value is then compared with a table given by for example standard NS-EN 206 [29]. The standard used may vary from country to country. This table is shown in Table 2.2 and assigns the measured flow value to a flow class ranging from F1 to F6. F1 equals low workability and consistency, and F6 equals high workability and consistency.

Table 2.2: Flow classes [29].

Flow Class	Flow (mm)
F1	≤ 340
F2	350 to 410
F3	420 to 480
F4	490 to 550
F5	560 to 620
F6	≥ 630

Figure 2.2: Slump flow test concrete spread.

2.3.3 Kelly ball test

The Kelly ball test, or ball penetration test, is a test that is similar to the slump test, serving as a simple on-site check of the consistency for control purposes. It is rarely used outside the United States. Necessary equipment for carrying out the test is the Kelly ball, see Figure 2.3, with its name stemming from the inventor of the test, J. W. Kelly. It consists of a 13.6 kg metal ball with a diameter of 152 mm attached to a stem which slides through a metal frame. When testing, the apparatus is placed on a larger surface of fresh concrete with the frame resting on the concrete. The ball's weight will make it sink into the concrete, and the penetration depth is what is used as a measurement of the consistency and the workability. Because it requires less equipment and can be carried out in the concrete form, it is faster and simpler than the slump test [19, 21].

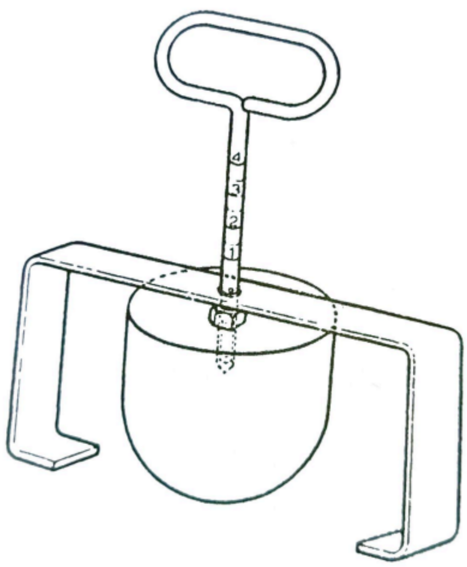


Figure 2.3: Kelly ball [21].

Chapter 3

Methodology

3.1 Developing the measurement concept

The first step in creating a sensor system was to develop a measurement concept. Through the literature study it was not discovered any existing concept for measuring consistency of fresh concrete with sensors *in situ*. Methods that use sensors exist but are all limited to being performed in a laboratory. Research into existing concrete sensors used *in situ* only showed sensors used for monitoring concrete during the hardening period or already hardened concrete, see [32] and [33] for examples. This meant that a measurement concept had to be made from scratch. An attempt to dive into the properties defining fresh concrete only gave insight into how complex a material it is. Advice from concrete experts was needed to limit the workload of acquiring sufficient knowledge about concrete to be able to say what can be measured from it. The main teaching from the counseling was that there are no single or few parameters that can be measured directly to define the consistency of fresh concrete. Based on the definition presented in section 2.2, this makes sense:

> Consistency of concrete is best defined as the relative mobility or ability of freshly mixed concrete to flow. In a way, it describes the degree of wetness of a concrete mixture.

The definitions for consistency of fresh concrete discovered through the literature study were all vague like the one above. Existing tests for measuring the consistency and the workability of fresh concrete do not measure it directly. The slump test, slump flow test and Kelly ball test presented in section 2.3 are all based on the observed behavior of fresh concrete. Consulting with professionals gave the impression that experience is the main factor for determining whether a batch of fresh of concrete has the required consistency or not. A concrete laboratory and a concrete factory were visited to gain experience that could help with developing a measurement concept. Both gave valuable insight that could not have been gained through mere reading, but raised more questions than provide solutions to already existing issues.

Going back to the basis of the project, the goal was to create a sensor system that could replace the slump test. The deep dive into the properties of concrete showed that looking at the problem from a simplified perspective was needed to be able to create anything at all. Developing a method that could be tested and correlated with the slump test was deemed a good direction for the project. Based on this, a concept was developed where the idea was to have a sensor system that could be dropped into the fresh concrete. The resulting characteristic of the impact created by the sensor system landing in the concrete would be correlated with the slump test to determine the consistency. Exactly which part of the characteristic that could be used must be found through testing.

In an ideal complete solution, the sensor system can be controlled via an application on a phone or a tablet using Bluetooth to connect with it. Measurements would be carried out in the following way:

1. A sample of fresh concrete is extracted from the batch that is to be examined.
2. The concrete is placed in a 20 L bucket.
3. The sensor system is activated with the app.
4. The sensor system is dropped into the fresh concrete. Figure 3.1 shows an illustration of the measurement setup.
5. Results from the measurement is shown in the app.
6. The sensor system is deactivated with the app.

If this way of measuring the consistency of fresh concrete is possible, it will be much simpler and more efficient than the slump test and slump flow test. It is important to note that a finished solution would probably look different from the one presented above, as it is just a concept.

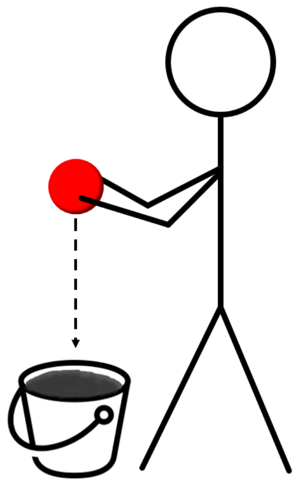


Figure 3.1: Illustration of measurement setup for ideal complete sensor solution.

3.2 Making a prototype

3.2.1 Hardware

Choosing components

The next step in the process was to choose components for constructing a prototype. Prerequisites for the prototype were that it must be able to measure the force of an impact and transmit raw data to a PC. Transmitting the data to a PC was needed because there was still uncertainty connected to how the measurement would be used to correlate with the slump test measurements. Because the sensor system would be dropped into the fresh concrete, a wired connection to a PC would be impractical. Therefore, it was decided that the data should be transmitted via Bluetooth. Based on this, a complete prototype of the sensor system would consist of an accelerometer, for measuring the impact; a microcontroller board with a Bluetooth radio, for controlling the system and transmitting data; and necessary parts for powering the system with a battery.

Arduino Nano 33 BLE

A microcontroller board that met the mentioned requirements was the Arduino Nano 33 BLE. It is based on the nRF52840 microcontroller, see [34]. The datasheet for the Nano 33 BLE can be found at [35]. A full pinout diagram can be found at [36]. Noteworthy specifications are shown in Table 3.1.

Table 3.1: Noteworthy specifications for Arduino Nano 33 BLE [37].

Parameter	Value / Note
Operating voltage	3.3 V
Input voltage	5 - 21 V
Clock speed	64 MHz
CPU flash memory	1 MB
SRAM	256 kB
Numb. of digital I/O pins	14
PWM pins	All digital pins
UART	Yes
SPI	Yes
I2C	Yes
Numb. of analog input pins	8 (ADC 12 bit 200 ksamples)
Analog output pins	Only through PWM (no DAC)
External interrupts	All digital pins
USB	Native to nRF52840 processor
Length	45 mm
Width	18 mm

An operating voltage of 3.3 V limits the accompanying accelerometer to be sufficient with a supply voltage of 3.3 V. The Nano has a 5 V output pin, but it can only be used if it is powered via the USB port. Other than that, the availability of both analog and digital input pins, and the built in SPI and I2C, opened up for a vast number of accelerometers to be used. The small size of the board helps with keeping the system as small as possible.

EVAL-ADXL372Z

An accelerometer that met the requirements was the ADXL372. The datasheet can be found at [38]. The evaluation board variant of the accelerometer, the EVAL-ADXL372Z, was chosen to simplify hooking up the sensor system, eliminating the need for a PCB. A user guide for the EVAL-ADXL372Z can be found at [39]. Noteworthy specifications are shown in Table 3.2.

Table 3.2: Noteworthy specifications for ADXL372 [38].

Parameter	Value / Note
Measurement range	± 200 g
Cross axis sensitivity	± 2.5 %
Output resolution	12 Bits
Scale factor	100 mg/LSB
RMS noise (normal operation)	3.5 LSB
RMS noise (low noise mode)	3 LSB
ODR	400 - 6400 Hz
HPF, -3 dB corner	0.24 - 30.48 Hz
LPF, -3 dB corner	200 - (ODR / 2) Hz
Operating voltage range	1.6 - 3.5 V
SPI	Yes
I2C	Yes

The EVAL-ADXL372Z is equipped with three factory-installed capacitors for bypass, see Figure 3.2 for reference: two 0.1 μF capacitors, C1 and C2, and a 10 μF capacitor, C3. C2 and C3 are VS bypass capacitors for reducing analog supply noise and C1, located between VIO and GND, is for reducing digital clocking noise. It also has readied holes for installing headers. High specifications for the measurement range and output data range were good, given the uncertainty connected to the peak acceleration and frequency of the impact [38].

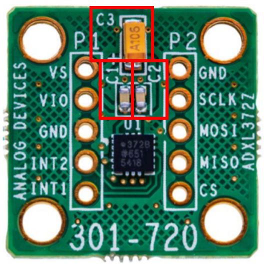


Figure 3.2: Top layout of EVAL-ADXL372Z, snippet from [39].

Battery setup

A PowerBoost 500C accompanied with a 250 mAh 3.7 V LiPo battery made up the battery setup for powering the system. This was chosen to keep the system small and practical, with the PowerBoost allowing for recharging the battery, as well as boosting the voltage to the necessary 5 V needed for powering the Arduino. Even though the PowerBoost comes with a battery connector, it did not fit with the connector on the acquired battery. Therefore, a separate connector was added to the circuit. More information about the PowerBoost can be found at [40, 41, 42].

Complete circuit

With all the necessary components acquired, the system was connected on a perf-board. Figure 3.3 shows the finished circuit without the battery, which was connected with a strong double-sided tape after the picture was taken. Figure 3.4 shows a circuit diagram of the wire-connections. A resistor, shown as R1 in the circuit diagram, was added to reduce noise on the accelerometer, as recommended in the datasheet. The added separate connector can be seen in green, placed between the Nano and the PowerBoost, in Figure 3.3.

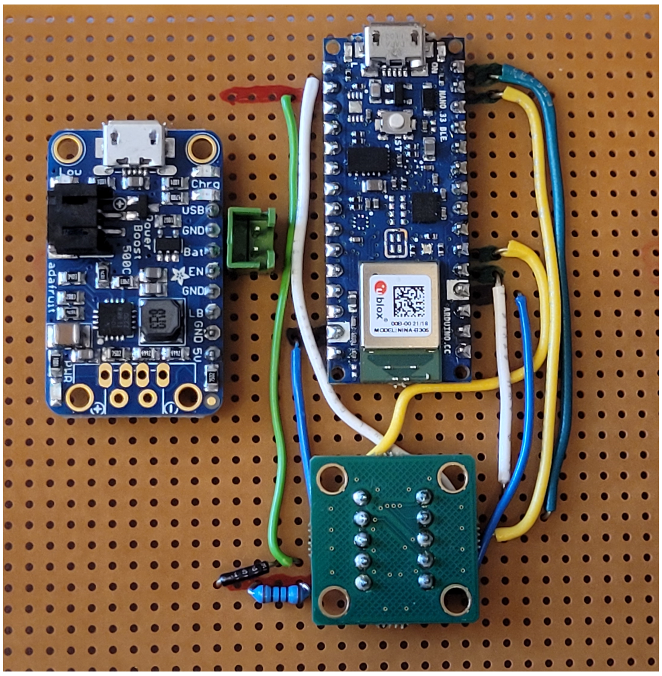


Figure 3.3: Picture of sensor system circuit, without battery.

3.2.2 Software

Software for the sensor system was divided into two main programs: One program that manages the sampling of data on the accelerometer and transmits the raw data via Bluetooth to an external device for processing; and one program that receives said data and processes it. Henceforth these will be referred to as the sampling

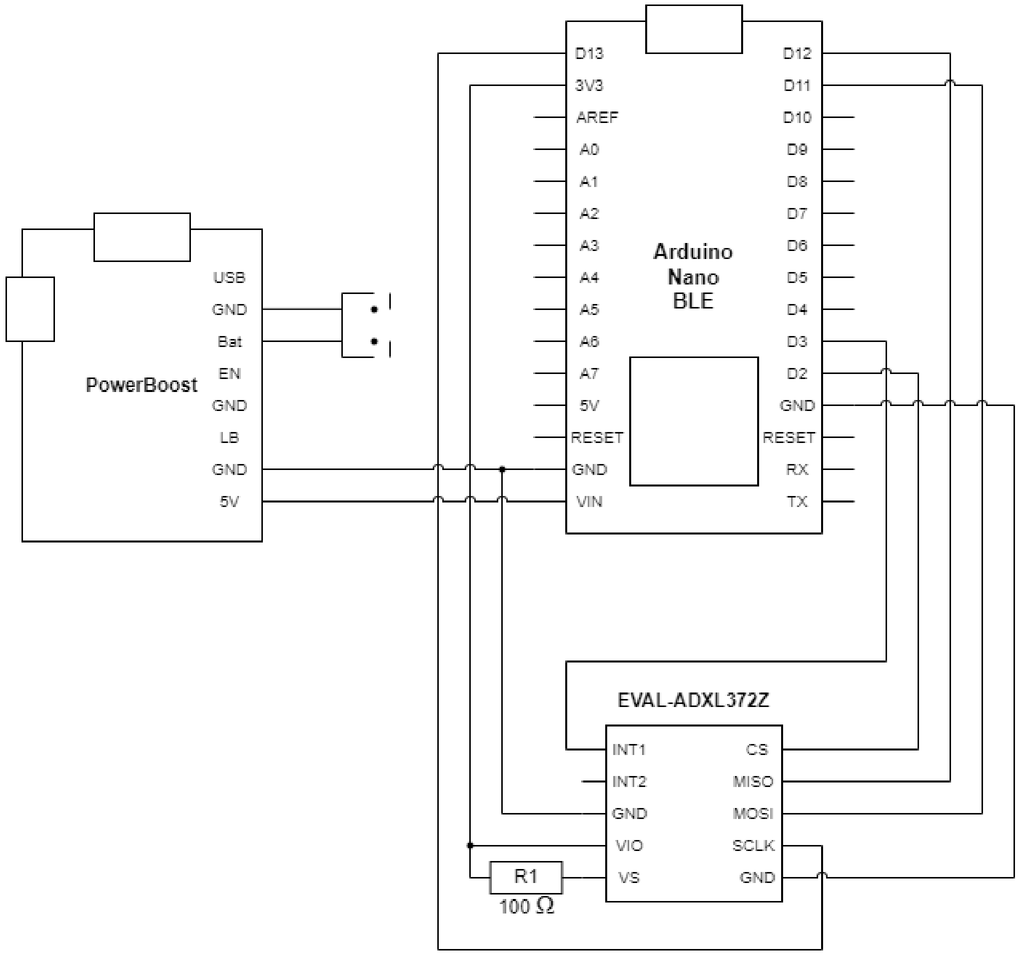


Figure 3.4: Circuit diagram of sensor system wire-connections.

program and the processing program, respectively. Since an Arduino microcontroller is used to interface with the accelerometer, the sampling program is written in C++. Python has several Bluetooth libraries available which simplifies the programming work, making Python a good choice for writing the processing program.

Sampling program

In total, the sampling program consists of three scripts: A main script, a script with functions for setting up the accelerometer, and a header file for the accelerometer functions script; all respectively shown in Appendix A, Appendix B and Appendix C. The ADXL372 has a vast amount of settings, which alter the functionality of the accelerometer. It can, for example, be set up to detect an impact and measure the peak acceleration, or just sample continuously. Because the proper way to set up the accelerometer for impact testing was unknown, the accelerometer functions script was created to be as general as possible, allowing for quick adjustments of the settings. See [38] for all possible settings.

Figure 3.5a shows a flowchart of how the sampling main script works. For testing purposes, it was set up to sample continuously for a given time. The sampling time is defined by the output data rate of the accelerometer and the “NUMB_OF_BYTES” constant, line 24 in Appendix A, with the constant being a multiple of the output data rate. The constant must also be multiplied by 2 to correct for the fact that each sample is represented as a 16-bit value, i.e., 2 bytes. For example: If the ODR is set to 3200 Hz, a sample time of 4 seconds is achieved by setting "NUMB_OF_BYTES" to 25 600.

Processing program

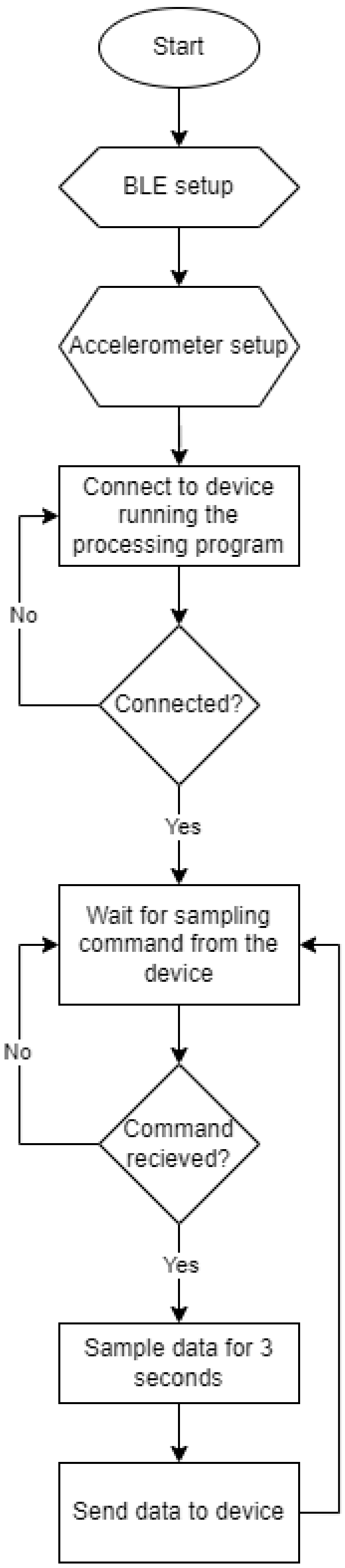
The processing program can be split into two parts. One part handles the Bluetooth connection with the Arduino, and the other part processes the received data, saves it into a CSV file and plots a graph of it. For testing purposes, the processing program was setup up to save the raw data of the accelerometer. The Arduino transmits the data as bytes, meaning the calculation into g-force is done in the processing program. Figure 3.5b shows a flowchart of how the processing program works, and it can be viewed in its entirety in Appendix D. It is important to note that the flowchart is a simplified explanation of how the program works. In addition, some parts of the code shown in Appendix D are redundant but have not been removed because of a lack of time.

3.2.3 Designing the housing

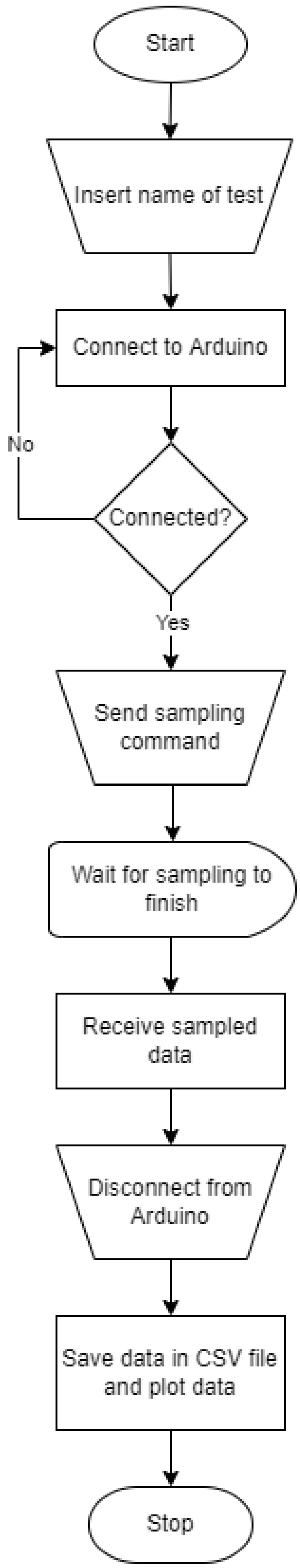
Dropping the sensor system into fresh concrete requires a robust housing to protect the electronics. It also needs to be able to handle a pH-value of 13. For the sake of simplicity, it was determined that a generic marking buoy would serve as the starting point of creating the housing, see Figure 3.6. It was split in two and polished to remove bumps on the surface. Two aluminum rings that could be screwed together were made in a CNC milling machine and glued to each part of the buoy. An O-ring was installed at the point where the aluminum rings were squeezed together to prevent fresh concrete from getting to the inside of the housing and possibly damaging the electronics. The finished housing is seen in Figure 3.7a.

3.3 Finished prototype

After the housing was complete, the sensor circuit shown in Figure 3.3 was attached to it using divinycell [43], metal thread inserts glued into holes made in the divinycell, and nylon screws. Figure 3.7b and Figure 3.8 shows the finished prototype when open. Figure 3.9 shows the prototype when closed.



(a) Sampling program flowchart

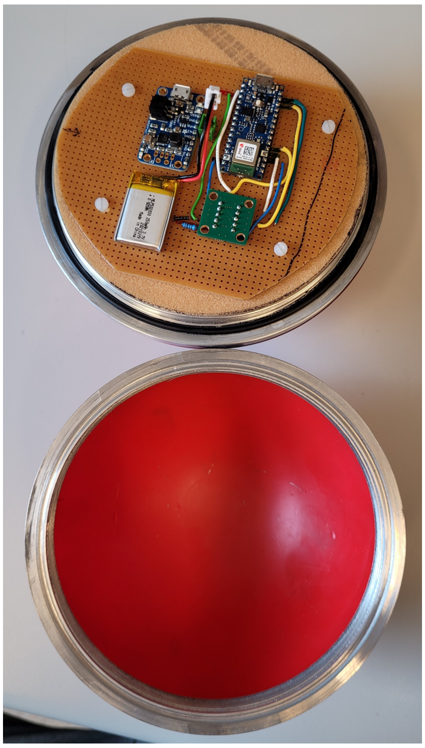


(b) Processing program flowchart

Figure 3.5: Flowcharts for both programs.

Figure 3.6: Marking buoy used as starting point for sensor housing.

(a) Picture of sensor housing.



(b) Picture of finished prototype when split.

Figure 3.7: Pictures of sensor housing and finished prototype

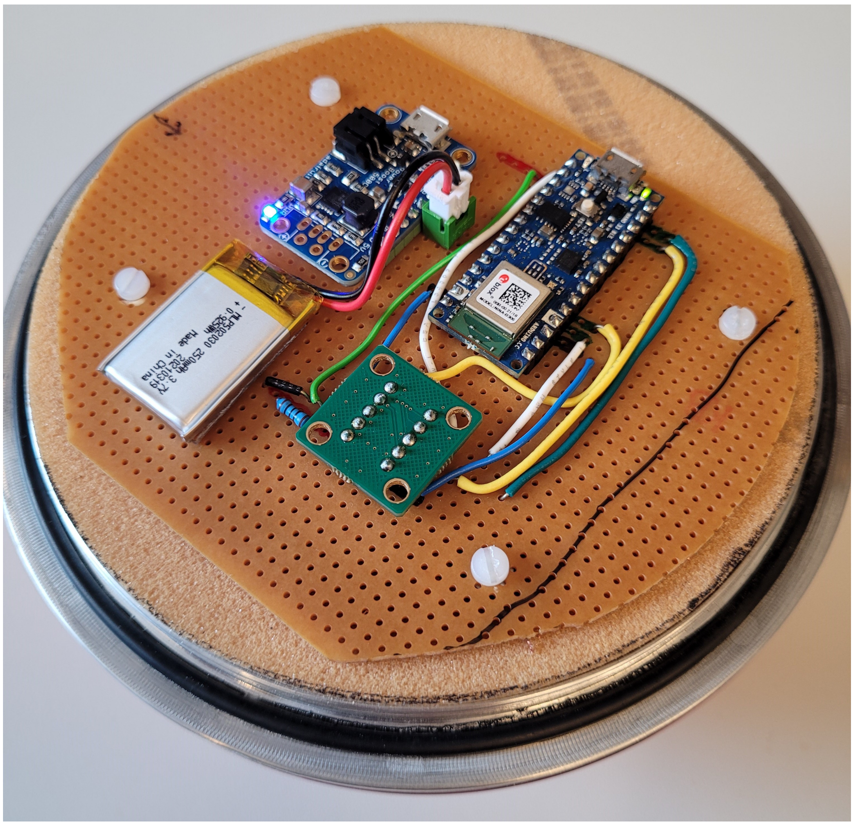


Figure 3.8: Picture of finished prototype, circuit half.

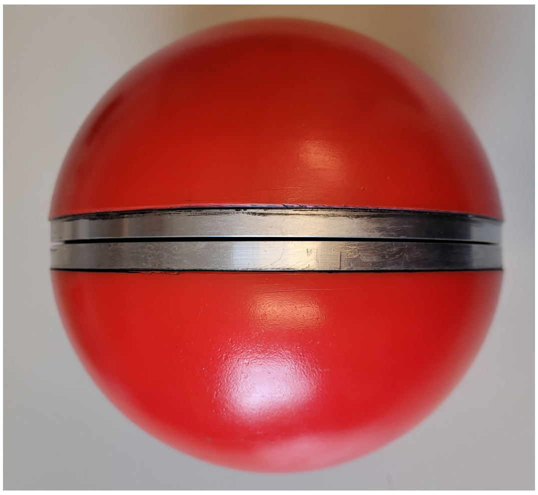


Figure 3.9: Picture of prototype when closed.

3.4 Testing

With the prototype complete, testing could commence. Because of limited time left on the project, only two days of testing took place. The location for testing was SINTEF's concrete testing laboratory, see [44]. On both days, prototype testing was performed at the same time as other testing was carried out in the laboratory. This resulted in not getting the ideal setup for testing the prototype.

3.4.1 Sensor setup

As mentioned in Section 3.2.2, the accelerometer was configured to sample continuously for a given time. More specifically, it was set to sample for 3 seconds. The settings for the accelerometer can be seen in the accelerometer setup function starting on line 171 in the main script, see Appendix A. Relevant settings can also be seen in Table 3.3. Some of the settings need to be explicitly set because of how the script is set up, but do not affect this specific type of measurement in any way. "NUMB_OF_BYTES" is set to 9600 to achieve 3 seconds sampling time with the mentioned settings.

Table 3.3: Relevant accelerometer settings for testing setup.

Setting	Value / Note
FIFO mode	Bypassed
ODR	1600 Hz
Low noise mode	On
Bandwidth	800 Hz
High pass filter corner	Corner 3 (0.99 Hz at ODR 1600 Hz)
Low pass filter	Enabled
High pass filter	Enabled
Mode of operation	Full bandwidth measurement mode

3.4.2 Testing day 1

Testing day 1 consisted mainly of general testing of the prototype, given this was the first time it was dropped into fresh concrete. Because testing of the prototype took place at the same time as other testing was happening at the lab, only seven drop tests were performed. This came mainly because of a prototype test requiring two people: One person controlling it from a nearby laptop and one person dropping it. This made doing a test possible only when one of the workers at the laboratory was available to help.

The concrete used for the tests was of such a high consistency that it was beyond the limits of the slump test, meaning the slump flow test was used instead. At first, the concrete consisted of the different ingredients mentioned in Section 2.1. Later, fiber was added into the mixture in three stages, with testing being performed between each mixing stage. An explanation of what effects fiber has on a concrete mixture is beyond the scope of this report. See [45] for information about fiber reinforced concrete. For all drop tests, the prototype was dropped directly into the mixer, see Figure 3.10. The height it was dropped from varied, but was not recorded.

3.4.3 Testing day 2

Testing day 2 consisted mainly of testing how the height the prototype was dropped from affected the measured peak acceleration of the impact. The goal was to see if

Figure 3.10: Picture of concrete mixer the prototype was dropped into during testing.

any linearity between the height and peak acceleration could be found. For this day, 14 tests were performed. Of these, the first 6 were drops directly into the mixer, see Figure 3.10, and the rest into a 20 L bucket, see Figure 3.11. Unlike the mixer drops, the bucket drops were carried out by one person. The concrete used was the same type as for testing day 1. For the drops into the mixer, the prototype was dropped from about 22 cm above the concrete, measured from the bottom of the housing. Drops into the bucket were performed at different heights. Table 3.4 shows the drop height for the bucket tests.

Table 3.4: Testing day 2 bucket drops height for the 8 last tests of the day.

Drop number	Height dropped from
7	20 cm
8	20 cm
9	30 cm
10	30 cm
11	40 cm
12	40 cm
13	40 cm
14	40 cm

Figure 3.11: Picture of 20 L bucket with fresh concrete the prototype was dropped into during testing.

Chapter 4

Results

4.1 Results from testing day 1

Testing day 1 gave no measurement that is worth presenting with graphs, but gave good insight into the weaknesses of the prototype and possible improvements that could be made to make testing go smoother. Although seven drop tests were performed, only 5 of them were successful. During test number five and test number seven the sensor system disconnected during the drop. No explanation was found for the disconnection on test number 5. For test number 7 it was discovered to be because of low power on the battery.

As mentioned in Section 3.2.1, an accelerometer with a $\pm$ 200 g measurement range was chosen to have some leeway with regards to the peak acceleration of the impact. For the developed ideal complete solution presented in Section 3.1, the drop height was thought to be about 1 m. To get an idea of the peak acceleration this would give, test number 6 was performed from a height of about 1 m, resulting in a peak acceleration on the z-axis of about $-$ 35 g.

After the testing day was complete, it was discovered that the O-ring had detached in some areas, and that concrete residue had managed to slip into the threads, destroying the aluminum slightly. This resulted in the prototype needing some repairs before more testing could be performed. Possible improvements to the prototype, the test setup, and the general execution of the tests are presented in Section 5.

4.2 Results from testing day 2

From testing day 2, only the measurements of the bucket drops are worth presenting as graphs. Results from test number 7 to 14 can be seen in Figure 4.1 to Figure 4.8. They all show the total acceleration across all axis, calculated with this equation:

$$a_{tot} = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{4.1}$$

where a_{tot} is the total acceleration, a_x is the measured acceleration on the x-axis, a_y is the measured acceleration on the y-axis, and a_z is the measured acceleration on the z-axis, all representing measurements from one sample. The x-axis on Figure 4.1 to Figure 4.8 has been cut to mostly include the impact and not the full 3 seconds of sampling. The graphs should be viewed in correlation with Table 3.4.

Getting a consistent result from drops from the same height is the first step in trying to find a possible relationship between the drop height and peak acceleration. Only tests 11, 12, 13 and 14 can be used to say anything about this given these are the only ones were several drops were taken from the same height. Test 11 and 12 shows the same measurement, but test 13 and 14 are nowhere near. Other than that, it does not appear to be a significant change in peak acceleration when the height is increased by 10 cm.

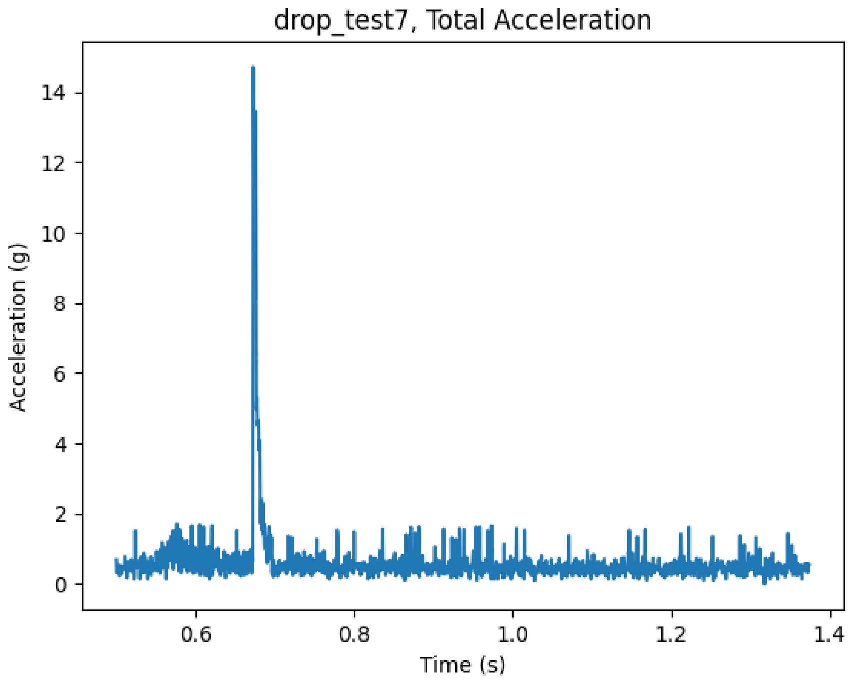


Figure 4.1: Total acceleration measured on test 7, testing day 2.

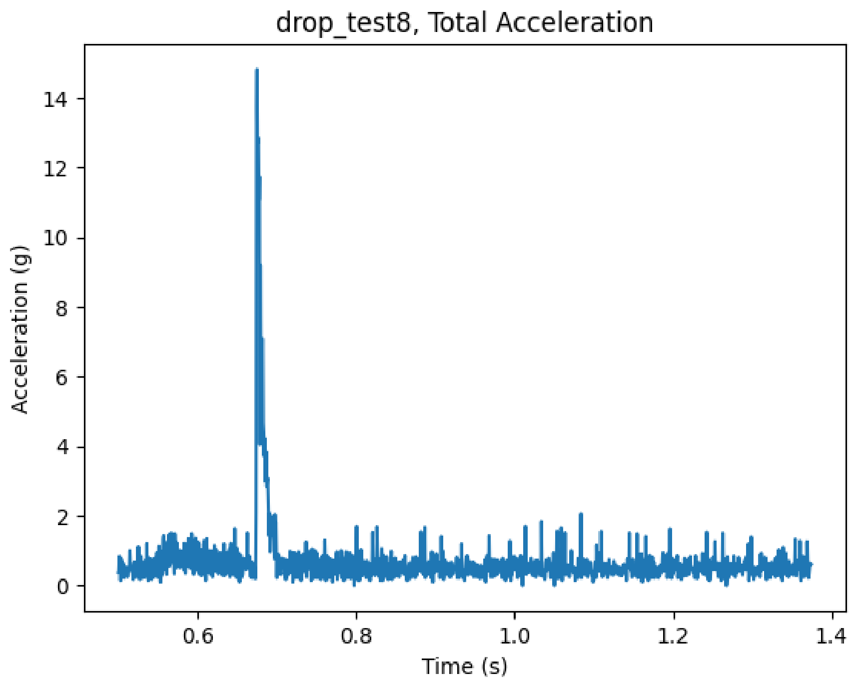


Figure 4.2: Total acceleration measured on test 8, testing day 2.

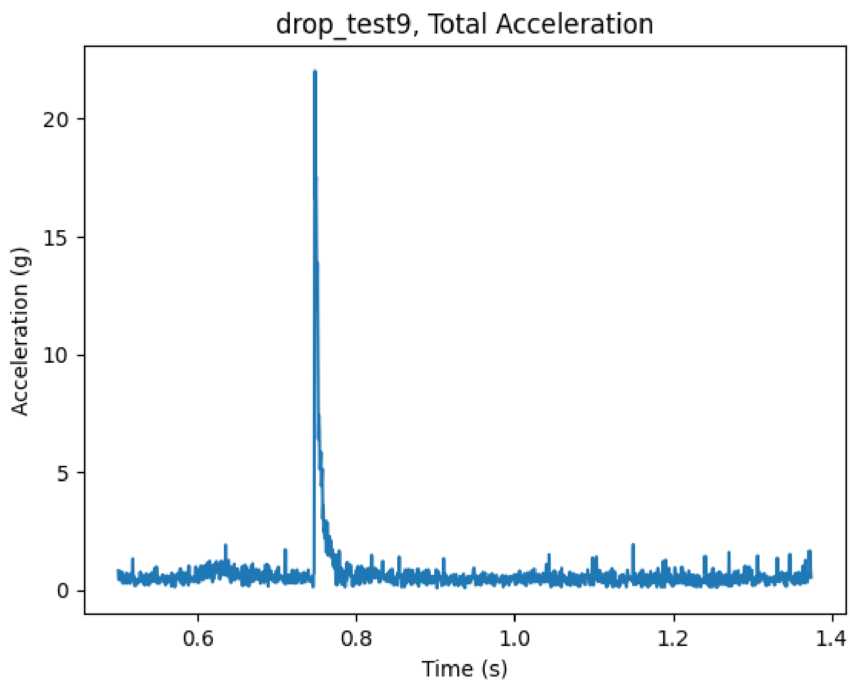


Figure 4.3: Total acceleration measured on test 9, testing day 2.

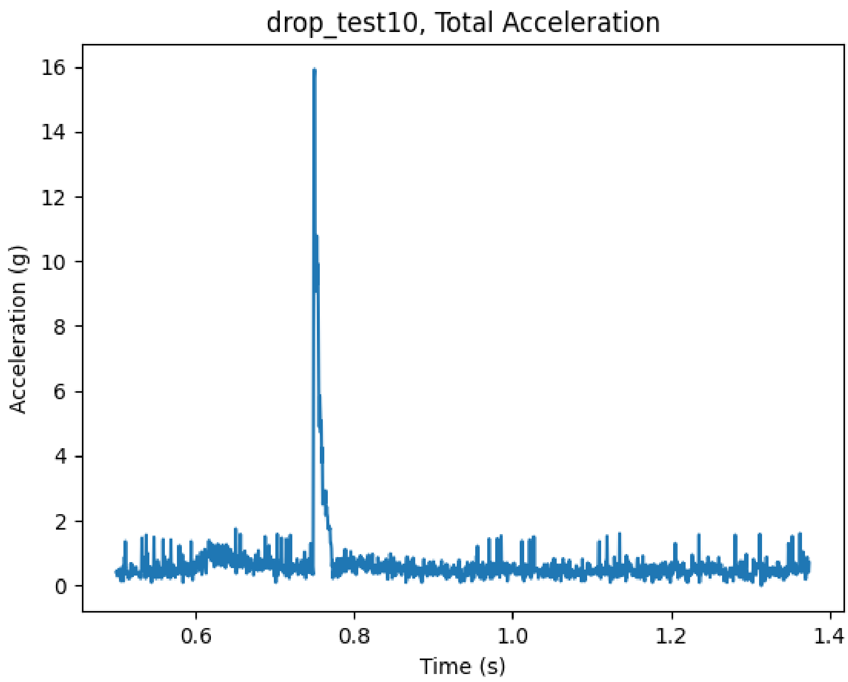


Figure 4.4: Total acceleration measured on test 10, testing day 2.

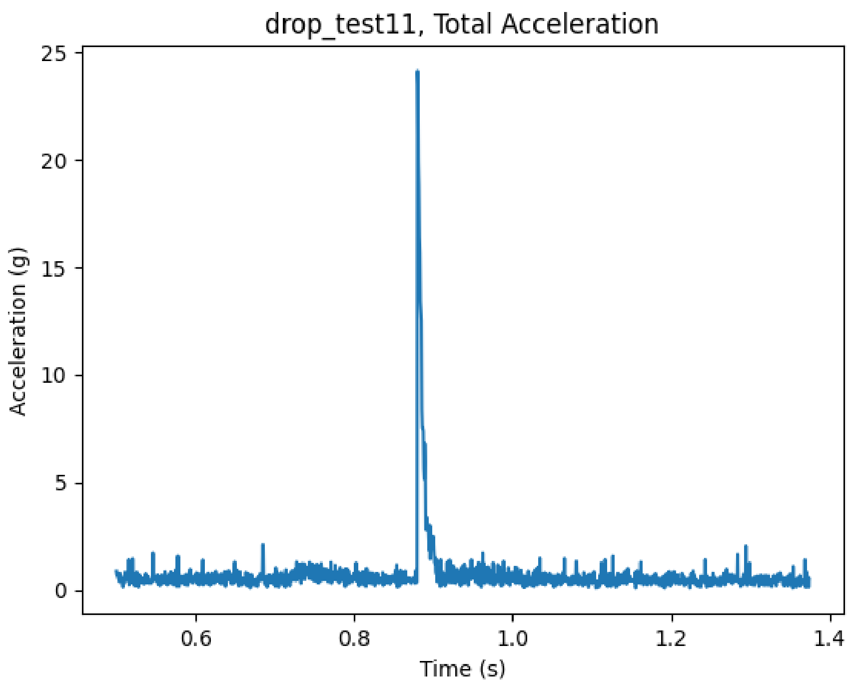


Figure 4.5: Total acceleration measured on test 11, testing day 2.

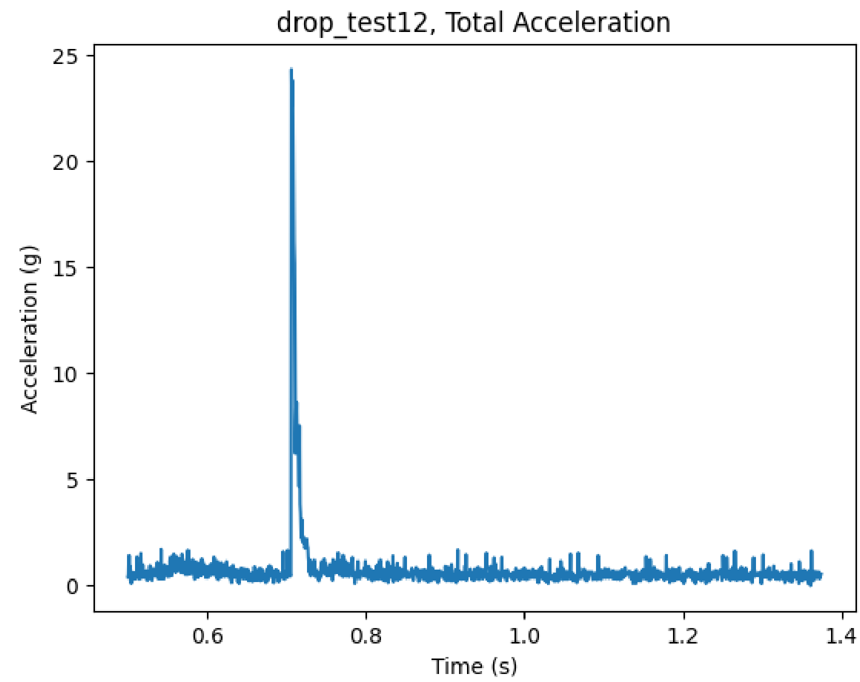


Figure 4.6: Total acceleration measured on test 12, testing day 2.

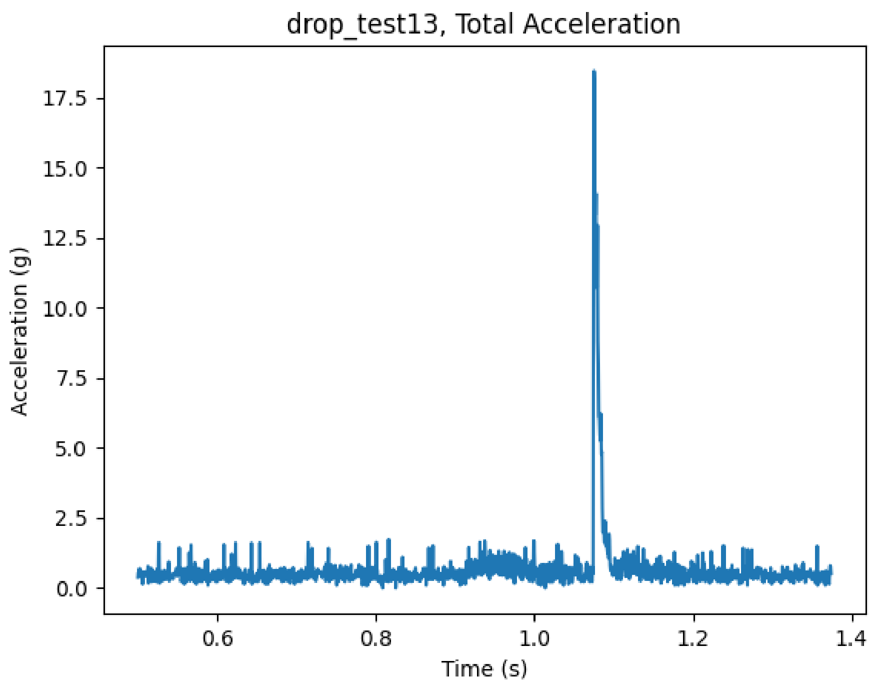


Figure 4.7: Total acceleration measured on test 13, testing day 2.

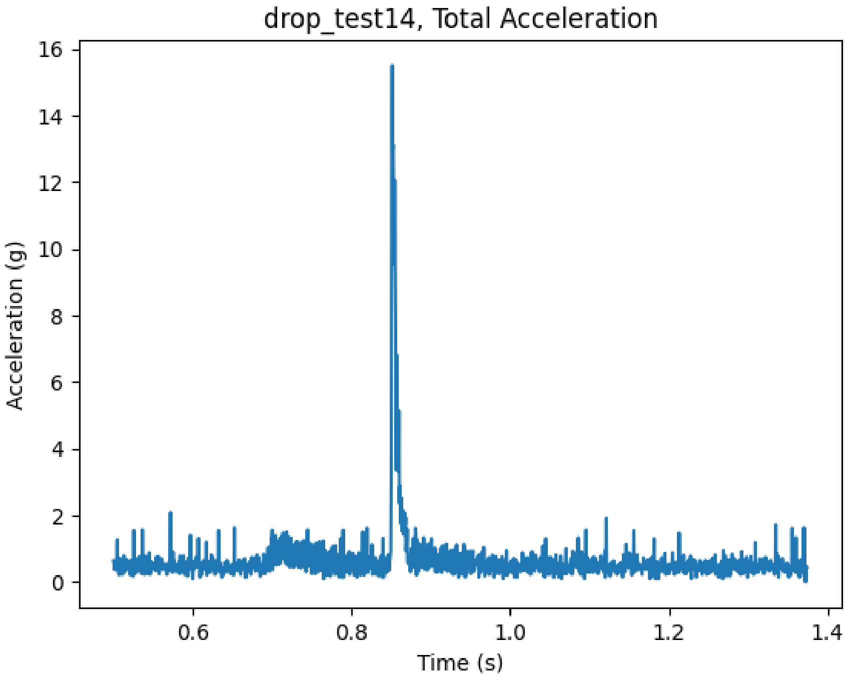


Figure 4.8: Total acceleration measured on test 14, testing day 2.

Chapter 5

Discussion

5.1 Lessons learned from testing

Making any sort of conclusion on whether this measurement concept might work or not is difficult given the few amounts of tests that were done. Testing was only possible on days when other tests took place at the concrete laboratory. Few days of this combined with national holidays made the available days of testing even more limited. However, the tests gave valuable insight into how further testing should be carried out and possible improvements that could be made to the concept and prototype. Much time at the start of the project was spent on investigating concrete and its properties to develop a concept that could work for consistency measurements. At one point, it was concluded that the only way to achieve enough knowledge to develop a working concept was to get hands on experience with concrete. This became even more evident during the testing days. The more tests that were performed, the more questions arose.

One of the questions was related to how the cohesion of the concrete mixture would affect the measurements. It is natural to think that if the sensor system were to fall in an area with a large concentration of coarse aggregates, the peak acceleration and general characteristic of the impact would be different than if it was to land in an area with few coarse aggregates. Researching this further would be required to see if it would set limitations to the concept or straight up disprove if the concept could work.

The cohesiveness of a concrete mixture is one of the things looked at when performing the slump test and slump flow test. It is a subjective qualitative perception rather than a measured variable of the concrete, meaning there is no way for the sensor system to measure it. Hypothetically, if the sensor system could measure the consistency of fresh concrete with the same accuracy as the slump test, it would not be possible to get a perception of the cohesiveness, which would be a major drawback.

Another question was: How does the way the system is dropped affect the impact? Dropping the system in the exact same way every drop was nearly impossible when the tests were carried out by just one person, i.e., the bucket tests. The drops into the mixer were not observed properly to see how similar the way of dropping was for each drop. The question here is related to what can only be described as a lazy drop, where the ball slides out of your fingers, or a sudden drop, where the ball is instantaneously let go from your hands. It is thinkable that a lazy drop might slow down the acceleration of the ball, making it have a lower speed when it hits the concrete than with a sudden drop, ultimately changing the characteristics of the impact.

Looking at the physics behind the force of the impact, we know that there are two variables that we are in control of which would affect the peak acceleration. These are the height the sensor system is dropped from and the weight of the system, both of which there was no existing data for how heavily they would affect the peak acceleration. The idea was to identify this through testing.

Dropping the system from different heights is easily done. The weight of the system, on the other hand, would require alterations to the housing. Combined, these two variables would decide the required measurement range on the accelerometer. Seeing the $-$ 35 g from the test when the system was dropped from about 1 m gives the impression that $\pm$ 200 g is more than needed for a system of this weight. The complete prototype weighed in at 737 grams. This means that an accelerometer with a lower range would probably be a better option, given the weight is not altered. A lower range will in most cases mean a better resolution, making it easier to see small differences in the measured acceleration.

During testing on day number 2 it was realized that consistent measurements for drops from the same height is the first step of testing the measurement concept. Without consistent results from tests with the same drop height in concrete of one specific consistency, there could be no possible way of measuring a difference in the consistency of fresh concrete. The expectation beforehand was that consistent results would occur, but after several drop tests from the same height it was rejected. Results from the four drops from 40 cm, see Figure 4.5 to Figure 4.8, showed almost the same peak acceleration for the first two drops, but the last two were far off. As mentioned previously, the way the system was dropped might have affected the measurements.

Another aspect of uncertainty that might have affected the measurements is the fact that if the concrete is not continuously in motion, it starts hardening. In fact, the reason for no additional tests on day two was that the concrete became too stiff. Hardened concrete essentially affects the consistency. It was expected that the peak acceleration would increase as the concrete hardened, which was the opposite of what happened with the two last bucket drops from 40 cm.

An important note from the bucket drops is that for the later drops the concrete was stirred slightly to keep it from hardening. Which drop this was started at was not noted down. The possibility that a change in concentration of aggregates took place between the drops from the same height is highly present, which might have caused the difference in peak acceleration. As this is neither measurable nor possible to observe, it makes it impossible to test if it has any impact on the measurements.

5.2 Test setup

Controlling all uncertainty aspects related to the tests would be a must to test the measurement concept thoroughly. A contraption that would ensure the sensor system was dropped in the same way every test would exclude any impact imprecise human hands might have on the measurements. Such a contraption could also give an exact measure of the height the system was dropped from. Dropping the ball into the mixer instead of in a bucket of concrete would make it possible to keep the concrete from hardening. This would allow for several tests to be performed on a batch without the consistency changing and affecting the measurements. However, this would still not make it possible to have a sense for the cohesion of the mixture or control if the sensor system was dropped into an area with a relatively high or low concentration of coarse aggregates. A possible workaround for the latter would be to do drops in several places in the mixer instead of just in one place.

Using a phone application to control the sensor system and save the test data would be preferable over bringing a laptop to the lab. This is because the concrete laboratory is a harsh environment for a computer. If more testing was to be carried out then either a phone application should be developed or a laptop must be set up in a proper place and used by a person who does not handle the concrete.

5.3 Analyzing the results

Because of a lack of time, the results from the tests were not properly analyzed. Only the peak acceleration was looked at. The first step in analyzing the data would be to filter it. The results from the bucket drops, see Figure 4.1 to Figure 4.8, have a lot of noise. Processing the data in Python was investigated, giving the impression that creating a Python script for filtering the data would take a lot of time. Finding software designed for analyzing impact measurements was considered as an alternative. A program called "VibrationData Toolbox", which is specified as a signal analysis and structural dynamics software, looked promising at first glance, albeit no extensive investigation into the software's capabilities was performed. The mentioned software can be checked out at [46].

5.4 Housing

The housing was designed to be as simple as possible to create. As long as it served its purpose of protecting the electronics, it was good enough. Of possible improvements that could be made to the housing, the notable part would be a possibility to alter the weight of the system. If a new prototype were to be built with this in mind, it would have to be larger. Making a housing out of metal instead of plastic is an alternative. It would increase weight but might cause the system to be unable to connect with an external device over Bluetooth.

5.5 Software

The software was developed to make the system ready for testing. For a finished solution the code would need to undergo substantial changes, especially the processing program. As it is set up right now, the sampled data from the accelerometer is saved on the Arduino before transferring to the PC. This means that with the current settings, a total of 28.8 kB of data is saved on the Arduino. This was only possible because the Nano has 256 kB of RAM. The full 2 Mbit speed advertised for the Bluetooth was not achieved, for unknown reasons. Continuous transfer was attempted at first, but tests showed a loss of about 30% of the data. The indicate mode was chosen instead of the standard read mode to ensure all data would come through. A simplified explanation of the modes is that the indicate mode is where the receiver must acknowledge that it has read the package before the transmitter sends a new one, while the read mode is where the transmitter sends out data as fast as possible and the receiver must try and catch it before a new package is transmitted.

Transfer speeds went down to about 7 to 8 transfers a second in indicate mode. Because of this the decision was made to transfer the data after the sample time was complete and not continuously while sampling. An investigation was done into why the transfer speed was so slow and possible ways to improve it, to no prevail. Suggestions were found that there could be possible limitations in the Arduino BLE library, but this has not been confirmed. All functions for the Bluetooth setup in the sampling program were from this library.

Transmitting the data to the PC with indicate mode took up to twenty seconds. If the sampling speed or sampling time were to be increased, the transmission time would be even longer. The amount of RAM on the Arduino also limits the amount of data that can be sampled with the current setup.

Sending the accelerometer data without processing was decided as the best solution because of limitations in the Arduino BLE library. This was also the only way to achieve a transmission time that was not unpractically long and to make it possible to separate the different samples from each other. To expand on the latter: The data in a Bluetooth package is represented as a byte-array. With a maximum of 244 bytes of data in 1 package, a total of 122 samples can be sent per package. Each sample is a 12-bit value but needs to be represented as a 16-bit value. As 122 samples are sent as a neatly packed byte-array, which are all signed values, the processing program needs to separate the samples into 1 and 1 byte and then handle the MSBs and LSBs of the values individually to get the sign correct. To clarify: The values from the accelerometer are represented as two's complement values. The MSBs and LSBs are then combined and converted to g-force.

The biggest drawback with the current processing program is that it must disconnect from the sensor system to save data in a CSV file. Preferably, the sensor would send a keyword indicating all data has been transmitted so that the processing program could save it to a CSV file and prepare for a new test without disconnecting. Connecting to the sensor system took up to twenty seconds. Without disconnecting between tests, more tests could be performed in a shorter time.

5.6 Chosen components

Questions can be asked about whether the components chosen for the prototype were the best-suited ones. The accelerometer showed noise above what was specified as typical noise performance in the datasheet, see [38]. Possible causes for this are many, and no investigation was done to identify exactly what caused it. It is suspected that the main cause for the noise was the power supply from the Nano, but this needs to be examined. Other than the noise, there were no significant drawbacks to the ADXL372. A vast number of operation modes and settings gives great freedom to the user. However, it is best suited for impact detection. I.e., applications with focus on the peak acceleration. For detailed characterization of an impact, an accelerometer with a higher resolution is suggested. If a new accelerometer was to be used for further testing of the measurement concept, it would need to be able to capture the peak acceleration and have a high sampling rate, allowing for all fine details of the impact characteristic.

Using an Arduino Nano 33 BLE was initially decided on to simplify the programming work. In hindsight, it might have made it too simple, but nothing can be said confidently without diving into how the Arduino BLE library functions work. If the issues regarding the transfer speed were known when choosing a microcontroller board, more research would have gone into it before deciding. It was lucky the Nano had enough RAM to store many samples from the accelerometer so that it could transfer all of them after the sampling time had finished, as this was not considered when choosing a microcontroller board.

5.7 Taking inspiration from the Kelly ball test

When attempting to find inspiration for developing the measurement concept, the Kelly ball test was discovered, see Section 2.3.3. This test supported the idea of measuring the consistency by dropping something into the concrete. As the Kelly ball test is not used here in Norway, it was impossible to observe it in practice. An idea for further development of the measurement concept for the sensor system is to imitate the Kelly ball test. One could create a sensor system that weighed the same as the Kelly ball and had an accelerometer with higher resolution and lower measurement range, which would base the consistency on the characteristic of the system sinking into the concrete. This idea has not been appropriately researched and is merely an idea presented for possible further work on the project.

5.8 Source criticism

All sources used to support the claim of low productivity in the construction industry compared to other industries, as presented in the background for the master thesis in Section 1.1, are from 2016 – 2018. No newer articles were found about this subject, albeit not a substantial amount of time went into researching this. The important thing to note is that things have most likely changed over the past couple of years, but in which direction is unknown as no newer studies were found. The study from Statistics Norway also showed severe uncertainties connected to the calculations, related to how prizes were calculated and the definition of “construction industry”. To expand on the latter: Results showed an increase in productivity when the definition for “construction industry” was expanded to include all business areas connected to the work at a construction site; not a decrease. These discoveries are relevant to this thesis in the way that they raise questions about the reason and background for the project itself.

5.9 General reflections on the project

Most of the total four semesters on this project were spent learning about concrete and developing a measurement concept. Concrete is such a complex material that identifying and understanding its relevant and important characteristics would be impossible in the limited period of the project. Trying to limit it to only learning the parts that were needed for developing a measurement concept was impossible because in a way everything was relevant. At a certain point, further investigation into the theory of concrete was halted in favor of trying to develop something that could be the basis of a master’s thesis. The developed measurement concept was presented to experts on concrete and the feedback was positive. This was the deciding factor as to why this exact measurement concept was chosen to investigate further by developing a prototype and testing.

In hindsight, I should have been more effective with the literature study and research into the theory of concrete. Getting to the prototype development and testing part took much longer than it should have. The blame mainly falls on inadequate and unsystematic work methods, but some blame can be pointed to the difficulty in finding and getting in contact with people that could help. The COVID-19 pandemic is responsible for the latter because the university was in full or partial lockdown for much of the 1st year worked on the project.

Chapter 6

Conclusion

Results from the few tests that were performed is little to go by when trying to conclude if the developed measurement concept will work as an equivalent to the slump test. Without getting consistent results from identical drops, there is not any possibility of getting measurements that can be correlated with slump tests measurements. If further testing is to be carried out, alterations must be made to the testing setup to exclude any influence uncertainties might have on the results. In addition, if the measurement concept was to be tested to the fullest, one would have to perform tests with all possible types of concrete mixtures to see what effect they would have. If further testing is performed, and the results does not meet the prerequisites for the system, the measurement concept should be abandoned.

Creating the prototype took longer than expected, but managed to serve its purpose as a device created for testing the measurement concept, although it is far from ideal in its functionality. Making changes to the software will help make testing go smoother and is easy to do. Creating a different housing with capabilities of adjusting the weight of the system would take time, but is highly doable. Although a new circuit must be constructed if the existing one cannot be inserted into it. Creating a circuit with different parts would require a rework of both the housing and the software. Abandoning the developed measurement concept for a new one would require a new prototype to be made.

Working on a project like this have given me valuable knowledge in concept development. Given the experience gained, I am certain that if the project had been started on today, getting to the prototype and testing part would go substantially faster.

For possible further work, I recommend running more tests with identical drops to see if consistent results can be achieved. If consistent results are achieved, the next step will be to plan for testing of the sensor system in correlation with the slump and slump flow test. Many of the ideas and thoughts presented in Chapter 5 can also be researched further.

Bibliography

[1] *Statistisk sentralbyrå*. URL: https://www.ssb.no/. (accessed May 12, 2022).

[2] *Produktivitetsfall i bygg og anlegg*. URL: https://www.ssb.no/bygg-bolig-og-eiendom/artikler-og-publikasjoner/produktivitsfall-i-bygg-og-anlegg. (accessed May 12, 2022).

[3] S. Chandrasekaran R. Agarwal and M. Sridhar. *Imagining construction's digital future*. URL: https://www.mckinsey.com/business-functions/operations/our-insights/imagining-constructions-digital-future. (accessed May 12, 2022).

[4] J. Koeleman, M. J. Ribeirinho, D. Rockhill, E. Sjödin and G. Strube. *Decoding digital transformation in construction*. URL: https://www.mckinsey.com/business-functions/operations/our-insights/decoding-digital-transformation-in-construction. (accessed May 12, 2022).

[5] F. Barbosa et al. *Reinventing construction: A route to higher productivity*. URL: https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/Operations/Our%20Insights/Reinventing%20construction%20through%20a%20productivity%20revolution/MGI-Reinventing-construction-A-route-to-higher-productivity-Full-report.pdf. (accessed May 12, 2022).

[6] *Digitalisering*. URL: https://www.bnl.no/politikk/politiske-saker/digitalisering/. (accessed May 12, 2022).

[7] Y. M. Anton. *Introduksjon*. URL: https://www.sintef.no/projectweb/sitecast/. (accessed May 12, 2022).

[8] M. F. Mogos. *SiteCast*. URL: https://www.sintef.no/prosjekter/2018/sitecast/. (accessed May 12, 2022).

[9] Y. M. Anton. *Samarbeider for å gjøre plasstøping mer lønnsomt*. URL: https://www.sintef.no/siste-nytt/2019/samarbeider-for-a-gjore-plasstoping-mer-lonnsomt/. (accessed May 12, 2022).

[10] *About SINTEF - Applied research, technology and innovation*. URL: https://www.sintef.no/en/sintef-group/this-is-sintef/. (accessed Jun. 8, 2022).

[11] *SINTEF research institutes*. URL: https://www.sintef.no/en/sintef-research-institutes/. (accessed Jun. 8, 2022).

[12] *Gunrid Kjellmark*. URL: https://www.sintef.no/en/all-employees/employee/gunrid.kjellmark/. (accessed Jun. 8, 2022).

[13] *Om Unicon*. URL: https://www.unicon.no/om-unicon/unicon-norge/. (accessed Jun. 8, 2022).

[14] *Concrete*. URL: https://en.wikipedia.org/wiki/Concrete#Composition. (accessed May 4, 2022).

[15] *Scientific Principles*. URL: http://matse1.matse.illinois.edu/concrete/prin.html. (accessed May 4, 2022).

[16] C. R. Gagg. 'Cement and concrete as an engineering material: An historic appraisal and case study analysis'. In: *Engineering Failure Analysis* 40 (May 2014), pp. 114–140. DOI: https://doi.org/10.1016/j.engfailanal.2014.02.004.

[17] R. Rajapakse. 'Chapter 3 - Concrete Construction'. In: *Construction Engineering Design Calculations and Rules of Thumb*. Butterworth-Heinemann, 2017. Chap. 3, pp. 15–70. DOI: https://doi.org/10.1016/B978-0-12-809244-6.00003-2.

[18] M. I. Hamakareem. *Compaction of concrete - methods and results of improper vibration of concrete*. URL: https://theconstructor.org/concrete/compaction-of-concrete-methods/14028/. (accessed May 4, 2022).

[19] E. P. Koehler and D. W. Fowler. *Summary of concrete workability test methods*. Aug. 2003.

[20] *Compaction of Concrete*. URL: https://www.boral.com.au/sites/default/files/media/field_document/Compaction%20of%20Concrete.pdf. (accessed May 4, 2022).

[21] A. M. Neville. 'Fresh concrete'. In: *Properties of concrete*. Vol. 5. Pearson, 2012. Chap. 4, pp. 186–203. ISBN: 978-0-273-75580-7.

[22] T. A. Fedorovich and A. Drozdov. 'Automatic control of the concrete mixture homogeneity in cycling mixers'. In: vol. 317. Mar. 2018. DOI: 10.1088/1757-899X/317/1/012043.

[23] G. Mishra. *Workability of concrete - Types and effects on concrete strength*. URL: https://theconstructor.org/concrete/workability-of-concrete-types-strength/11739/. (accessed May 4, 2022).

[24] *What is workability of concrete? Types, mechanism*. URL: https://civiltoday.com/civil-engineering-materials/concrete/93-workability-of-concrete-definition-types-details. (accessed May 4, 2022).

[25] *Concrete slump test*. URL: https://en.wikipedia.org/wiki/Concrete_slump_test#cite_note-Gambhir-1. (accessed May 4, 2022).

[26] *What is concrete slump test? step-by-step procedure*. URL: https://civiltoday.com/civil-engineering-materials/concrete/79-concrete-slump-test-standard-equipment-procedures-cautions. (accessed May 4, 2022).

[27] G. Mishra. *Different properties of fresh concrete for construction works*. URL: https://theconstructor.org/concrete/properties-of-fresh-concrete/6490/. (accessed May 4, 2022).

[28] *Testing fresh concrete - Part 1: Sampling and common apparatus. NS-EN 12350-1*. 2019.

[29] *Testing fresh concrete - Part 2: Slump test. NS-EN 12350-2*. 2019.

[30] *Concrete — Specification, performance, production and conformity. NS-EN 206*. 2013.

[31] SINTEF. 'Kvalitetskontroll av fersk betong'. In: *Byggforskserien* (May 2015). ISSN: 2387-6328. Art. no. 520.027.

[32] *Concrete sensors.* URL: https://www.hilti.com/content/hilti/W1/US/en/business/business/trends/concrete-sensors.html. (accessed May 30, 2022).

[33] A. R. Alizadeh. *The best concrete sensor in 2020.* URL: https://www.giatecscientific.com/education/the-best-concrete-sensors-2020/. (accessed May 30, 2022).

[34] *nRF52840. Product specification.* URL: https://content.arduino.cc/assets/Nano_BLE_MCU-nRF52840_PS_v1.1.pdf. (accessed Jun. 1, 2022).

[35] *Arduino Nano 33 BLE. Product reference manual.* URL: https://docs.arduino.cc/resources/datasheets/ABX00030-datasheet.pdf. (accessed Jun. 1, 2022).

[36] *Arduino Nano 33 BLE.* URL: https://content.arduino.cc/assets/Pinout-NANOble_latest.pdf. (accessed Jun. 1, 2022).

[37] *Arduino Nano 33 BLE.* URL: https://store.arduino.cc/products/arduino-nano-33-ble. (accessed Jun. 1, 2022).

[38] *ADXL372.* URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL372.pdf. (accessed Jun. 1, 2022).

[39] *EVAL-ADXL372Z user guide.* URL: https://www.analog.com/media/en/technical-documentation/user-guides/EVAL-ADXL372Z-UG-1113.pdf. (accessed Jun. 1, 2022).

[40] lady ada. *Adafruit PowerBoost 500 + Charger. Overview.* URL: https://learn.adafruit.com/adafruit-powerboost-500-plus-charger. (accessed Jun. 1, 2022).

[41] lady ada. *Adafruit PowerBoost 500 + Charger. Downloads.* URL: https://learn.adafruit.com/adafruit-powerboost-500-plus-charger/downloads. (accessed Jun. 1, 2022).

[42] lady ada. *Adafruit PowerBoost 500 + Charger. Pinouts.* URL: https://learn.adafruit.com/adafruit-powerboost-500-plus-charger/pinouts. (accessed Jun. 1, 2022).

[43] *Divinycell - PVC core materials.* URL: https://www.diabgroup.com/products-services/divinycell-pvc/. (accessed Jun. 7, 2022).

[44] *Betonglaboratorier.* URL: https://www.sintef.no/laboratorier/betonglaboratorier/. (accessed Jun. 4, 2022).

[45] G. Mishra. *Fiber reinforced concrete - types, properties and advantages of fiber reinforced concrete.* URL: https://theconstructor.org/concrete/fiber-reinforced-concrete/150/. (accessed Jun. 5, 2022).

[46] *VibrationData toolbox.* URL: https://endaq.com/pages/vibration-shock-analysis-software-vibrationdata-toolbox?__hssc=9408618.2.1594775292510&__hstc=9408618.446179f56627e5f366b7be3b5441ab4a.1566482603586.1594750104586.1594775292510.397&__hsfp=569979075&hsCtaTracking=d7e8af39-c2c6-45ce-9352-05992fa2b87d%7Cd0292227-6a7b-49a7-8ef0-eb5e8e1b9158. (accessed Jun. 7, 2022).

Appendix A

Sampling program - Main script

```
/**
*******************************************************************************
    Main program for handling sampling of data on an ADXL372 accelerometer
    and sending the data via Bluetooth to a PC where it is to be analyzed and
    processed.
    @file Drop_Test.ino
    @author T. Thorvaldsen
    @version 1.1 2022-05-06 (yyyy-mm-dd)
  @par Revision History:
  - Version 1.0, April 2022: Inital version.
  - Version 1.1, May 2022: Removed redundant code lines.
*******************************************************************************
*/

/****************************** Include Files ******************************/
#include <ArduinoBLE.h>
#include <SPI.h>
#include "ADXL372.h"

/********************************* Constants ********************************/

// General constants //
const int BUFFER_SIZE = 244;
const int NUMB_OF_BYTES = 9600;
const int NUMB_OF_TRANSMISSIONS = (NUMB_OF_BYTES / BUFFER_SIZE) +
                                  (NUMB_OF_BYTES % BUFFER_SIZE != 0);
const bool FIXED_LENGTH = false;
const char KEYWORD[] = "run";

// Bluetooth UUIDs //
const char*const service_UUID = "6e8ec787-7016-4312-9ce7-a1976b8c5c24";
const char*const x_data_char_UUID = "1d3478a3-59db-4f59-8c27-a00a3a2a7a8e";
const char*const y_data_char_UUID = "08e2eead-cc49-45f8-a24a-c0118ce005a2";
const char*const z_data_char_UUID = "3fab2d16-53e2-46e8-bbe2-b171f12f5564";
const char*const read_char_UUID = "a23be471-deb8-4fa2-8d22-10537858bbf3";

/**************************** Global Variables *****************************/

// Flags used to control sampling and packing of data //
bool sample_data = false;
bool data_packed = false;
```

```
bool keyword_recieved = true;

// Array used for transmitting 244 bytes of data per transmission //
uint8_t x_data_send_buffer[BUFFER_SIZE];
uint8_t y_data_send_buffer[BUFFER_SIZE];
uint8_t z_data_send_buffer[BUFFER_SIZE];

// Arrays used to store sampled data //
uint8_t x_data_bytes[NUMB_OF_BYTES];
uint8_t y_data_bytes[NUMB_OF_BYTES];
uint8_t z_data_bytes[NUMB_OF_BYTES];

// Structure type used to store one data sample //
AccelerBytes_t accel_data;

/*************************** Bluetooth Attributes ***************************/

// BLE Service //
BLEService testService(service_UUID);

/** BLE x-data, y-data and z-data characteristics with custom 128-bit UUIDs.
    Read and indicate properties available to central. Size of characteristic
    buffer is 244 bytes, but buffer size can vary.
*/
BLECharacteristic x_data_char(x_data_char_UUID, BLERead | BLEIndicate,
                              BUFFER_SIZE, FIXED_LENGTH);
BLECharacteristic y_data_char(y_data_char_UUID, BLERead | BLEIndicate,
                              BUFFER_SIZE, FIXED_LENGTH);
BLECharacteristic z_data_char(z_data_char_UUID, BLERead | BLEIndicate,
                              BUFFER_SIZE, FIXED_LENGTH);

/** BLE read characteristic with custom 128-bit UUID. Write and write without
    response properties available to central. Size of characteristic buffer
    is 244 bytes, but buffer size can vary.
*/
BLECharacteristic read_char(read_char_UUID, BLEWriteWithoutResponse | BLEWrite,
                            BUFFER_SIZE, FIXED_LENGTH);

/********************************* Functions ********************************/

/**
  Packs the sampled accelerometer data into arrays.
  @param x_msb_val - The 8 most significant bits of the x-axis value.
  @param x_lsb_val - The 4 least significant bits of the x-axis value.
  @param y_msb_val - The 8 most significant bits of the y-axis value.
  @param y_lsb_val - The 4 least significant bits of the y-axis value.
  @param z_msb_val - The 8 most significant bits of the z-axis value.
  @param z_lsb_val - The 4 least significant bits of the z-axis value.
*/
void pack_Data(uint8_t x_msb_val, uint8_t x_lsb_val, uint8_t y_msb_val,
               uint8_t y_lsb_val, uint8_t z_msb_val, uint8_t z_lsb_val){
  static unsigned int bytes_counter = 0;

  x_data_bytes[bytes_counter] = x_msb_val;
  x_data_bytes[bytes_counter + 1] = x_lsb_val;
  y_data_bytes[bytes_counter] = y_msb_val;
  y_data_bytes[bytes_counter + 1] = y_lsb_val;
  z_data_bytes[bytes_counter] = z_msb_val;
```

```
  z_data_bytes[bytes_counter + 1] = z_lsb_val;

  bytes_counter += 2;

  // Check if sampling is completed.
  if(bytes_counter == NUMB_OF_BYTES){
    sample_data = false;
    data_packed = true;
    bytes_counter  = 0;
  }
}

/**
  Callback function for Bluetooth event handler that checks if the
  characteristic has been written too and if the written value matches the
  keyword that indicates that sampling of data should be carried out.
  @param central - The connected bluetooth central.
  @param characteristic - The bluetooth characteristics.
*/
void read_Value_Updated(BLEDevice central, BLECharacteristic characteristic){
  static byte rx_buffer[BUFFER_SIZE];
  static int data_length = read_char.readValue(rx_buffer, BUFFER_SIZE);

  for(int i = 0; i < 3; i++){
    if(rx_buffer[i] != KEYWORD[i]){
      keyword_recieved = false;
      break;
    }
  }

  if(keyword_recieved){
    sample_data = true;
  }
}

/**
  Setup for Bluetooth.
*/
void init_BLE(void){
  // Start initialization of Bluetooth LE.
  if(!BLE.begin()){
    while(1); // If initialization fails, run forever.
  }

  // Set advertised local name.
  BLE.setLocalName("Drop Test Device");

  // Set Advertised service UUID.
  BLE.setAdvertisedService(testService);

  // Add the characteristics to the service:
  testService.addCharacteristic(x_data_char);
  testService.addCharacteristic(y_data_char);
  testService.addCharacteristic(z_data_char);
  testService.addCharacteristic(read_char);

  // Add service to device.
  BLE.addService(testService);
```

```

  // Initialize callback function for write event on read characteristic.
  read_char.setEventHandler(BLEWritten, read_Value_Updated);

  // Start advertising.
  BLE.advertise();
}

/**
  Setup for ADXL372 accelerometer.
  See datasheet for proper understanding:
  @see(https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL372.pdf)
*/
void accel_setup(void){
  set_Pwr_Ctrl_Reg(LOW_THRESH, SETTLE_TIME_370ms, LPF_DISABLE,
                   HPF_DISABLE, STANDBY);

  set_FIFO(XYZ_FIFO, BYPASSED);

  set_Timing_Ctrl_Reg(ODR_1600Hz, WUR_52ms);

  set_Measurement_Ctrl_Reg(AUTOSLEEP_OFF, DEFAULT_MODE, LOW_NOISE, BW_800Hz);

  set_High_Pass(CORNER_3);

  set_Pwr_Ctrl_Reg(LOW_THRESH, SETTLE_TIME_370ms, LPF_ENABLE,
                   HPF_ENABLE, FULL_BW);
}

/**
  Main setup.
*/
void setup() {
  // Initialize Serial communication.
  Serial.begin(9600);

  // Initialize SPI bus.
  SPI.begin();
  SPI.beginTransaction(SPISettings(10000000, MSBFIRST, SPI_MODE0));

  // Run setup function for Bluetooth LE.
  init_BLE();

  // Set blue LED in output mode.
  pinMode(LEDB, OUTPUT);

  // Set pin 2 in output mode.
  pinMode(CS_PIN, OUTPUT);

  // Set inital value for the blue LED and pin 2:
  digitalWrite(LEDB, HIGH);
  digitalWrite(CS_PIN, HIGH);

  // Run get device ID function to check if accelerometer is functioning.
  get_Device_ID();

  // Run accelerometer setup function.
  accel_setup();
```

```
}

/**
  Main loop
*/
void loop() {
  // Variable used to store status register value.
  static byte status_reg;

  // Listen for Bluetooth LE peripherals to connect.
  BLEDevice central = BLE.central();

  // If a central is connected.
  if(central){
    // Turn on the ble LED.
    digitalWrite(LEDB, LOW);

    // While the central is connected to the peripheral:
    while(central.connected()){
      // If the sample data flag = true, sampling should commence.
      if(sample_data){
        // Check value of status register.
        status_reg = check_Status();

        // If data ready bit in status register is 1, save and pack data:
        if((status_reg &= 0b00000001) == 1){
          get_Data(&accel_data);

          pack_Data((uint8_t)accel_data.x_msb, (uint8_t)accel_data.x_lsb,
                    (uint8_t)accel_data.y_msb, (uint8_t)accel_data.y_lsb,
                    (uint8_t)accel_data.z_msb, (uint8_t)accel_data.z_lsb);
        }
      }

      // If sampling complete and data packed, send data to central:
      if(data_packed){
        int send_counter = 0;
        int correction_val = 0;

        for(int i = 0; i < NUMB_OF_TRANSMISSIONS; i++){
          for(int j = 0; j < BUFFER_SIZE; j++){
            x_data_send_buffer[j] = x_data_bytes[j + send_counter];
            y_data_send_buffer[j] = y_data_bytes[j + send_counter];
            z_data_send_buffer[j] = z_data_bytes[j + send_counter];
          }

          if(NUMB_OF_BYTES - send_counter < BUFFER_SIZE){
            correction_val = NUMB_OF_BYTES - send_counter;
            x_data_char.writeValue(x_data_send_buffer, correction_val);
            y_data_char.writeValue(y_data_send_buffer, correction_val);
            z_data_char.writeValue(z_data_send_buffer, correction_val);
          }
          else{
            x_data_char.writeValue(x_data_send_buffer, BUFFER_SIZE);
            y_data_char.writeValue(y_data_send_buffer, BUFFER_SIZE);
            z_data_char.writeValue(z_data_send_buffer, BUFFER_SIZE);
          }

```

```
          send_counter +=  BUFFER_SIZE;
        }

        // Reset flags to prepare for new sampling.
        data_packed = false;
        keyword_recieved = true;
      }
    }

    // Turn off blue LED.
    digitalWrite(LEDB, HIGH);
  }
}
```

Appendix B

Sampling program - Accelerometer functions

```
/**
*****************************************************************************
    Program with all functions for controlling the ADXL372.
    @file ADXL372.cpp
    @author T. Thorvaldsen
    @version 1.1 2022-05-06 (yyyy-mm-dd)
  @par Revision History:
  - Version 1.0, April 2022: Inital version.
  - Version 1.1, May 2022: Removed redundant code lines.
*****************************************************************************
*/

/******************************* Include Files ******************************/
#include <SPI.h>
#include "Arduino.h"
#include "ADXL372.h"

/********************************* Functions ********************************/

/**
  Read and returns the value of a specific register on the accelerometer.
  @param address - Address of the register that is to be read.
  @return Value of read register.
*/
unsigned int read_Reg(byte address){
    int return_val;

    address = ((address << 1) | READ);

    digitalWrite(CS_PIN, LOW);

    SPI.transfer(address);
    return_val = SPI.transfer(0x00);

    digitalWrite(CS_PIN, HIGH);

    return(return_val);
}
```

```

/**
  Writes a value to a specific register on the accelerometer.
  @param address - Address of the register that is to be read.
  @param data - Value to be written to register.
*/
void write_Reg(byte address, byte data){
    address = ((address << 1) | WRITE);

    digitalWrite(CS_PIN, LOW);

    SPI.transfer(address);
    SPI.transfer(data);

    digitalWrite(CS_PIN, HIGH);
}

/**
  Packs data to be written to FIFO register.
  Then calls write function, sending packed fifo value and fifo register
  address.
  @param format - Chosen format.
  @param mode - Chosen mode.
*/
void set_FIFO(FIFO_FORMAT format, FIFO_MODE mode){
    byte data = ((format << 3) | (mode << 1));

    write_Reg(FIFO_CTL, data);
}

/**
  Calls write function, sending reset value and reset register address.
*/
void reset(void){
    write_Reg(SRESET, 0x52);
}

/**
  Calls write function, sending offset values and offset register addresses.
  @param x_offset - Chosen offset for x-axis.
  @param y_offset - Chosen offset for y-axis.
  @param z_offset - Chosen offset for z-axis.
*/
void set_Offset(unsigned int x_offset, unsigned int y_offset,
                unsigned int z_offset){
    write_Reg(OFFSET_X, x_offset);
    write_Reg(OFFSET_Y, y_offset);
    write_Reg(OFFSET_Z, z_offset);
}

/**
  Calls write function, sending interrupt value and interrupt register address.
  @param setting - Chosen interrupt setting.
*/
void set_Interrupts(unsigned int setting){
    write_Reg(INT1_MAP, setting);
}

```

```
/**
  Packs data to be written to timing control register.
  Then calls write function, sending packed timing control value and timing
  control register address.
  @param odr - Chosen output data rate.
  @param wur - Chosen wake up rate.
*/
void set_Timing_Ctrl_Reg(OUTPUT_DATA_RATE odr, WAKEUP_RATE wur){
    byte data = ((odr << 5) | (wur << 2));

    write_Reg(TIMING, data);
}

/**
  Packs data to be written to measurement control register.
  Then calls write function, sending packed measurement control value and
  measurement control register address.
  @param enable - Autosleep enable/disable value (1 or 0).
  @param link_mode - Chosen link/loop mode.
  @param noise_setting - Chosen noise mode.
  @param bw - Chosen bandwith.
*/
void set_Measurement_Ctrl_Reg(AUTOSLEEP enable, LINKLOOP link_mode,
                              NOISE_OP noise_setting, BANDWIDTH bw){
    byte data = ((enable << 6) | (link_mode << 4) | (noise_setting << 3) | bw);

    write_Reg(MEASURE, data);
}

/**
  Packs data to be written to power control register.
  Then calls write function, sending packed power control value and
  power control register address.
  @param threshold - Chosen instant on threshold value.
  @param settle_time - Chosen filter settling time value.
  @param low_pass_toggle - Low pass filter on or off.
  @param high_pass_toggle - High pass filter on or off.
  @param mode - Chosen operation mode.
*/
void set_Pwr_Ctrl_Reg(INSTANT_ON_THRESH threshold, FILTER_SETTLE settle_time,
                      LPF_TOGGLE low_pass_toggle, HPF_TOGGLE high_pass_toggle,
                      OP_MODE mode){
    byte data = ((threshold << 5) | (settle_time << 4) | (low_pass_toggle << 3)
                 |(high_pass_toggle << 2) | mode);

    write_Reg(POWER_CTL, data);
}

/**
  Calls write function, sending high pass filter corner value and high pass
  filter address.
  @param corner - High pass filter
*/
void set_High_Pass(HPF_CORNER corner){
    write_Reg(HPF, corner);
}

/**
```

```
  Reads value of status register 1.
  @return Value of status register 1
*/
int check_Status(void){
  return read_Reg(STATUS_1);
}

/**
  Reads values of the data registers for all axis.
  @param accel_data - Struct for holding data.
*/
void get_Data(AccelerBytes_t *accel_data){
    static short tmp;

    accel_data->x_msb = read_Reg(X_DATA_H);
    tmp = read_Reg(X_DATA_L);
    accel_data->x_lsb = (tmp &= 0b11110000);

    accel_data->y_msb = read_Reg(Y_DATA_H);
    tmp = read_Reg(Y_DATA_L);
    accel_data->y_lsb = (tmp &= 0b11110000);

    accel_data->z_msb = read_Reg(Z_DATA_H);
    tmp = read_Reg(Z_DATA_L);
    accel_data->z_lsb = (tmp &= 0b11110000);
}

/**
  Reads values of the device ID register. If device doesn't check out
  there might be something wrong with the accelerometer so it goes into
  an infinite loop.
*/
void get_Device_ID(void){
    static byte device_ID;
    device_ID = read_Reg(DEVID);

    if(device_ID != DEVID_VAL){
      while(1);
    }
}

/**
  Self test function to check if the accelerometer works as it should.
*/
void sf_Test(void){
    static byte check = 0;

    set_Pwr_Ctrl_Reg(LOW_THRESH, SETTLE_TIME_370ms, LPF_ENABLE,
                     HPF_DISABLE, FULL_BW);

    write_Reg(SELF_TEST, 1);

    Serial.println("Self test in progress");

    while(check != 2){
        check = read_Reg(SELF_TEST);
        check &= 0b00000010;
    }
```

```

    check = read_Reg(SELF_TEST);
    check &= 0b00000100;
    if(check == 4){
        Serial.println("Self test PASSED");
    }
    else{
        Serial.println("Self test FAILED");
    }

    set_Pwr_Ctrl_Reg(LOW_THRESH, SETTLE_TIME_370ms, LPF_DISABLE,
                     HPF_DISABLE, STANDBY);
}
```

Appendix C

Sampling program - ADXL372.cpp header file

```
/**
******************************************************************************
    Header file for ADXL372.cpp.
    @file ADXL372.h
    @author T. Thorvaldsen
    @version 1.1 2022-05-06 (yyyy-mm-dd)
  @par Revision History:
  - Version 1.0, April 2022: Inital version.
  - Version 1.1, May 2022: Removed redundant code lines.
******************************************************************************
*/

#ifndef ADXL372_H_
#define ADXL372_H_

#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#include "Arduino.h"

#ifdef __cplusplus
extern "C"{
#endif

/* Register addresses */
#define ADI_DEVID           0x00u   /* Analog Devices, Inc., accelerometer ID */
#define MST_DEVID                  0x01u   /* Analog Devices MEMS device ID */
#define DEVID               0x02u   /* Device ID */
#define REVID               0x03u   /* product revision ID*/
#define STATUS_1            0x04u   /* Status register 1 */
#define STATUS_2            0x05u   /* Status register 2 */
#define FIFO_ENTRIES_2            0x06u   /* Valid data samples in the FIFO */
#define FIFO_ENTRIES_1            0x07u   /* Valid data samples in the FIFO */
#define X_DATA_H            0x08u   /* X-axis acceleration data [11:4] */
#define X_DATA_L            0x09u   /* X-axis acceleration data [3:0] | dummy LSBs */
#define Y_DATA_H            0x0Au   /* Y-axis acceleration data [11:4] */
#define Y_DATA_L                    0x0Bu   /* Y-axis acceleration data [3:0] | dummy LSBs */
#define Z_DATA_H            0x0Cu   /* Z-axis acceleration data [11:4] */
#define Z_DATA_L            0x0Du   /* Z-axis acceleration data [3:0] | dummy LSBs */
#define X_MAXPEAK_H         0x15u   /* X-axis MaxPeak acceleration data [15:8] */
#define X_MAXPEAK_L         0x16u   /* X-axis MaxPeak acceleration data [7:0] */
#define Y_MAXPEAK_H                0x17u   /* X-axis MaxPeak acceleration data [15:8] */
#define Y_MAXPEAK_L         0x18u   /* X-axis MaxPeak acceleration data [7:0] */
#define Z_MAXPEAK_H         0x19u   /* X-axis MaxPeak acceleration data [15:8] */
#define Z_MAXPEAK_L         0x1Au   /* X-axis MaxPeak acceleration data [7:0] */
#define OFFSET_X                 0x20u   /* X axis offset */
#define OFFSET_Y                0x21u   /* Y axis offset */
```

```
#define OFFSET_Z                0x22u   /* Z axis offset */
#define X_THRESH_ACT_H          0x23u   /* X axis Activity Threshold [15:8] */
#define X_THRESH_ACT_L          0x24u   /* X axis Activity Threshold [7:0] */
#define Y_THRESH_ACT_H          0x25u   /* Y axis Activity Threshold [15:8] */
#define Y_THRESH_ACT_L          0x26u   /* Y axis Activity Threshold [7:0] */
#define Z_THRESH_ACT_H          0x27u   /* Z axis Activity Threshold [15:8] */
#define Z_THRESH_ACT_L          0x28u   /* Z axis Activity Threshold [7:0] */
#define TIME_ACT                0x29u   /* Activity Time */
#define X_THRESH_INACT_H        0x2Au   /* X axis Inactivity Threshold [15:8] */
#define X_THRESH_INACT_L        0x2Bu   /* X axis Inactivity Threshold [7:0] */
#define Y_THRESH_INACT_H        0x2Cu   /* Y axis Inactivity Threshold [15:8] */
#define Y_THRESH_INACT_L        0x2Du   /* Y axis Inactivity Threshold [7:0] */
#define Z_THRESH_INACT_H        0x2Eu   /* Z axis Inactivity Threshold [15:8] */
#define Z_THRESH_INACT_L        0x2Fu   /* Z axis Inactivity Threshold [7:0] */
#define TIME_INACT_H            0x30u   /* Inactivity Time [15:8] */
#define TIME_INACT_L            0x31u   /* Inactivity Time [7:0] */
#define X_THRESH_ACT2_H         0x32u   /* X axis Activity2 Threshold [15:8] */
#define X_THRESH_ACT2_L         0x33u   /* X axis Activity2 Threshold [7:0] */
#define Y_THRESH_ACT2_H         0x34u   /* Y axis Activity2 Threshold [15:8] */
#define Y_THRESH_ACT2_L         0x35u   /* Y axis Activity2 Threshold [7:0] */
#define Z_THRESH_ACT2_H         0x36u   /* Z axis Activity2 Threshold [15:8] */
#define Z_THRESH_ACT2_L         0x37u   /* Z axis Activity2 Threshold [7:0] */
#define HPF                     0x38u   /* High Pass Filter */
#define FIFO_SAMPLES            0x39u   /* FIFO Samples */
#define FIFO_CTL                0x3Au   /* FIFO Control */
#define INT1_MAP                0x3Bu   /* Interrupt 1 mapping control */
#define INT2_MAP                0x3Cu   /* Interrupt 2 mapping control */
#define TIMING                  0x3Du   /* Timing */
#define MEASURE                 0x3Eu   /* Measure */
#define POWER_CTL               0x3Fu   /* Power control */
#define SELF_TEST               0x40u   /* Self Test */
#define SRESET                  0x41u   /* Reset */
#define FIFO_DATA               0x42u   /* FIFO Data */

#define ADI_DEVID_VAL           0xADu   /* Analog Devices, Inc., accelerometer ID */
#define MST_DEVID_VAL           0x1Du   /* Analog Devices MEMS device ID */
#define DEVID_VAL               0xFAu   /* Device ID */
#define REVID_VAL               0x02u   /* product revision ID*/

/* Constants */
#define CS_PIN                  2u
#define READ                    1u
#define WRITE                   0u
#define INT_PIN                 3u

/* Type definitions */
typedef struct{
    byte x_msb;
    byte x_lsb;
    byte y_msb;
    byte y_lsb;
    byte z_msb;
    byte z_lsb;
} AccelerBytes_t;

typedef enum{
    CORNER_0 = 0,
    CORNER_1,
    CORNER_2,
    CORNER_3
} HPF_CORNER;

typedef enum{
    XYZ_FIFO = 0,
    X_FIFO,
    Y_FIFO,
    XY_FIFO,
    Z_FIFO,
    XZ_FIFO,
    YZ_FIFO,
    XYZ_PEAK_FIFO
} FIFO_FORMAT;
```

```
typedef enum{
    BYPASSED = 0,
    STREAM_MODE,
    TRIGGER_MODE,
    OLDEST_SAVE
} FIFO_MODE;

typedef enum{
    ODR_400Hz = 0,
    ODR_800Hz,
    ODR_1600Hz,
    ODR_3200Hz,
    ODR_6400Hz
} OUTPUT_DATA_RATE;

typedef enum{
    WUR_52ms = 0,
    WUR_104ms,
    WUR_208ms,
    WUR_512ms,
    WUR_2048ms,
    WUR_4096ms,
    WUR_8192ms,
    WUR_24576ms
} WAKEUP_RATE;

typedef enum{
    AUTOSLEEP_OFF = 0,
    AUTOSLEEP_ON,
} AUTOSLEEP;

typedef enum{
    DEFAULT_MODE = 0,
    LINKED_MODE,
    LOOPED_MODE
} LINKLOOP;

typedef enum{
    NORMAL_OP = 0,
    LOW_NOISE
} NOISE_OP;

typedef enum{
    BW_200Hz = 0,
    BW_400Hz,
    BW_800Hz,
    BW_1600Hz,
    BW_3200Hz
} BANDWIDTH;

typedef enum{
    LOW_THRESH = 0,
    HIGH_THRES
} INSTANT_ON_THRESH;

typedef enum{
    SETTLE_TIME_370ms = 0,
    SETTLE_TIME_16ms
} FILTER_SETTLE;

typedef enum{
    STANDBY = 0,
    WAKE_UP,
    INSTANT_ON,
    FULL_BW
} OP_MODE;

typedef enum{
  LPF_ENABLE = 0,
  LPF_DISABLE,
} LPF_TOGGLE;

typedef enum{
```

```
  HPF_ENABLE = 0,
  HPF_DISABLE,
} HPF_TOGGLE;

/* Function declerations */
unsigned int read_Reg(byte address);
void write_Reg(byte address, byte data);
void set_FIFO(FIFO_FORMAT format, FIFO_MODE mode);
void reset(void);
void set_Offset(unsigned int x_offset, unsigned int y_offset, unsigned int z_offset);
void set_Interrupts(unsigned int setting);
void set_Timing_Ctrl_Reg(OUTPUT_DATA_RATE odr, WAKEUP_RATE wur);
void set_Measurement_Ctrl_Reg(AUTOSLEEP enable, LINKLOOP link_mode,
                              NOISE_OP noise_setting, BANDWIDTH bw);
void set_Pwr_Ctrl_Reg(INSTANT_ON_THRESH threshold, FILTER_SETTLE settle_time,
                      LPF_TOGGLE low_pass_toggle, HPF_TOGGLE high_pass_toggle, OP_MODE mode);
void set_High_Pass(HPF_CORNER corner);
int check_Status(void);
void get_Data(AccelerBytes_t *accel_data);
void get_Device_ID(void);
void sf_Test(void);

#ifdef __cplusplus
} /* Extern "C" */
#endif

#endif /* ADXL372_H_ */
```

Appendix D

Processing program

```
###############################################################################
#
# Program for controlling the drop test device and saving received data.
#
###############################################################################
#
# This program is an altered version of a program found at:
# https://github.com/Ladvien/arduino_ble_sense/blob/master/app.py
#
# The below license stems form the original program.
#
###############################################################################
#
# MIT License
#
# Copyright (c) 2020 Thomas Brittain
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.
#
###############################################################################

# Library imports #
import os
import asyncio
import matplotlib.pyplot as plt
from datetime import datetime
from typing import Callable, Any
from aioconsole import ainput
from bleak import BleakClient, discover


# Output file path #
output_file = "C:/Users/tobbe/OneDrive - NTNU/3S Project/Programmering/Drop_Test/drop_test_python/"


```

```python
# Sensor system device name #
DEVICE_NAME = "Drop Test Device"


# Function to get input of test name for data saving purposes #
def get_output_file_name():
    global output_file
    test_name = str(input("Test name: "))
    output_file += (test_name + ".csv")


# Class for saving data to CSV and plotting data #
class DataToFile:
    column_names = ["time", "x-data", "y-data", "z-data"]

    # Class initialization function - Defines write path for output file #
    def __init__(self, write_path):
        self.path = write_path

    # Function for writing data to CSV file #
    def write_to_csv(self, x_values: [Any], y_values: [Any], z_values: [Any]):

        # Calculations from byte to g #
        x_data = [(round((float((x_values[i] << 8) | x_values[i + 1]) / 160), 1)) for i in
                  range(0, len(x_values) - 1, 2)]
        y_data = [(round((float((y_values[i] << 8) | y_values[i + 1]) / 160), 1)) for i in
                  range(0, len(y_values) - 1, 2)]
        z_data = [(round((float((z_values[i] << 8) | z_values[i + 1]) / 160), 1)) for i in
                  range(0, len(z_values) - 1, 2)]
        times = [i for i in range(len(x_data))]

        # Raise exception if not all data lists are the same length #
        if len({len(x_data), len(y_data), len(z_data)}) > 1:
            raise Exception("Not all data lists are the same length")

        # Save data to CSV file #
        with open(self.path, "a+") as f:
            if os.stat(self.path).st_size == 0:
                f.write(",".join([str(name) for name in self.column_names]) + ",\n")
            else:
                for i in range(len(times)):
                    f.write(f"{times[i]},{x_data[i]},{y_data[i]},{z_data[i]},\n")

    # Function for plotting data #
    def plot_data(self, x_values: [Any], y_values: [Any], z_values: [Any]):

        # Calculates from byte to g #
        x_data = [(round((float((x_values[i] << 8) | x_values[i + 1]) / 160), 1)) for i in
                  range(0, len(x_values) - 1, 2)]
        y_data = [(round((float((y_values[i] << 8) | y_values[i + 1]) / 160), 1)) for i in
                  range(0, len(y_values) - 1, 2)]
        z_data = [(round((float((z_values[i] << 8) | z_values[i + 1]) / 160), 1)) for i in
                  range(0, len(z_values) - 1, 2)]
        times = [i for i in range(len(x_data))]

        # Plots data of all axis in one figure #
        plt.figure()
        plt.plot(times, x_data, 'r', label='x-data')
        plt.plot(times, y_data, 'g', label='y-data')
        plt.plot(times, z_data, 'b', label='z-data')
        plt.title("Accelerometer data")
        plt.xlabel("Time [s]")
        plt.ylabel("Acceleration [g]")
        plt.legend()
        plt.show()


# Class for handling Bluetooth connection #
class Connection:
    client: BleakClient = None

    # Class initialization function #
    def __init__(
```

```
            self,
            x_data_characteristic: str,
            y_data_characteristic: str,
            z_data_characteristic: str,
            write_characteristic: str,
            data_handler: Callable[[Any], None],
            data_plotter: Callable[[Any], None]
    ):

        # Define class variables #
        self.x_data_characteristic = x_data_characteristic
        self.y_data_characteristic = y_data_characteristic
        self.z_data_characteristic = z_data_characteristic
        self.write_characteristic = write_characteristic
        self.data_handler = data_handler
        self.data_plotter = data_plotter

        self.connected = False
        self.connected_device = None

        self.x_data = []
        self.y_data = []
        self.z_data = []
        self.delays = []

    # Function for what happens when disconnected from device #
    def on_disconnect(self, client):
        self.connected = False
        self.data_handler(self.x_data, self.y_data, self.z_data)
        self.data_plotter(self.x_data, self.y_data, self.z_data)
        self.clear_lists()
        print(f"Disconnected from {self.connected_device.name} at {datetime.now()}!")

    # Function for properly ending Bluetooth connection #
    async def cleanup(self):
        if self.client:
            await self.client.stop_notify(self.x_data_characteristic)
            await self.client.stop_notify(self.y_data_characteristic)
            await self.client.stop_notify(self.z_data_characteristic)
            await self.client.disconnect()

    # Function for initializing connection over Bluetooth #
    async def manager(self):
        print("Starting connection manager")
        while True:
            if self.client:
                await self.connect()
            else:
                await self.select_device()
                await asyncio.sleep(15.0)

    # Function for setting up connection with Bluetooth peripheral device #
    async def connect(self):
        if self.connected:
            return
        try:
            await self.client.connect()
            self.connected = await self.client.is_connected()
            if self.connected:
                print(f"Connected to {self.connected_device.name} at {datetime.now()}")
                self.client.set_disconnected_callback(self.on_disconnect)
                await self.client.start_notify(self.x_data_characteristic,
                                               self.notification_handler_x,
                                               force_indicate=True)
                await self.client.start_notify(self.y_data_characteristic,
                                               self.notification_handler_y,
                                               force_indicate=True)
                await self.client.start_notify(self.z_data_characteristic,
                                               self.notification_handler_z,
                                               force_indicate=True)

                while True:
                    if not self.connected:
```

```python
                        break
                    await asyncio.sleep(3.0)
            else:
                print(f"Failed to connect to {self.connected_device.name}")
        except Exception as e:
            print(e)

    # Function for selecting which Bluetooth peripheral to connect with #
    async def select_device(self):
        print("Bluetooth LE hardware warming up...")
        await asyncio.sleep(2.0)  # Wait for BLE to initialize
        devices = await discover()

        device_number = -1

        for i, device in enumerate(devices):
            if device.name == DEVICE_NAME:
                device_number = i
                break

        if device_number != -1:
            print(f"Connecting to {devices[device_number].name}")
            self.connected_device = devices[device_number]
            self.client = BleakClient(devices[device_number].address)
        else:
            print(f"No device was found with the name: {DEVICE_NAME}")

    # Function for clearing variable lists #
    def clear_lists(self):
        self.x_data.clear()
        self.y_data.clear()
        self.z_data.clear()
        self.delays.clear()

    # Function for handling what happens when notification occurs on x-data characteristic #
    def notification_handler_x(self, sender: str, _data_: Any):
        tmp_list_x = [_data_[i:i + 1] for i in range(len(_data_))]
        for i in range(len(tmp_list_x)):
            if i == 0 or i % 2 == 0:
                self.x_data.append(int.from_bytes(tmp_list_x[i], byteorder="little", signed=True))
            else:
                self.x_data.append(int.from_bytes(tmp_list_x[i], byteorder="little", signed=False))
        print(f"X-data notification event happened at {datetime.now()}")

    # Function for handling what happens when notification occurs on y-data characteristic #
    def notification_handler_y(self, sender: str, _data_: Any):
        tmp_list_y = [_data_[i:i + 1] for i in range(len(_data_))]
        for i in range(len(tmp_list_y)):
            if i == 0 or i % 2 == 0:
                self.y_data.append(int.from_bytes(tmp_list_y[i], byteorder="little", signed=True))
            else:
                self.y_data.append(int.from_bytes(tmp_list_y[i], byteorder="little", signed=False))
        print(f"Y-data notification event happened at {datetime.now()}")

    # Function for handling what happens when notification occurs on z-data characteristic #
    def notification_handler_z(self, sender: str, _data_: Any):
        tmp_list_z = [_data_[i:i + 1] for i in range(len(_data_))]
        for i in range(len(tmp_list_z)):
            if i == 0 or i % 2 == 0:
                self.z_data.append(int.from_bytes(tmp_list_z[i], byteorder="little", signed=True))
            else:
                self.z_data.append(int.from_bytes(tmp_list_z[i], byteorder="little", signed=False))
        print(f"Z-data notification event happened at {datetime.now()}")


###############
# Loops
###############

# Function for sending string to Bluetooth peripheral #
async def user_console_manager(connection: Connection):
    while True:
        if connection.client and connection.connected:
```

```python
            input_str = await ainput("Enter string: ")
            bytes_to_send = bytearray(map(ord, input_str))
            await connection.client.write_gatt_char(write_characteristic_uuid, bytes_to_send)
            print(f"Sent: {input_str}")
        else:
            await asyncio.sleep(2.0)


###############
# App main
###############

# Bluetooth characteristic UUIDs #
x_data_characteristic_uuid = "1d3478a3-59db-4f59-8c27-a00a3a2a7a8e"
y_data_characteristic_uuid = "08e2eead-cc49-45f8-a24a-c0118ce005a2"
z_data_characteristic_uuid = "3fab2d16-53e2-46e8-bbe2-b171f12f5564"
write_characteristic_uuid = "a23be471-deb8-4fa2-8d22-10537858bbf3"

# Main #
if __name__ == "__main__":

    get_output_file_name()

    # Create the event loop
    loop = asyncio.get_event_loop()

    data_to_file = DataToFile(output_file)
    data_to_file.write_to_csv([0], [0], [0])

    connection = Connection(
        x_data_characteristic_uuid,
        y_data_characteristic_uuid,
        z_data_characteristic_uuid,
        write_characteristic_uuid,
        data_to_file.write_to_csv,
        data_to_file.plot_data
    )
    try:
        asyncio.ensure_future(connection.manager())
        asyncio.ensure_future(user_console_manager(connection))
        loop.run_forever()
    except KeyboardInterrupt:
        print()
        print("User stopped program")
    finally:
        print("Disconnecting...")
        loop.run_until_complete(connection.cleanup())
```

NTNU

Norwegian University of
Science and Technology