

Roni Laine

BLUETOOTH LOW ENERGY DATA TRANSFER IN ENERGY HARVESTER SYSTEM

Comparison of platforms and boards

Master's thesis in technology
Faculty of Information Technology
and Communication Sciences
Examiners:
Postdoctoral Research Fellow
Tero Kivimäki
Professor Jukka Vanhala
Mar 2022

ABSTRACT

Roni Laine: Bluetooth Low Energy data transfer in energy harvester system
Master's thesis
Tampere University
Electrical Engineering
March 2022

Energy harvesting is a widely known concept when used on a larger scale such as solar panels, wind turbines, and hydro plants, but tiny-scale energy harvesters are less utilized. Energy harvesters are applied to tiny-scale devices to build an autonomous system, to get longer operation time than devices powered by batteries, and to enable new applications. Bluetooth is also an extensively used technology in consumer products as it is found in the majority of mobile devices. In contrast, Bluetooth Low Energy (BLE) is a new technology and more challenging to familiarize with. The thesis will present the reasons to use energy harvesting and BLE. Another essential topic is examining development environments for BLE microcontroller boards and combining boards in a system containing an energy harvester and a sensor. The system is located in an isolated area that is difficult to reach and must have an energy source available for the energy harvester. Finally, the thesis explains the steps for software development.

A prototype was built in a research group and this thesis examines research knowledge and technologies related to the prototype. The prototype system includes an electromagnetic induction-based energy harvester. As the energy harvester is one of the main parts of the thesis, it requires further examination. Other project members tested and constructed the energy harvester, and this thesis includes a cursory overview of its use and operation. The implementation phase of the thesis focuses on developing software for BLE devices. After studying BLE theory, the next step is comparing and testing development environments and boards using data transferring programs. Ampiq Apollo3 SDK, Mbed Studio, and Arduino IDE are compared with SparkFun Artemis and Arduino BLE boards. The intended prototype testing environment will be the inner circle of a car tire, influencing the choice of the optimal BLE board which limits the case size and install area of the prototype.

As a result, a program developed in the Mbed Studio could not make a BLE connection, setting the comparison to a new perspective, focusing more on theory rather than development. The unexpected outcome showed that extensive background research does not ensure functionality. The unsuccessful result did not prevent continuing development with other environments. Implications and ideas regarding future work are documented for the next developer. The other two environments were functional and allowed a meaningful comparison. The differences between boards are most significant when use cases and development environment are taken into account. For example, boards differ by size, shape, and energy consumption. The Bluetooth features are similarly available on all boards, but taking advantage of them varies by board. The thesis concludes with a discussion about future progression paths in energy harvesting and Bluetooth Low Energy applications.

Keywords: Energy harvesting, Bluetooth Low Energy, Mbed Studio, wireless data transfer

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Roni Laine: Bluetooth Low Energy datansiirto energiaharvesteri systeemissä
Diplomityö
Tampereen Yliopisto
Sähkötekniikan tutkinto-ohjelma
Maaliskuu 2022

Energian harvesterointi on laajasti tunnettu käsite, kun sitä käytetään isommassa mittakaavassa kuten aurinkopaneelit, tuuliturbiinit ja vesivoimalat mutta pienen mittakaavan energia harvesterit ovat vähemmän hyödynnettyjä. Energia harvesteria käytetään pienen mittakaavan laitteissa, jotta saadaan itsenäinen systeemi, pidempi toiminta aika kuin paristoilla toimivilla laitteilla ja mahdollistaa uusia sovelluksia. Bluetooth on myös laajasti tiedetty teknologia kuluttajalaitteista kuten suurimmassa osassa mobiililaitteista löytyy Bluetooth. Kun taas Bluetooth Low Energy (BLE) on uusi teknologia ja haastavampi perehtyä. Tutkielmassa esitellään syyt käyttää energian harvesterointi ja BLE teknologioita. Toinen tärkeä aihe, jota käydään läpi tutkielmassa, on kehitys ympäristöt Bluetooth Low Energy mikrokontrolleri laudoille ja lautojen yhdistäminen systeemiin, joka sisältää energia harvesterin ja sensorin. Systeemi on sijoitettu eristettyyn kohtaan, joka on vaikea saavuttaa ja kohteessa täytyy olla energia lähde saatavilla energia harvesterille. Lopuksi ohjelmiston kehitysvaiheet on selitetty.

Prototyyppi rakennettiin tutkimusryhmässä ja tämä tutkielma tarkastelee tutkimustietoa ja teknologioita liittyen prototyyppiin. Prototyyppi systeemi sisältää elektromagneettiseen induktioon perustuvan energia harvesterin. Harvesteri on yksi tutkielman pääosista, joten se vaatii tarkempaa tutkintaa. Muut projektin jäsenet tekivät testauksen ja rakensivat energia harvesterin ja tutkielma sisältää pintapuolisen katsauksen energia harvesterin käytöstä ja toiminnasta. Tutkielman toteutus vaihe keskittyy kehittämään ohjelmistoa Bluetooth Low Energy laitteille. Sen jälkeen, kun BLE teoria on opiskeltu seuraava vaihe on verrata ja testata kehitysympäristöjä ja lautoja käyttäen datansiirto ohjelmia. Ampiq Apollo3 SDK, Mbed Studio ja Arduino IDE:ä vertaillaan SparkFun Artemis laudoilla ja Arduino BLE laudoilla. Prototyypin tarkoitettu testausympäristö tulee olemaan auton renkaan sisäkehä, vaikuttaen optimaalisen BLE laudan valintaan joka rajoittaa kotelon kokoa ja prototyypin asennus aluetta.

Seurauksena saatiin Mbed Studioissa kehitetty ohjelma, joka ei pystynyt luomaan BLE yhteyttä asettaen vertailun uuteen näkökulmaan. Tulos oli odottamaton ja näytti että vahvaan taustatutkimus ei varmista toiminnallisuutta. Tämä ei estänyt käyttämästä sitä tuloksena ja jatkamaan kehitystä muilla ympäristöillä. Tulevaa työtä koskevat vaikutukset ja ideat dokumentoidaan seuraavaa kehittäjää varten. Kaksi muuta ympäristöä olivat toimivia ja mahdollistivat merkityksekkään vertailun. Erot lautojen välillä ovat kaikkein merkittävimmät, kun laudan käyttötarkoitus ja käytetty ympäristö otetaan huomioon. Esimerkiksi lautojen koot, muodot ja energiankulutus vaihtelevat. Bluetooth ominaisuudet ovat samalla tapaa saatavilla kaikissa laudoissa mutta niiden hyödyntäminen myös vaihteli laudoittain. Päättäen tutkielman ajatuksilla tulevaisuuden kehitys poluista energia harvesteri ja Bluetooth Low Energy sovelluksissa.

Avainsanat: Energy harvesting, Bluetooth Low Energy,

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

PREFACE

I would still be looking for a topic in the field of embedded systems without the opportunity Tero Kivimäki and Jukka Vanhala gave me. Every child is asked what do you want to become when they grow up? My answer has always been a researcher, and now I have accomplished that. I would change a thing or two in my academic career but still, I ended up in a favorable situation. If somebody asked a younger me, to write something in English voluntarily, the answer would have been absolutely no. I want to thank Tero Kivimäki for patiently fixing my grammar and faulty reasoning. His guidance permitted this thesis to become a reality. I am also grateful to my colleagues for helping me in times of struggle.

Tampere, 2.3.2022

Roni Laine

CONTENTS

1.INTRODUCTION	1
2.ENERGY HARVESTING.....	3
2.1 Energy harvesting technologies	4
2.1.1 Photovoltaic	4
2.1.2 Thermoelectric	5
2.1.3 Piezoelectric	5
2.1.4 Electrostatic	5
2.1.5 Magnetostrictive.....	5
2.1.6 Electromagnetic	6
2.2 Convenient operational environments for energy harvester.....	6
2.3 Issues related to energy harvesters	7
3.BLUETOOTH LOW ENERGY	10
3.1 Alternative wireless communication protocols	10
3.2 GATT server and client communication.....	12
3.3 UUID	13
3.4 Services and characteristics.....	14
3.5 Consumption of electricity	14
4.PROTOTYPE.....	16
4.1 Prototype structure.....	16
4.2 Prototype function	18
4.3 Testing the prototype	19
4.4 About the final prototype	21
5.DEVELOPMENT ENVIRONMENTS AND BOARDS	23
5.1 Ampiq Apollo3 SDK.....	23
5.2 Mbed Studio.....	25
5.3 Arduino IDE	27
5.4 Android mobile phone	29
5.5 Nordic BLE boards.....	29
6.TRANSFER DATA OVER BLE CONNECTION.....	31
6.1 Arduino test and debugging code.....	33
6.2 Accelerometer program to measure acceleration	37
6.3 Mbed characteristics update	38
7.RESULTS AND COMPARISON.....	41
7.1 Comparison of development environments	41
7.2 Suitable boards	42
7.3 Use BLE for data transfer.....	45
7.4 Applications under development and future possibilities	45

8. CONCLUSIONS.....	48
APPENDIX.....	50
REFERENCES.....	61

LIST OF IMAGES

<i>Fig. 1 Kinetic energy harvester applying electromagnetic induction. Physical representation of structure (a) and the circuit diagram (b).</i>	3
<i>Fig. 2 The circuit from energy harvester to consuming device.</i>	4
<i>Fig. 3 Prototype dimensions (a) and a magnet on spring and a frame (b)</i>	17
<i>Fig. 4 Energy harvester connected to muscle maintenance hammer</i>	20
<i>Fig. 5 Energy harvester test setup (vibrator)</i>	20
<i>Fig. 6 Rubber mount</i>	21
<i>Fig. 7 Battery level service Arduino IDE Serial Monitor output</i>	32
<i>Fig. 8 Arduino Nano 33 BLE as the blue colored board above and SparkFun Artemis Edge as the red colored board below. On the left operation started, disconnected, or turned off. In the middle connection is established. On the right button pushed and LEDs are lighted up on both boards.</i>	34
<i>Fig. 9 Program 1 BLE Central device Serial Monitor output</i>	36
<i>Fig. 10 Program 2 BLE peripheral device Serial Monitor output</i>	37
<i>Fig. 11 User inputs x, y, z, or all to get requested values as an output</i>	38
<i>Fig. 12 Phone app nRFConnect showing GattServer and GattClient</i>	39
<i>Fig. 13 Boards from left to right: Arduino Nano 33 BLE and Arduino Nano 33 BLE Sense, two Artemis Nano, two Edge and two Artemis DK boards.</i>	43
<i>Fig. 14 Mbed Studio SFE target list</i>	44
<i>Fig. 15 Mbed Studio connected target</i>	44

LIST OF SYMBOLS AND ABBREVIATIONS

AC	Alternating Current
BLE	Bluetooth Low Energy
CLI	Command Line Interface
DC	Direct Current
DK	Development Kit
GATT	Generic Attribute Profile
GCC	GNU Compiler Collection
IDE	Integrated Development Environments
IMU	Inertial Measurement Unit
IP	Internet Protocol
LED	Light-Emitting Diode
LowPAN	Low Power Wireless Personal Area Network
LPWAN	Low Power Wide Area Network
MEMS	Microelectromechanical System
OS	Operating System
PCB	Printed Circuit Board
RAM	Random Access Memory
SDK	Software Development Kit
SFE	SparkFun Electronics
UI	User Interface
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
Wi-Fi	Wireless Fidelity.

1. INTRODUCTION

The surrounding environment has multiple natural and human-formed energy sources. Those energy sources are already being exploited to produce electricity on all scales but shifting to a tinier scale is essential to provide power supply for modern large-scale sensor networks. Transforming the energy available in the surrounding environment into electricity with an energy harvester provides autonomous operation for sensors and electricity to perform wireless data transfer [1]. Measuring sensor data, maintaining a device at standby state, and transferring data consume the limited power in the system. Saving energy necessitates optimizing the data transfer and utilizing sensors with low energy consumption. The Bluetooth Low Energy (BLE) protocol performs this required data transfer. The fundamental research focuses on the BLE data transfer and software development environment in energy harvester-powered systems.

Business Finland's Smart Energy Finland Program brings companies and research facilities to the ENOMA project. The author of this master's thesis is employed by the University of Tampere, one of the research facilities of the ENOMA project. The role of the University of Tampere in the project is to develop an inductive energy harvester, simulation tools, and data transfer for the system. The prototype developed in the project is for the Nokian Tyres company. They manufacture car tires, and the prototype is used to power up the sensor in the tire and send the sensor data to the user. [1]

This thesis begins by introducing the concept of energy harvesting. The main objective of the energy harvester is to power up the sensor and the data transfer with the BLE board. Chapter 2 goes through different energy harvester technologies, where energy harvester application is possible, and applicable use cases. Finally, the chapter predicts what kind of issues might get in the way.

Chapter 3 explains BLE features and the reasoning behind selecting BLE for this project. The chapter includes discussion about BLE functionality and connection parameters. Additionally, it explains BLE server and client differences and introduces Bluetooth and BLE differences and alternative wireless protocols. Chapter 3 also contains BLE-specific features, such as services and characteristics.

The prototype harvester designing process is discussed in Chapter 4. It contains a description of electromagnetic inductive energy harvester structure and functionality. In addition, the chapter briefly explains testing methods used to design the prototype and concludes with the description of the prototype assembling process and its application.

Chapter 5 introduces different options for development environments to choose from. It searches for information about environments and thoroughly researches the ones used in the project software development. The embedded system software platform also requires a microcontroller board that the software can be uploaded or connected to. The chapter also presents more information about the microcontroller boards.

Chapter 6 explains program functionality and parts of the code. Programs perform wireless data transfer and establish a BLE connection. Functioning programs require multiple testing and debugging steps to be robust. The chapter demonstrates the process of transferring the accelerometer sensor data and updating characteristics using events. The events are scheduled with the EventQueue class that executes functions based on time priority. Arduino boards are considered before moving on to SparkFun boards.

The last chapter includes discussion about results, conclusions, and future possibilities. Comparison of the development environments is in the first subchapter. The next subchapter follows with the results about the most suitable boards for the project and compares the different scenarios for which they are suitable. The chapter introduces applications in which the research technologies could be used or would be auspicious for future development. The end of the chapter summarizes conclusions about the project and the accomplished results.

2. ENERGY HARVESTING

An energy harvester is a device that collects energy from its environment to power a system close by. Kinetic, chemical, solar, radiofrequency, and thermal are examples of different forms of energies used as source energy [2]. The source energy is converted to electrical energy. For example, a bicycle dynamo collects energy from wheel movement and converts it to electricity that lights the lamp. Solar panels generate electricity from sunlight. Thermoelectric devices generate energy from surrounding temperature differences. Movements such as vibration and rotation provide kinetic energy. Kinetic energy is one of the most preferred energy sources along with light and heat. Most of these are ambient sources such as wind or human-made such as vibrating machines. Those are constantly available energy sources that are not fully utilized compared to their potential. Energy harvesters can produce electricity ecologically from the sources mentioned above.

One of the main reasons to use energy harvesting in autonomous systems is that changing batteries can be laborious [3]. Energy harvester can be installed in a sealed place that cannot be accessed after installation. Battery-powered devices always need to be accessed to change the batteries.

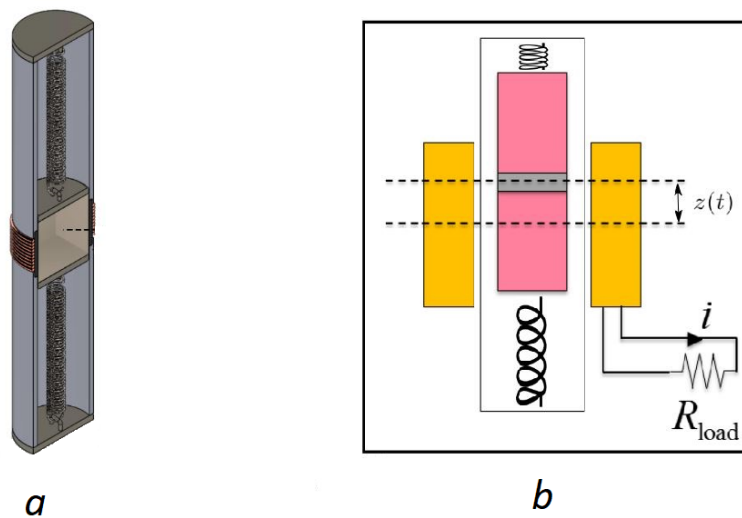


Fig. 1 Kinetic energy harvester applying electromagnetic induction. Physical representation of structure (a) and the circuit diagram (b).

This research focuses on powering up small devices and sensors with low energy consumption. Energy harvesters can also be used to power large systems, but they are not included in this research. The design mission is to get the system as tiny and efficient as possible. Figure 1 is a rough model of the energy harvester implemented in this research. Chiu *et al.* (2016) have a prototype of an electromagnetic energy harvester

similar to the Figure 1 double spring and coil harvester [4]. Later, Chapter 4 introduces a double coil and one spring prototype.

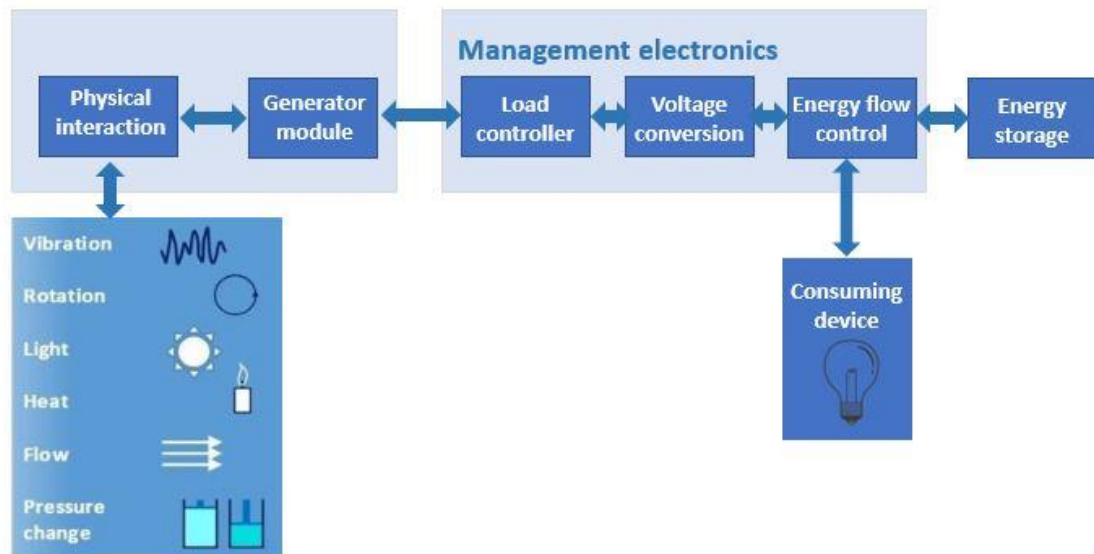


Fig. 2 The circuit from energy harvester to consuming device

In order to power the consuming device with an energy harvester, management electronics are needed to control the voltage feed. Tasks handled by the management electronics are shown in Figure 2. The management electronics provide a convenient operating voltage to the consuming device. Figures 1 and 2 are explained more carefully in Chapter 4.

2.1 Energy harvesting technologies

Various technologies are used to implement energy harvesting. The technology used is dependent on the source of energy. Kinetic source energy is converted to electricity using the electrodynamic phenomenon or piezoelectric materials. Kinetic energy harvesting efficiency depends on the amplitude and frequency of the energy source. The harvester is usually tuned to a certain frequency area so it might be tough to get electricity efficiently if the vibration frequency varies. From light, electricity is harvested using photovoltaic methods and from heat using the thermoelectric technique. [5] Energy harvesters convert the energy they collect from their surroundings to electrical energy which is appropriate for powering devices.

2.1.1 Photovoltaic

Converting light to electricity is a common way to harvest energy. Solar panels use the photovoltaic effect to produce electricity from light. The challenge with the light harvester is that light is not available constantly or in all locations. Clouds blocks sunlight at random

times and during different periods of time. The sunlight intensity varies during the daytime, and this energy source is not available at night. [6] To produce a significant amount of electricity, a vast amount of large and expensive solar panels are needed, which in turn require extensive space for installation. [5]

2.1.2 Thermoelectric

Thermoelectric energy harvesting produces electricity from the temperature differences between thermoelectric module terminals. Maintaining this temperature difference creates a thermoelectric effect and produces electricity. A Peltier module is this type of thermoelectric module. Peltier module terminals are made of two different materials, and applying temperature differences to the terminals generates an electric current. [7] Thermoelectric energy harvesting is viable only in places where the temperature difference is high enough. [5] The low efficiency and high cost of thermoelectric harvesting have prevented this method from becoming more prevalent. In recent years, escalations of both energy cost and demand have also made thermoelectric harvesting an appropriate technique for particular applications. Also, new materials to implement a more efficient thermoelectric module have been developed. [7]

2.1.3 Piezoelectric

The piezoelectric effect is based on a piezoelectric material, which generates electricity when mechanical strain puts pressure on it. Flow, rotation, or vibration can produce the mechanical strain on the piezoelectric material. The advantage of piezoelectric harvesting is its suitability to micro-scale and macro-scale fabrication. Also, power density is high compared to other kinetic harvester types. In some applications, a piezoelectric harvester is even capable of powering the consuming device without amplification or conversions. [8]

2.1.4 Electrostatic

In an electrostatic harvester, a change of pressure or vibration is employed to change the distance of capacitor plates. Moving the plates changes the capacitance of the capacitor which generates electricity. The generated electricity can be discharged from the capacitor to power devices.

2.1.5 Magnetostrictive

There are also magnetostrictive materials to generate electricity. Deforming magnetostrictive material induces changes to the magnetic field and generates electricity. [9]. The most common magnetostrictive energy harvester has a cantilever. The cantilever is made of magnetostrictive material, which oscillates in a pickup coil.

Magnetostrictive material does not depolarize like piezoelectric, but it needs the pickup coil as an additional part to produce electricity. [10] Ueno cites an example of a magnetostrictive harvester that utilizes vibration. The purpose of the harvester is to power the BLE transmitter and sensor, indicating that the magnetostrictive energy harvesting method is also suitable for battery-free BLE projects. [11] The shape of this type of magnetostrictive harvester is wide, and the strength of the beam might not be the most suitable for the car tire application in this project.

2.1.6 Electromagnetic

Electromagnetic induction occurs when a conductor travels commensurately to magnetic flux [6]. The electromagnetic harvester is presented in Figure 1, where the coil is the conductor, and the magnet is moving relative to it, generating magnetic flux. The electromagnetic induction is explained more carefully in Chapter 4, where the prototype built in this research is presented.

2.2 Convenient operational environments for energy harvester

Using an energy harvester can facilitate the maintenance of a system. The system maintenance interval will be longer because parts only need replacement if they wear out. It takes time to have energy harvesters developed in all the different environments where they can be used. The energy harvester must be optimized to its use scenario to provide adequate energy. Optimization produces challenges since the voltage and current might be low initially. Devices powered by energy harvesters operate in a certain voltage range. Voltage must be dropped or amplified to match the voltage range, but all these conversions cause energy losses. The embedded device intended to be powered by a harvester must be designed to be extremely low in power consumption. This applies not only to the harvesters but also to the devices powered by them, which must be further optimized.

Sensor placements in rapidly moving objects usually complicate the wiring to the main device. Local power source can solve this problem. A local power source allows sensors to be placed relatively far from the device processing the output data. Kinetic energy is available in rapidly moving objects, so using an energy harvester as a power source for the sensor is viable. Another problem related to installing sensors on moving objects is that data transfer must happen wirelessly. Even if the sensor is attached to a rotating object, the rotation will not cause interference with low-energy wireless data transfer operations. The energy harvester can provide energy for both the sensor and the data transfer operation.

Wiring to a remote location is expensive and laborious [5]. Energy harvester can provide energy for totally autonomous systems. La Rosa *et al.* have a system aiming for autonomous operation with an energy harvester without batteries for backup. They also selected BLE for the energy-efficient wireless communication protocol. [12] Autonomy is an advantage over other systems and offers more possibilities and use cases. Energy harvesters have their restrictions, for example, they need to be located in the immediate vicinity of the energy source.

An energy harvester is an alternative solution when a proximity energy supply for a system is needed. In an extreme example, an energy harvester can be more viable than batteries in a remote area. The first assumption is that the autonomous energy harvester systems work without errors. Solar panels located in the middle of the desert provide energy for a temperature sensor and wireless data transfer. The system is autonomous and has energy storage that might not last for the whole night when source energy is unavailable. This solution can replace batteries if the system does not depend on measuring the data during the night since it is possible for the system to be incapacitated some period of the night. Somebody must change the batteries if the system is powered by batteries. Even if a large number of batteries are stored close to the system, every time a battery runs out, somebody must change it. Old batteries produce chemical waste, traveling causes emissions, and a human or a robot to maintain the system is required. [3]

The most persuasive factor that the energy harvesters are useful is the assumption of their running indefinitely. Energy harvesters instead of batteries should be used as a power supply for longer operation. Setting up a large number of sensors without maintenance might allow using more sensors than planned. The initial design could include three sensors limited by maintenance, but an autonomous wireless sensor network might permit a considerably greater number of sensors. Installing two hundred sensors instead can provide considerably more data for an extended period of time. Problems can emerge when sensor are not working, then having two hundred sensors is more problematic. Also, producing energy harvesters is more complicated than producing batteries.

2.3 Issues related to energy harvesters

Multiple issues must be considered when planning to employ an energy harvester. Energy harvesting sources might be available only at a certain time, and source intensity might vary related to the location. If the amount of energy available is not at the required level and another energy source exists, different types of energy harvesters or multiples

of the same types of harvesters can be utilized. [6] More energy is harvested with multiple harvesters permitted to power a larger number of energy-consuming devices. Different types of harvesters also permit keeping the system running when one of the energy sources is not available. When one energy harvester malfunctions, another can keep the system running for a while. Multiple harvesters can be ready in idle mode to back up the main harvester if it fails. An energy harvester could also be used in parallel with a battery to prevent malfunction or provide missing energy requirements.

The energy harvester is usually under high stress. Source energy will strain the harvester, so all parts of the harvester have to sustain the stress. The harvester is designed to last in these circumstances by assembling it from resistant materials. The harvester can be attached to the energy source, but the measuring and data transfer components should be separated from the energy source if possible to prevent depleting the components and check any disturbance to operation. A kinetic energy harvester utilizing vibration moves back and forth, so the rest of the circuit would also undergo high stress. Installing sensors and communication units on the stationary parts of the system can protect them and prevent malfunction. The arrangement is possible if the sensor can measure required quantities from the stationary part. Also, the energy source movement should be more like vibration rather than rotation allowing stationary wires to be installed. The wires would break in a rotating motion when they twist over the limit. If rotation motion is back and forth and never goes over the limit, it is possible to have wires in that movement too.

Energy harvesters do not stress the main electric grid but replacing the main power source of a system with an energy harvester might be expensive. Nowadays, new solutions are still under development and not mass-produced otherwise, a more practical solution might be still more expensive than replacing batteries. Harvester components can be expensive and not made of the most environmentally friendly materials. Pollution is especially linked to the manufacturing process of the energy harvester and the end phase of its life cycle. However, most components of the harvester are recyclable and replacing more polluting power sources is still a more ecological option than not choosing an energy harvester with its many advantages. Most of the smaller scale devices are powered by batteries or accumulators, and they can be charged with fossil fuel. When those batteries run out of energy, they need to be replaced in all the sensors, which can be counted in the hundreds [3]. An energy harvester is a valid option in a battery-powered system in which an energy source exists and the system energy consumption is not too high. Energy harvesters cannot be utilized in locations where source energy does not exist. Tiny and lightweight harvesters are simple to install. Sometimes even a tiny change

to the system might change its behavior critically, so the original behavior of the energy source should not be affected too much by an external device. Experiments must be performed at the testing phase to ensure that the harvester does not cause any negative influences on the source energy system.

Energy harvester output voltage and current vary between different methods and harvesters. Phase, frequency, and amplitude vary in AC (*Alternating Current*) output and if the output is DC (*Direct Current*), the magnitude level varies [6]. Voltage conversion is frequently required to match device requirements. Transforming voltage lower or filtering high-voltage spikes might be required to protect and maintain the proper functioning of the device. These methods are required for usually unstable energy harvester output.

3. BLUETOOTH LOW ENERGY

BLE (*Bluetooth Low Energy*) is the main communication protocol used in this project. BLE development started in 2010, so it is a relatively new technology, but it is undergoing fast development [9]. BLE differs from standard Bluetooth in terms of the low energy aspect. In BLE, throughput is considerably lower than in other communication systems in order to save energy [13]. BLE is not appropriate if a large amount of data transfer is needed. Instead, for example, reading one sensor and sending that data wirelessly without using much energy is the proper use case for BLE. Energy consumption and throughput go together. Lowering energy consumption also lowers data throughput speed. Losing in one factor might represent a gain in another important factor when using BLE. The role of the device is either receiver or sender that significantly affects the energy consumption. More about the roles master and slave in the following chapters.

BLE can connect to a normal Bluetooth device that has Bluetooth 4.0 or newer and supports dual mode Bluetooth. For example, most phones can be used as a data receiver and send commands [9]. Older devices that support only Bluetooth Classic cannot connect with BLE devices, at least not directly. Both Bluetooth versions operate on a 2.4 GHz frequency band. [14]

Prior knowledge about Bluetooth Low Energy was not needed. The starting phase was getting deeper knowledge about normal Bluetooth than the basic consumer-level. Consumers can unconsciously understand basic electronic device energy harvesters like solar panels in a calculator, but for this project, a deeper understanding of energy harvesters is essential. The research will follow low energy and energy harvesting aspects of BLE software development.

3.1 Alternative wireless communication protocols

The ZigBee with IEEE 802.15.4 protocol is claimed to be a better low-power wireless communication technology than BLE in industrial use because of its wider range, long-term battery operation, flexibility in several dimensions, and reliability of mesh networking architecture. Interesting results regarding our research were found in Lee *et al.*'s work on intra-vehicular communication [15]. BLE performs well in intra-vehicular communication when Wi-Fi is interfering in the car. [15] That information supports the choice of BLE in the project-specific application. Another interesting point in the Lee *et al.* research was that ZigBee has long-term battery operation, but later in their research,

they mention that BLE has extremely low power consumption [15]. That should mean long-term battery operation in most circumstances for BLE also. Even though the 802.15.4 protocol is much used in wireless sensor networking, BLE is catching up with range and throughput. Combination chips support both ZigBee and BLE protocols lowering the competition between them. [16]

Z-Wave protocol is claimed to be more reliable and easier to use compared to ZigBee. Z-wave is used to run a network of nodes to control electrical devices. Transmitting short messages to send commands for nodes is an especially good example. [17] Transmission speed, range, and frequency band are lower compared to BLE. Z-Wave operates at the 1 GHz band, which usually has less interference from other sources. [18][19]

The 6LowPAN technology opened as Low Power Wireless Personal Area Network, and the number 6 comes from Internet Protocol (IP) version 6. The IP-based technology makes this different from BLE. IPv6 offers unique addresses to all users. 6LowPAN is a network protocol in which frequency band is adjustable. [18][19] BLE has also moved in the direction of IPv6. Support for the IPv6 mesh network might be coming to BLE in the near future. That would enable 6LowPAN support in BLE. [20]

Wi-Fi (*Wireless Fidelity*) is a powerful way to transfer data wirelessly, but it can consume too much power in low-energy applications. Otherwise, Wi-Fi is successful and popular technology suitable for many applications. Wi-Fi operates most commonly in the 5 GHz or 2.4 GHz bands. Many different companies have products using Wi-Fi connections, whereas BLE products are slowly coming to new companies as the development progresses. [21] High energy consumption makes Wi-Fi technology irrelevant for this research, and deeper inspection is unnecessary for widely known technology.

Sigfox and LoRaWAN are Low Power Wide Area Network (LPWAN) technologies, ensuring a longer transmission range than BLE. Transmission packet size is relatively small but is suitable for applications where transmission happens rarely, and the device must operate for a long time in remote areas. Data is transmitted from a base station to connected end devices with immediate access. Always-on base stations allow immediate access. Connected end devices have energy-saving features. For example, there is no need to listen to radio channels before transmitting. [22]

3.2 GATT server and client communication

GATT (*Generic Attribute Profile*) has all the definitions of how data is transferred between BLE connections. The GATT profile has all the services that the server offers. In the client, the GATT profile receives data from the offered services. The GATT server must have a Generic Access service that contains the device name and appearance. These are necessary facts needed for connection but quite useless alone.

GattServer and GattClient are the roles given for the boards to determine actions as transferring or receiving data. Server and client must somehow inform each other about their existence and look for each other. Advertising is a way for a device to declare its existence to other devices. The correspondingly scanning device is looking for other devices. One device can do both scanning and advertising consecutively. The server is usually scanning for clients that are advertising. [23]

The interesting part of the GATT profile is how receiver and transmitter are identified. Usually, master or central is the client, and slave or peripheral is the server. However, client and server can both be peripheral or central and even act as both at the same time. [24] The client is the one who wants data, so the client also makes the connection. [23] This research uses the client as master/central and server as slave/peripheral. The client will be at the user's end, providing user commands and responding with values received from the server. The client also searches for available devices, so having the client as a master allows it to connect to multiple sources. The server device will have sensors to measure required values. The server device is specified as a slave that advertises its existence and waits for a connection. The slave can connect only to one device that is suitable since one user at a time will read the data.

GattServer and GattClient communicate in four ways: *read*, *write*, *notify*, and *indicate*. The *read* command is used to read a value from the counterpart. *Write* is used to send the data. For example, in the code of this project, the client reads a command from the server. That command tells which axis acceleration value to write to the server. The client writes the requested acceleration data, and the server reads it and finally prints it. Another way to send data is *notify* by sending data without a request. *Indicate* also sends data without a request, confirming that the data is acknowledged. *Notify* and *indicate* commands require that the client subscribe to data updates. [13] Slaves as a server can connect only to one master as a client at a time, but the master can have several slaves connected. After a connection has been established, the server will stop advertising, but the client can keep scanning for other servers after the connection. In this case, sensors are servers, and the phone can be a client, which allows the phone to

connect all the sensors. If the sensors are servers, only one can connect to the phone at a time. Being able to connect to all servers simultaneously makes it easy to read and request data from the servers with the client device. A laptop or phone that acts as a client must be in close range to keep the Bluetooth connection strong. Usually, the client is a master device that is more powerful than server slaves. The slave can be a sensor and just send data, but the master must process this data and connect to many slaves. That is why laptops or phones are good masters with more processing power. A master BLE board can receive data, but the data processing can be done with a more powerful laptop.

3.3 UUID

The master device is scanning for slaves who are advertising their information. The slave is advertising themselves all the time and waiting for connection before it does anything else. The slave will advertise its name (optional) and service UUID (*Universally Unique Identifier*). Then, the chosen service runs the correct characteristics for the program like accelerometer or sensor data service. The master device is scanning with this same name and UUID parameters to form a connection. The UUID is a 128-bit long Bluetooth universally unique identifier. Services and characteristics have this specific UUID. Bluetooth has ready assigned UUID numbers for some example services. Ready assigned services are 16-bit long UUIDs and are approved in Bluetooth Special Interest Group. If there is already a defined UUID in Bluetooth specification, that UUID has a shorter form. In case the UUID is not assigned for the service, a random UUID can be generated. With a randomly generated UUID, the long 128-bit form must be used. [9] A good practice is to generate a new UUID for the final product to ensure there will be no connection issues. An interesting sidenote from a Silicon Labs article about overlapping UUIDs is that the handle of a UUID is always unique. At the testing phase, seeing the same UUID multiple times is not rare and usually does not prevent the device from working correctly. [23]

Three different test cases were created to test service and characteristic UUID. The first test was the example code UUID, whose origin was unknown. However, even with an unknown origin, it worked instantly. Another example of the same kind had different UUID numbers which indicates that accelerometer service or characteristics are not fixed. The second test was with a random UUID generator, and the third test selected random numbers for the UUID. Every single one of those tests worked. The outcome of the tests is that setting up a UUID is simple as long as the tests use the correct number of characters, which are either in number or hexadecimal format.

3.4 Services and characteristics

Data is organized with services in GATT. Services include pieces of data on characteristics. Characteristics are finally at the top level since they contain data that users will use and see. Services and characteristics both have UUIDs for identification. Characteristics have attributes that explain what data they contain, and another attribute contains the data value. Characteristics are defined for example as *read* or *write*. Depending on the defining characteristic, a value can be read, or a new value can be assigned with write. Characteristics can be defined as *notify* or *indicate* when a client subscribes to get information about characteristic value changes. When a subscription is indication-based, the client must acknowledge that it received the data. Characteristic definitions are not limited to one; the same characteristic value can be for example written, read, and notified. The client does not know server service attributes so after connection it discovers what attributes a given server service offers. After attributes are discovered, data transfer between devices can start. [9]

Examples of BLE services, such as Battery service and Heart Rate service, are included in Mbed. Both services continuously send data similarly to the accelerometer service that will be used in this project. The difference is that the accelerometer service sends more data and preferably multiple values at the same time. Since x, y, and z values give more accurate acceleration results, all three axes can be read at the same time.

3.5 Consumption of electricity

In the project setup, 900 values are gathered from an accelerometer at a frequency of 15 kHz. The measured data is transmitted over BLE, and the energy consumption of the transmitter is measured. Results indicate 240 μA average current consumption, a minimum of 4.5 μA and a maximum of 4500 μA . At the deep sleep state, power consumption is 1.8 μA [5]. Sleep mode and deep sleep are essential to save energy. Between connection events, the device is not performing any tasks still consuming power, so setting devices to sleep saves power. In the sleep mode, the device shuts down everything unnecessary. The sleep timer must be turned on in order for the device to wake up again. [25] Sleep state consumes a minor amount of power but multiple times less than normal operating mode. If the device has multiple clocks in a sleep state, it is possible to turn off excess clocks. Otherwise, clocks are processing continuously and consuming energy. The wait command is synchronous in the code and consumes much energy so it is better to turn on sleep mode rather than waiting and thus consuming electricity. [26]

The intervals between advertising and scanning events and their duration should be set in a way suitable for the application. For example, a short scanning period consumes less power at the cost of the possibility of missing other device advertising events. The same result happens if periods between scanning events are too long. It might result in multiple advertising events being missed, and this also goes the other way around. If an advertising event is too short-term or rarely executed, establishing a connection can take long time or even fail. Setting up the periods and time intervals between events determines connection success rate and speed. Periodic advertising and scanning one after another took longer to manage the connection since the devices could be both scanning at the same time. One device must change to an advertising event while the other is still scanning. This also goes the other way around when both devices are advertising the other must change to scanning. There is a possibility that the devices start simultaneously, and periods are similar, preventing establishing a connection.

4. PROTOTYPE

A harvester prototype was designed and implemented to prove the hypothesis that a harvester can successfully operate as a power source for a system consisting of a sensor and a BLE transmitter. Electromagnetic induction is used as the harvesting technology. Vibration motion provides kinetic source energy for the energy harvester. The structure and the operation of the prototype are explained in this chapter.

The simulation tool created in the project is a finite element method (FEM) -based tool implemented on top of the MATLAB platform. The simulation tool is computing magnetic quantities. The analysis of simulation results is used to design the prototype. Measurements of coil maximum current and output voltage need to be known to calculate the amount of electricity the powered device can consume. Different simulation systems are used for other properties. [5]

The prototype operation is explained briefly here. First, the harvester converts source energy to electricity. The harvester produces electricity at a certain voltage range, which is converted by energy management electronics shown in Figure 2 to adjust the output for consuming devices. The circuit contains energy storage to store extra energy and keep the circuit running longer when the energy harvester is not producing electricity. Finally, energy management electronics provide electricity to a device powered by the harvester. The prototype harvester structure is simple and inexpensive compared to other harvesters.

4.1 Prototype structure

The harvester frame is made of polyacetal, and the frame is wound with copper coil. The coil is divided into two separate loops to capture even smaller vibrations effectively. Wires come out from the coil to the management electronics. Inside the frame is a permanent magnet resting on top of a spring, as shown in Figure 3. The magnet is installed inside of the frame to a hole. The hole is similar in shape to the magnet, and the diameter is slightly larger compared to the diameter of the magnet to keep the magnet in place sideways but permits it to vibrate up and down. The magnet is a cylindrical-shaped neodymium magnet. A few 1-mm sized holes are drilled in the bottom of the frame to let air flow out and permit the magnet to vibrate with less resistance. Polyacetal is chosen as the frame material to have as little friction between magnet and frame as possible. There exist materials with a lower coefficient of friction, but the durability of the

material is equally relevant to the low coefficient of friction. Polyacetal provides both required features. Energy is stored in a supercapacitor to hold energy for longer operation in case new energy is not generated for a while. [5] Everything is assembled on a tiny round-shaped base to connect the harvester to the inner side of a car tire.

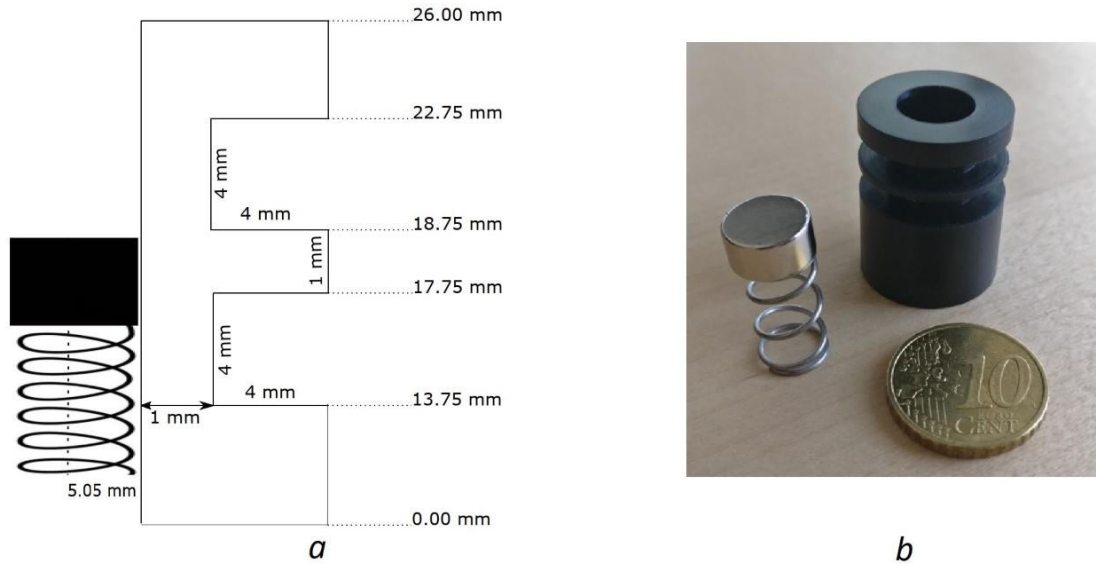


Fig. 3 Prototype dimensions (a) and a magnet on spring and a frame (b)

Harvester size relates to the required power. The larger frame allows a larger magnet and more turns on the coil, providing higher output power. A smaller design is preferred if the power production requirement is not the dominant demand. A tiny-size harvester does not affect the source system much, as the weight and the air resistance are insignificant. The air resistance does affect rotating systems more than vibrating systems. Implementing the harvester system on a smaller scale is also advantageous since the source system providing kinetic energy to the harvester might have little free space for harvester placement. Even though the harvester dimensions in Figure 3(a) are not large, it usually takes more space than a battery designed for the same output level. The harvester should not draw an enormous amount of power from the energy source system that endangers the energy source system operation.

The selected spring is made of stainless steel. The diameter of the spring is 9.14 mm, and the length is 15.75 mm in the resting position. The magnet is a nickel-plated permanent magnet with 10 mm x 5 mm dimensions. The copper coil is still missing in Figure 3, but the coil position in the frame has a 0.1 mm diameter where the copper wire is applied 2000 turns. The coil is done two times since the frame has two spots for different coils for more constant output. Also, a supercapacitor is added to produce a more constant output. A supercapacitor with 220 mF capacitance is operating as energy

storage. All the components are placed on a 20 mm x 30 mm Printed Circuit Board (PCB). [5]

A harvester is powering an acceleration sensor and a BLE transmitter to transfer data. A BLE module is used to transfer the data wirelessly. The plain module is smaller than the board, having dimensions of 10 x 15 mm, not including the accelerometer sensor [27]. The separate accelerometer sensor is tinier than the board, decreasing the case dimensions. BLE boards can include both sensor and BLE transmissions, but many of the commercially available boards also include pointless parts for this project device. Extra components take space and might consume limited energy in a harvester-powered system.

The prototype accelerometer ADXL1003 is a low-noise, wide-bandwidth, MEMS (*Microelectromechanical System*) accelerometer. The ADXL1003 has a low power consumption of around 1 mA when the accelerometer operates. The accelerometer has fast recovery from power-saving mode and small packaging dimensions of 5 mm x 5 mm x 1.8 mm. The power supply is limited, so all energy-saving factors are essential for sensors. The ADXL1003 is stable and immune to external shocks, which is important for installation location. The temperature range in the ADXL1003 is from -40 °C to +125 °C, which should satisfy the cold temperature climate of Finland and operate well in other countries too. The operation voltage range is from 3 V to 5.25 V, which can be achieved with an energy harvester. [28]

Prototype BLE communication is implemented with the Artemis module. The module has a 48 MHz regular frequency and 96 MHz available turbo frequency. The module has BLE version 5.0 and a 32-bit ARM Cortex-M4F processor. Electricity consumption is around 6 μ A per MHz. Low power allows the module to run even with a single coin-cell battery indicating that low energy harvester output power should also be suitable for the module. Energy harvester output is not constant like that of a battery, so it does not guarantee compatibility, as previously noted. The Apollo3 chip has 1 MB of flash memory and 384 k of RAM. Most importantly, a built-in BLE antenna is included. The Artemis module has an Arduino core making it compatible to be programmed as Arduinos. [29]

4.2 Prototype function

The electromagnetic induction harvester utilizes Faraday's law of induction. Based on this law, the conductor generates an electromotive force when the time-varying magnetic flux is present [30]. Vibration moves the magnet back and forth, creating a time-varying

magnetic flux. When the permanent magnet moves inside the coil, it moves magnetic flux through the coil surface, creating electromotive force to the conductor coil. The coil could also be moved but wiring the coil would be harder. The preferred solution is to move the magnet. [5] The coil is wound outside, and the magnet is located inside the frame, so moving the coil would make outside parts move leading to more malfunctions. When the magnet is inside the frame, it is also protected from dirt and other distractors.

The prototype energy harvester is located inside a car tire. High acceleration and pressure exist inside a rotating tire. Rotation motion combined with vibration is exploited by the energy harvester. Road roughness causes vibration motion that the energy harvester mainly utilizes. The tire rotation motion is not effectively producing energy since the energy harvester is designed to utilize vibration. The energy harvester powers up a sensor that can measure tire acceleration. Later the sensor can be upgraded to measure more variables such as temperature, pressure, force, and friction between tire and road. Important information delivered quickly in real-time to the driver improves driving safety. The system that provides real-time information from the tire is called a smart tire system. The prototype focuses on measuring acceleration and in the developing phase, the voltage of the harvester is also measured to optimize the rest of the circuit. In the final product, the energy harvester should function correctly without the need to measure the output voltage anymore. Accelerometer sensor data is the desired information. The prototype is designed to be as tiny and lightweight as possible as it shall not affect driving capabilities, and installing the mount does not provide a large surface area. The energy harvester and the system must be robust because inner tire forces are fairly high. [31]

The sensor measures a considerable amount of data to be transmitted wirelessly. In order for the energy harvester to provide a sufficient amount of energy to the system, management electronics have to amplify the electricity and optimize the sensor for low power usage. The energy conversion efficiency of the energy harvester is precisely optimized to function well in the car tire or in other applications where energy is limited.

4.3 Testing the prototype

Several prototype variations were built for testing purposes, and different methods were applied to simulate energy harvester vibration and rotation. The following paragraphs briefly describe the energy harvester tests about the prototype implemented during the ENOMA project. The author performed the testing related to the wireless communication and BLE features, as explained in Chapter 6.



Fig. 4 Energy harvester connected to muscle maintenance hammer

A muscle maintenance hammer is cheap and convenient for straightforward functionality testing. The test setup is not close to the final environment, but it helped to get results at the beginning to design measuring systems and simulation settings. Shaking the energy harvester prototype with a hand quickly became exhausting, and even after resting, results were weaker than with the muscle hammer. The harvester connected to a muscle hammer is shown in Figure 4. The white part in Figure 4 is a 3D-printed holder for the harvester. The muscle hammer is more mobile, cheaper, and easier to access than to the vibrator shown in Figure 5. The muscle hammer is convenient for testing whether the energy harvester gives any output to ensure the correct operation.

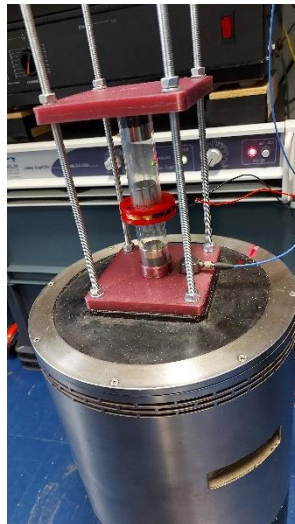


Fig. 5 Energy harvester test setup (vibrator)

The setup shown in Figure 5 is for testing the electrical output of the energy harvester by vibrating the energy harvester between red plates. The energy harvester is tightened to the top of the vibrator with a custom framework built around the harvester. The vibrator weights around 40 kg becoming fairly immobile. Therefore, setup is being tested in the laboratory and connected to a PC and measuring equipment. Frequency and amplitude are adjustable in order to get results from the sought-after area. Predictions and

analyzing can be taken quite far with this test setup but the prototype will face rotation and varying conditions in a real-life environment that is hard to predict.

The multimeter has a digital number screen for testing electrical outputs voltage and current. Frequency and amplitude cannot be measured with a multimeter. The multimeter scale is not enough to measure low changing voltage that is required in a low power system. Testing can be done with an oscilloscope that has frequency and amplitude displayed on the screen. The oscilloscope can also store measured values that help to interpret the values and to notice slight changes. The multimeter is only capable of displaying the current value.

4.4 About the final prototype

The prototype is installed inside a car tire with a piece of rubber as shown in Figure 6. All the parts of the system: energy harvester, sensor, and BLE module must fit on this rubber mount. The circular rubber mount diameter is only about 4 cm where everything needs to be firmly connected.



Fig. 6 Rubber mount

The final prototype implemented in the ENOMA project will provide a driver important information about the behavior of the car via an accelerometer sensor in one or up to all four tires giving information about acceleration. Acceleration values are quantities that the product can measure without modifications. Modifications are reasonably simple so changing the sensor module and code to measure other attributes later is feasible. During the project, the low power wireless communication and accelerometer sensor combination is developed as far as possible. The final use scenario is yet unknown, but it is possible that product development can take different routes in the future.

The thesis author contributed to the final product of the project by researching the development environments and process of BLE development which are introduced in the next chapters. The results of this research can be used by a person with no previous experience or knowledge of BLE to choose the right platform and to get on with developing data transfer and code. Having an energy harvester as a power source gave development a certain direction to the approach. Everything was designed to manage functioning under the low power output of the energy harvester. That meant coding energy-saving features like sleep modes to the communication systems. Unfortunately, electricity is not available all the time, so circuit operation shuts off occasionally and has to restart rapidly when electricity is available.

5. DEVELOPMENT ENVIRONMENTS AND BOARDS

The ENOMA project had formed a certain shape for program and components earlier, but alternative options and solutions are continuously developing during research inspecting other options for existing wireless communication methods. Developing boards for testing had dimensions too large to fit in the ENOMA project car tire. Learning to develop BLE methods in different platforms with multiple boards contributes to the aim of developing even smaller modules that can be used in the project. The same methods used in larger boards can also be applied to tiny modules. The reasons to use boards first rather than tiny modules are easy connection plugs and more physical testing methods available. For example, debugging with LEDs (*Light-Emitting Diodes*) that are later pointless in most applications. Initialization for the environments needed the internet to install the required software, but development can continue offline. During development, the possibilities to have more functions and develop some other applications were kept in mind to open more future applications.

Coming up with examples and instructions varies between different boards and development environments. One certain board had examples only on a certain platform. All the environments had some basic examples to get started with. In the example, another device is a laptop or phone most of the time. The problem with that is that even the official example codes were supposed to operate between two boards without modification that was not the case.

The thesis compares which environment is convenient for what kind of user and in what kind of situation. The comparison examines reasons that make the specific development environment hard or easy and explains how to use that specific environment for developing and setting it up. The first subchapter is a collection of the software to enable development with Apollo boards. The next two subchapters are about Integrated Development Environments (IDE) and compatible boards for the environment. After those subchapters, other developing platforms and developing boards are discussed.

5.1 Ampiq Apollo3 SDK

Development was started with Ambiq Apollo3 SDK (*Software Development Kit*) because this method was previously used in the project to develop the code to measure the tire

acceleration. Connecting two devices with a Bluetooth connection should be possible when BLE boards are developed in the same environment. It had to be taken into consideration when changing development environments that constructing connections between boards might get harder when development environments are different. Ambiq Apollo3 SDK is not easy to start with since it contains many different things to configure before code can even be uploaded. There were multiple programs to install, for example Python3, make, and Git Bash. [32] Then another step was choosing which code editor to use, such as Notepad++, which did the job at the beginning. The make command was used to upload code to the board, but even some example codes makefile needed modification to work properly. Keeping track of file locations was important otherwise, the makefile would not work. If the file is one folder above or lower than it is supposed to be, dependencies between library files might get out of track too.

The best-fitted board for this environment is Edge. Edge boards have sensors that make different kinds of examples possible. AmpiqSuiteSDK had more examples for Edge like printing accelerometer x, y, and z coordinates. The SparkFun tutorial for setting up AmpiqSuiteSDK, used an Edge board [32]. Later it was tried with other boards like Artemis Dev Kit and Artemis Nano, resulting in problems with the example codes that were missing or did not work. More deep research was needed to modify makefile and codes for boards other than Edge, where examples were more set and better prepared. This method supports multiple Artemis boards so even if it is not the easiest method, it might end up being useful.

This chapter introduces set up for the Ampiq Apollo3 SDK environment for SparkFun Artemis boards. Firstly, a Bash or command line is needed to install since it is used in almost all other installations and in uploading the code in the board with this guide. Git Bash is recommended in the guide, and previous knowledge led to selecting it again. Because of having a Windows operating system (OS) installing make is needed. After installing GNU MCU Eclipse Windows Tools, the make command will not work before adding it to the Windows PATH. After searching “advanced system settings” in the Windows taskbar and then clicking “environment variables”, it is possible to add to Windows PATH. Python3 is the next thing that needs to be installed. This part gave struggles since in Git Bash command Python3 leaves it hanging, which means nothing happens after giving the command, but Bash is not giving an error which means Python3 is working. Entering “:wq” breaks the hanging, and Bash is again ready to be used. Trying to fix the hanging issue for a long time, but it is a Git Bash problem because in the Windows command line python3 command gives the Python version, and it works. Later realized that Python3 was working because uploading code became possible. There

might still be hidden environmental issues that have not come up yet. Also, pycryptodome and pyserial packages need to be installed with pip3 or pip command. These commands can also be problematic and not work, but to get around the issues, change the name of python3.exe to python.exe. Then pip commands worked, followed by changing the name back to python3 required by the example. The GNU ARM Embedded Toolchain compiler is installed and added to the PATH. Using the “arm-none-eabi-gcc -version” command for example in Git Bash it is possible to see if GCC (*GNU Cross-Compiler*) is successfully installed. The compiler was also useful in another development environment Mbed Studio. All the necessary files and example codes can be received from Ambiq Suite SDK GitHub. A new problem was encountered when downloading the files included in the zipped folder. This GitHub included linked folders, and in those folders the text was blue like in internet links, and none of the content inside that folder was included in the download. An easy solution was to download the content inside that link folder separately and locate it in the right folder on the local PC. It is not a good solution, though, because there might be many of the same kinds of folders, which is exactly what happened. After executing the make command to upload example code, two more link folders were still found without content. After fixing those and looking at the board COM port number from the Device Manager, the environment is ready to execute the first upload. [32]

5.2 Mbed Studio

The next development environment was Mbed Studio which had a more user-friendly UI (*User Interface*) and a simpler way to upload code to the board. The Mbed Studio programs are written with C and C++ languages [26]. The Mbed Studio environment is meant to be the new development area in this project, and the purpose is to compare it to others. The Mbed OS online version was first in consideration, but SparkFun Artemis boards were not on the list of programmable boards. Boards from different manufacturers could have worked but since SparkFun boards were added in the Mbed Studio desktop version, it was chosen. Another option was Mbed CLI (*Command Line Interface*), but it would be quite similar to Ampiq Apollo3 SDK so Mbed Studio performance was tested. Starting with Mbed Studio was not simple either because the target board had to be chosen, which was simple to manage but then creating an active program was more complicated. The active program can be made by adding the Mbed operating system library to the project. The problem was not knowing it was missing, and it took some time to figure out what must be added to get the program active. It is understandable that the program will not work without the OS, but manually adding OS was troublesome. This

became an issue after trying other examples than those in the Mbed Studio which did not have the OS library included. After comparing the Mbed Studio Blinky program to other examples, it is figured out that the OS was missing. After learning how to make an active program and adding necessary libraries, the code was simple to upload. Uploading takes a long time when building and running a new active program the first time but the next run is significantly shorter, although still longer than in the Ampiq Apollo3 SDK environment. Then facing the next problem, nothing was printed to the Serial Monitor. After some research, it is found out that Arm Compiler 6, which Mbed Studio uses, is not compatible with SparkFun Artemis boards to get print in Serial Monitor, which was necessary for debugging. In the previous environment Arm Embedded GCC compiler was installed for Ampiq Apollo3 SDK. The same compiler is needed to get the print commands to appear in Serial Monitor. Adding the new compiler is done by editing one ".json" file. Now Mbed Studio can use the same compiler as the previous environment. After installing the newest compiler version, Mbed Studio warned that the version had to be between 9 and 10. The version was changed to recommended Arm Embedded GCC Compiler version 9-2020-q2-update.

Mbed Studio compiling time is relatively long compared to previous and upcoming environments. The clean build sometimes takes somewhere between 10 and 40 min when it compiles all the included libraries again. After a clean build, the build file is generated, making the next build faster. Still, even with the build file, Mbed Studio is many times slower than other platforms.

A compatible board to this environment is Artemis Dev Kit, shortened as Artemis DK (*Development Kit*). Other SparkFun Artemis boards were also in the target list but after plugging them in Mbed Studio did not recognize them. Artemis DK dimensions 3.81 x 9.4 cm are not suitable for the project because the size is too large for the Figure 6 rubber mount. Artemis DK was suitable for testing purposes and code development. All the three Artemis SparkFun boards that were tested in the project have a USB-C connector to power up the board with cable and program the device. [33] There is one exception; the Edge board needs an extra serial basic breakout adapter to add a USB-C connector. In the Edge board, holding down button 14 and pressing reset were required to upload code with all environments. What is special about the Artemis DK board is that it has a memory unit. This makes different kinds of programming possible. The method is called drag-and-drop programming when the files are dropped to the memory programming the Artemis DK microcontroller. The drag-and-drop method does not need Mbed Studio and it is introduced as an alternative way to program this specific board.

5.3 Arduino IDE

Arduino IDE should be installed to begin environment set up. The first required functionality is adding additional boards to the board manager and libraries to the library manager. Central and peripheral code both needs the ArduinoBLE library included. Peripheral also need the Arduino_LSM9DS1 library to run the accelerometer sensor. When using the Arduino Nano 33 BLE board and SparkFun RedBoard Artemis Nano, they are not in the basic Arduino board list, so it is necessary to add them from the board manager. They can be found after including Arduino Mbed OS Nano boards and SparkFun Apollo3 boards. It is also possible to use other SparkFun Apollo3 boards, but the Nano version can be used with the same code as Arduino Nano 33 BLE. Arduino Nano 33 BLE Sense is chosen as a peripheral device because the Sense version included a sensor like the 9-axis inertial sensor, which allows measuring acceleration [34]. Later turned out that basic version Arduino Nano 33 BLE also had 9-axis inertial sensor so both boards could perform as peripheral and central. The Sense board had a few extra sensors that were not used in this project. Previous knowledge helped to get example codes easily running on both boards. Testing was done first by establishing a Bluetooth Low Energy connection between the boards and testing the accelerometer separately before establishing a connection to have smaller debugging blocks.

Compatible boards to the Arduino IDE environment were the Arduino Nano 33 BLE and Artemis Nano. Also, all the other boards used in this project should be supported by Arduino IDE. Arduino Nano 33 BLE has an alternative board with the same name except "Sense" added to the end of the name. The Sense version has more sensors than the normal version. The Arduino Sense board is used to send the data because it has a 9-axis inertial sensor (IMU). The IMU sensor is an LSM9DS1, and the 9-axis spreads to three 3-axis combinations of the values accelerometer, gyroscope, and magnetometer. [34] The Sense version has two disadvantages compared to a normal one. Sense has more power-consuming components, so it draws more power and costs more. An inertial sensor is needed at the energy harvester side, but any extra energy consumption should be avoided. In the end, Arduino Sense will be simulating the data and the connection that would be coming from the final product. As a data receiver, Arduino Nano 33 BLE can be used in the final product since there is enough space and power available for the receiver side. The receiver is connected to a phone, laptop, external power supply, or the electronic power system of a car.

Other boards were also tested with Arduino IDE environments like Artemis DK. Even though it had good support in other environments, there were some issues establishing a connection in Mbed Studio with Artemis DK boards. Arduino IDE was used

to confirm that the boards were unbroken. The Artemis DK upload tool was missing in Arduino IDE, which was expected for another manufacturer's board. Foremost SparkFun boards had to be added to the board manager by adding ".json" file URL (*Uniform Resource Locator*) to "additional board manager" in the preferences tab, thus allowing SparkFun Apollo3 boards to be found in board manager. In many cases, the bootloader is the program that loads the code on the board. For Artemis DK, uploading code to the board does not happen that way. The code can be compiled and exported as a binary file. Artemis DK appears as a mass storage device in a PC that allows drag-and-drop programming to the device. The compiled binary file is dragged to Artemis DK storage, and when dropped there it programs the board. The appropriate code was uploaded to Server and Client, and a connection was established proving that the boards were working properly. Artemis Nano is more similar to Arduino boards and does not need any extra methods. Basic Serial port programming to Artemis Nano with Arduino IDE is possible. The Edge board can also be used, and as previously mentioned holding down button 14 and pressing reset were required to upload code to this board.

There are many important Arduino Nano 33 BLE specifications regarding this project. Board dimensions are 45 x 18 mm, which are exactly the same as those of Arduino Nano. The plug-in connector is Micro-USB which is different from that of Arduino Nano. Arduino Nano 33 BLE is developed from Arduino Nano with some added extra features and upgraded components. The processor is replaced in the Arduino BLE version with a more powerful processor nRF52840 from Nordic Semiconductors, which is later introduced. The Arduino BLE version operates only at 3.3 V while the original Arduino board also operates at 5 V [35]. This processor has low power modes, which are important in the project. Even the BLE version is improved from the original Arduino Nano for some reason it has a lower price. The ArduinoBLE library is compatible with this board that helps at the starting phase of code development. [34]

Various challenges can occur even with a familiar development environment. Sending data had some issues, including losing the connection between the boards. The connection hung if the program tried to use many characteristics, even they were not needed. The exact reason why multiple characteristics broke the connection were not found. Maybe they collided or tried to read at the same time. It was simple to evade by using one characteristic for all the values. This might not be the optimal solution if x, y, and z acceleration must be read at the exact same moment since reading characteristics can have a delay between reading operations. Other examples had separated several characteristics, but the surrounding program was different from the one in use. It is possible to use multiple characteristics but the bottom level program requires

modifications to have control over the characteristics. There was also an issue with reading the received data and the data did not print as expected. Random values came out when float numbers were desired. The data sending program used unsignedChar instead of float, giving random output when unsigned character variables got float values stored. After the fix, the print had no more issues.

5.4 Android mobile phone

Installing the nRFConnect app was a simple way to start BLE connection testing. An Android phone with the nRFConnect app worked as a slave or master without any coding. Having two developing boards with slave code and master code meant the possibility that any of them could be wrong and debugging connection issues were more complex. Using a phone as a master and developing slave code to board made testing and debugging simpler. That way made sure the slave code worked. The next step was to change the phone role to slave and develop a master code. The phone was able to connect to both master and slave, so now it should be possible to connect two boards. Later on, new problems appeared while connecting two boards together, but the environment with the phone was ready for debugging. In the phone app assigning UUIDs was not needed, but to connect two boards, they need to have the same UUIDs assigned. The board connection parameters are not necessarily right after being able to connect to the Android phone. From phone operating systems, only Android OS was used because of availability and previous knowledge. Also, other operating systems Apple iOS, Linux, and Microsoft Windows have gotten BLE support in earlier version updates. A similar application to nRFConnect for iPhone is LightBlue.

Developing with Android phone was easy to start with the ready-made app nRFConnect, but when it comes to reading more complex output, the app would need some modification. Making a new app to read multiple values and represent them with a user-friendly UI would be convenient. The phone could be the data receiver in the car for the user but driving and using a phone is not a suitable combination. A stationary monitor would be preferred in the target application.

5.5 Nordic BLE boards

Another BLE manufacturer Nordic has its boards and focuses on the BLE side, so developing with their environment and devices might be a beneficial choice. When developing in previously mentioned environments, the Nordic name and their example programs came up many times in the search results. Also, many of the example programs were found only for Nordic boards. Those programs might be modified to other

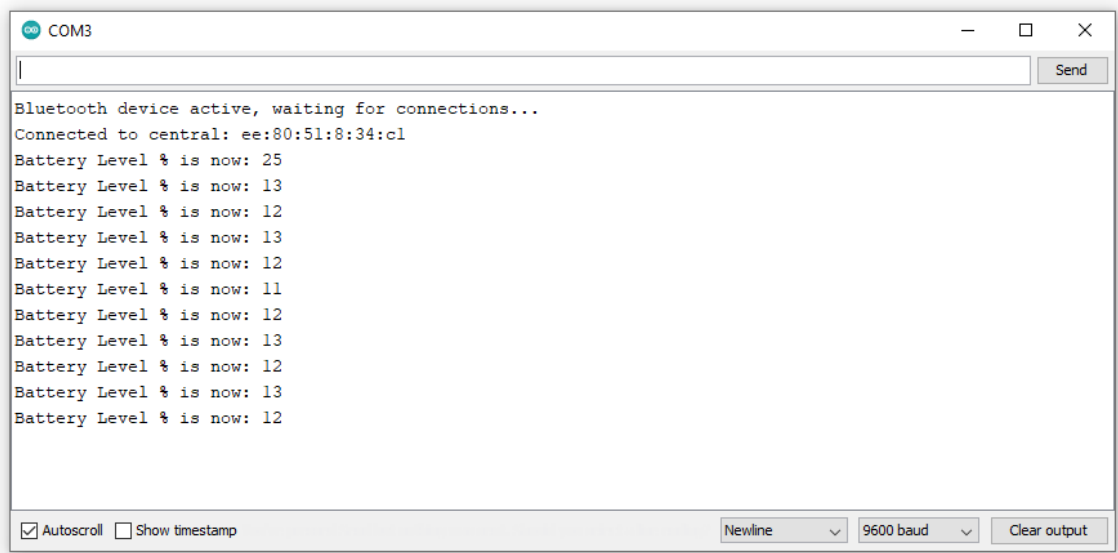
environments, but since Nordic usually has everything required, it can be a good option. Like the Nordic phone app, nRFConnect worked with other manufacturers' boards and was the first phone app to be mentioned when searching for one. The online version of the Mbed Studio did not have SparkFun Apollo3 boards, but they had the Nordic nRF51 DK board, for example. In many places, Nordic would have been more convenient. Nordic was not chosen in this project because development had started with SparkFun Apollo3 boards, so they were instantly available and project members had previous knowledge of using them. Arduino boards were chosen to have something to compare with and get started with a familiar development environment.

6. TRANSFER DATA OVER BLE CONNECTION

Testing of environments and boards is performed in this chapter. After selecting the right environment and suitable board, the device is programmed to transfer data. Set-up begins by running a simple test program on one board. The next step is to get two boards connected and to transfer data. The connection is required before other features can be used or tested. The program developed to establish the connection is explained in this chapter and the program code is added in the appendix section of the thesis.

Terminology related to establishing the BLE connection is explained in this section. The terms connectable and non-connectable are self-explanatory. The establishment of a connection is allowed to a connectable device and not allowed to non-connectable one. More importantly, it should be checked that the parameters in the program code are set to connectable. For the project, there is no reason to use the non-connectable setting, but it should be acknowledged as a possible reason that connection is not successful. Scannable and non-scannable are quite similar to connectable and non-connectable. It must be determined if the device can handle scan requests. In the advertising device, this is set to scannable. [36]

Advertising types are legacy, extended and periodic. Primary and secondary advertisement channels should be introduced parallel to advertising types. The number of channels used determines the amount of data that can be transferred, depending on the advertising type. Legacy advertising is the basic advertising type that uses primary advertising channels 37, 38, and 39 [24]. Extended advertising also adds secondary advertising channels 0 - 36 in order to transfer a higher amount of data. Some older devices do not support extended advertising. Having both legacy and extended advertising enabled ensures proper operation in most devices. Periodic advertising is used to send data in fixed intervals. This method can save energy as the device is not constantly advertising, and the connection is not on all the time. Periodic advertising has the same issue as extended advertising, all the older devices do not have support for it. That the support for periodic advertising is lacking was found out during research. For example, it was noticed that the Artemis DK board does not support it. [36]



The screenshot shows the Arduino IDE Serial Monitor window for COM3. The output text is as follows:

```
Bluetooth device active, waiting for connections...
Connected to central: ee:80:51:8:34:c1
Battery Level % is now: 25
Battery Level % is now: 13
Battery Level % is now: 12
Battery Level % is now: 13
Battery Level % is now: 12
Battery Level % is now: 11
Battery Level % is now: 12
Battery Level % is now: 13
Battery Level % is now: 12
Battery Level % is now: 13
Battery Level % is now: 12
```

At the bottom of the window, there are controls for 'Autoscroll' (checked), 'Show timestamp' (unchecked), 'Newline' (dropdown), '9600 baud' (dropdown), and 'Clear output'.

Fig. 7 Battery level service Arduino IDE Serial Monitor output

Battery service pushes data from server to client without request. The *notify* command in the server is used to create this effect that data is pushed every time the data changes. The battery level is not coming from the real battery. The number is made-up to have a certain starting point and merely updated over time, simulating changing data. The purpose of the battery service program was to test the BLE connection and get any data transferred over the connection. Everything succeeded that is shown in Figure 7 Serial Monitor output.

Battery service could be implemented to the accelerometer program introduced in Subchapter 6.2. The service name changed to describe new functions of the service renaming it sensor data service. Removing Battery service features is not necessary by inserting battery data and accelerometer data to separate characteristics, allowing to data transfer rate to be controlled for different characteristics. The program can send battery level data for instance every 5 minutes and on request, rather than battery level popping up 900 times per minute like accelerometer data. The research project does not need a battery level since an energy harvester powers the system. However, the amount of electrical energy the harvester produces might interest the user. Also, modifying battery service to show energy harvester output could be beneficial in the testing phase for debugging.

This project represents embedded system development where programming is part of the project. Basic coding from studies and hobbies was necessary since the coding part takes most of the time and in embedded systems uploading the code to the board takes the second-longest since interruptions happen on serial port communication. Changing between boards, development environments and programs for an unknown

reason, even the same settings sometimes resulted in unsuccessful upload. Fixing and building everything again takes some time. In personal experience, compiling code to a PC program versus compiling to a board results, many times, in different kinds of errors, and the amount of time to upload code again takes longer.

6.1 Arduino test and debugging code

Arduino Sketch programming uses a C++ coding dialect. The Arduino code “setup” function runs once at the beginning and the “loop” function repeatedly runs the code. Libraries and header files are added with the “include” command. Variables are assigned by writing variable data types and giving a name describing their use. Information about run time events is printed to Serial Monitor with “Serial.print” and its variation commands. The serial communication baud rate is set to 9600 or 115200, which is set in the code and suits the BLE board processors.

After checking that the programming environment and boards were functioning, the first task was to generate a BLE connection between the boards and send data over the connection. All source codes are included at the end of the thesis as an appendix. Program 1 is uploaded to SparkFun Apollo3 Edge board, assigning it to be the central device. After initialization, the central device starts to scan for peripherals. After finding the peripheral, it establishes a connection to it. Arduino Nano 33 BLE is running Program 2, assigning it to be the peripheral device. Program 2 is advertising its service that the central device can subscribe to.

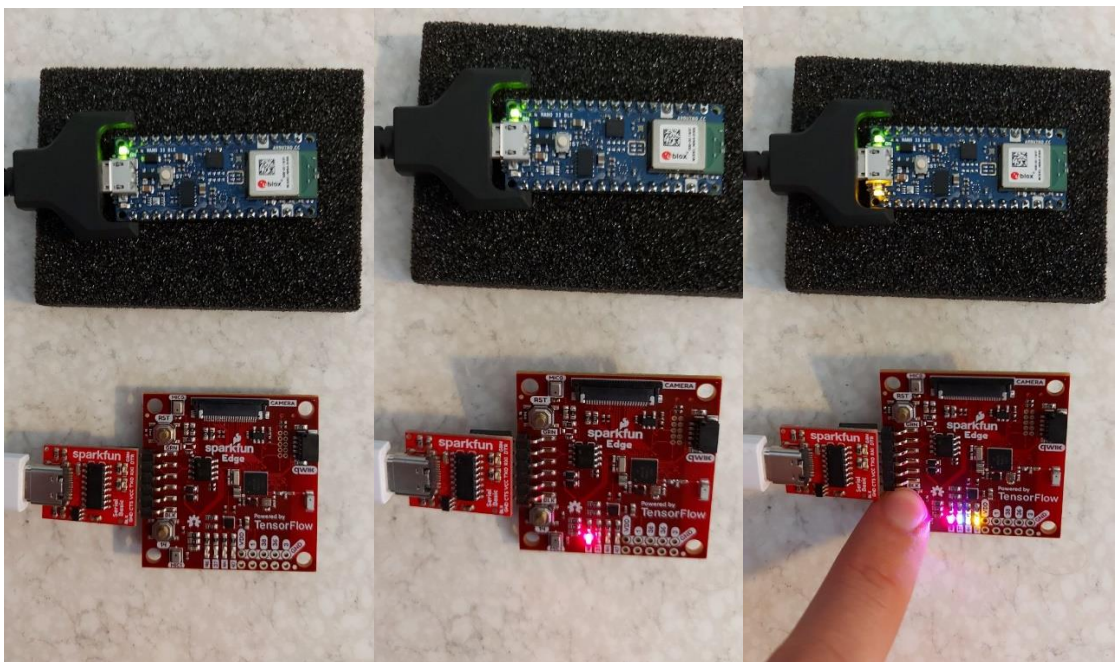


Fig. 8 *Arduino Nano 33 BLE as the blue colored board above and SparkFun Artemis Edge as the red colored board below. On the left operation started, disconnected, or turned off. In the middle connection is established. On the right button pushed and LEDs are lighted up on both boards.*

Only the green power LED of the Arduino is on when boards are powered up, and they are not connected. This is shown in Figure 8 left section. In the prototype system, Arduino is used as a peripheral device powered by a harvester, so the energy consumption has to be minimized; for example, the power LED should be turned off. The Edge board is the central device that the user controls, and the power LED should be turned on. For testing purposes, these changes were not necessary. The focus is on connection and sending data wirelessly, shown in the middle and right section of Figure 8. The Edge board red LED lights up when a connection is established. After the connection is established, the user pushes the Edge board button shown in Figure 8 right section. The pushing button lights up all four LEDs of Edge to indicate that the button is pushed. More importantly, a button push at the central device sends a command to the peripheral wirelessly lighting up yellow LED from Arduino.

There are several reasons why two different manufacturer boards were used. Only the Edge board had another push button in addition to the reset button. That made this example and physical testing possible. The program was coded and uploaded with Arduino IDE, so having another manufacturer board suitable for the environment allowed more flexibility, and verifying early on that a BLE connection can be established with different boards since multiple types of boards were desired to be tested in the project.

In order that everything can be recreated and the issues can be avoided, next step is undergoing a detailed review of the code, starting from establishing the BLE connection and controlling the LED in Program 1. The first issue that came across was that the program did not run without opening Arduino IDE Serial Monitor because of line 42:

```
while(!Serial);
```

which waited Serial Monitor to open before progressing with program execution. The line was first helpful for debugging to see the initialization and start of the program. Later, removing that line permitted the Arduino to run every time it was launched. That was necessary for proper operation when the device was plugged into an external power source where Serial Monitor is unavailable.

Libraries are included at lines 19 - 21 to have access to Edge board LEDs and buttons. The following lines:

```
am_hal_gpio_pinconfig(AM_BSP_GPIO_BUTTON14, g_AM_HAL_GPIO_INPUT);
```

```
am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_RED,g_AM_HAL_GPIO_OUTPUT_12);
am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_BLUE,g_AM_HAL_GPIO_OUTPUT_12);
am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_GREEN, g_AM_HAL_GPIO_OUTPUT_12);
am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_YELLOW, g_AM_HAL_GPIO_OUTPUT_12);
```

configure button and LEDs. The Edge board has a button at pin 14 and four LEDs. If using a different board, pins should be configured to fit the board specifications. Changing “config” to “set” lights up LEDs or “clear” shutting down the LED.

```
am_hal_gpio_output_set(AM_BSP_GPIO_LED_COLOR);
am_hal_gpio_output_clear(AM_BSP_GPIO_LED_COLOR);
```

The text “COLOR” should be changed to indicate which LED is controlled. The LEDs are addressed by their color name, for example yellow.

After the LEDs are configured, it is possible to start the scanning in the setup function. After finding the peripheral, scanning is stopped to save energy and for this test, connecting to one peripheral is enough. Function “controlled” is called next when a connection is established with the peripheral, checking from Serial Monitor that attributes are discovered and LED characteristics are found to continue with the test. Characteristic must enable *write* for sending a command to peripheral board. Those commands are triggered with the push button by writing a byte to control the LED in the peripheral.

The Program 2 peripheral code is uploaded to Arduino. Peripheral advertises its service and waits for connections. Before getting into more detail about the code functioning, the following line:

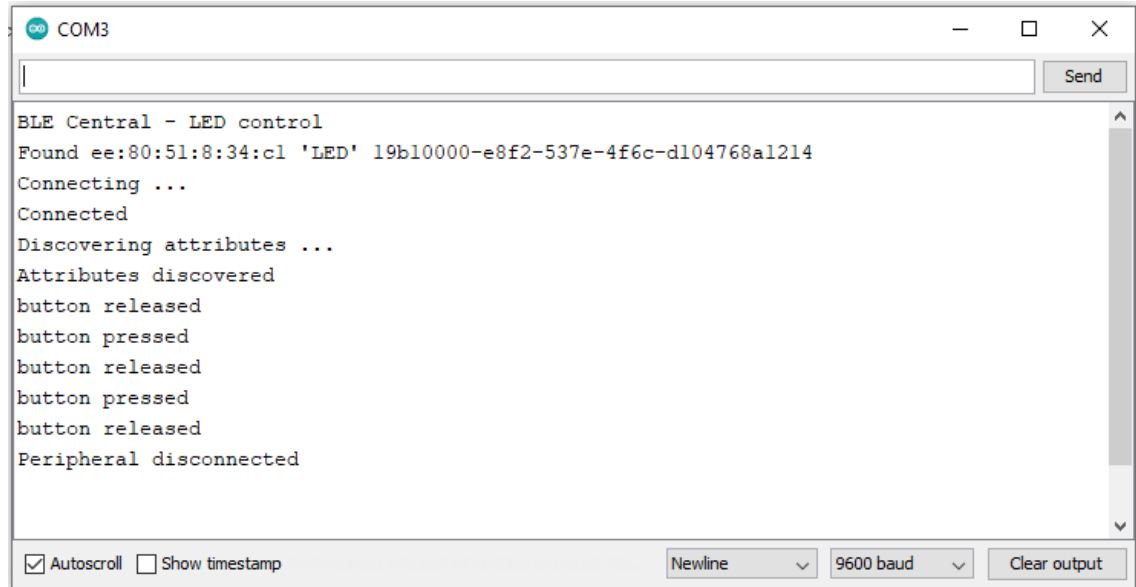
```
BLEByteCharacteristic switchCharacteristic("19B10001-E8F2-537E-4F6C-D104768A1214", BLERead | BLEWrite);
```

conveys what should be carefully checked and understood. Configuring characteristics have the type of transferred data, characteristics UUID, and data transfer modes. The first problem was that modes were *read* and *notify* (BLERead|BLENotify). *Notify* makes the data to be sent continuously, but in our case, the need is to request the data only at a certain time. The *write* command was the right choice to get the request. Since the characteristics were originally not configured, a writable error happened. After changing *notify* to *write*, the values were still not correct since at line:

```
BLEByteCharacteristic
```

the word “Byte” went unnoticed. Between the words “BLE” and “Characteristic” should be for example “Float” or “Int” to get the requested values. Before finding this issue, the program managed to send bytes over to come out correctly (since the data was defined to be in bytes), so code was built to send bytes even after the problem was found.

When central is connected to Program 2, it waits for the value to be written in order to control the LED. When a device is disconnected, it continues advertising automatically. The peripheral device might not be physically accessible to the user. Establishing a new connection remotely is essential in that case.



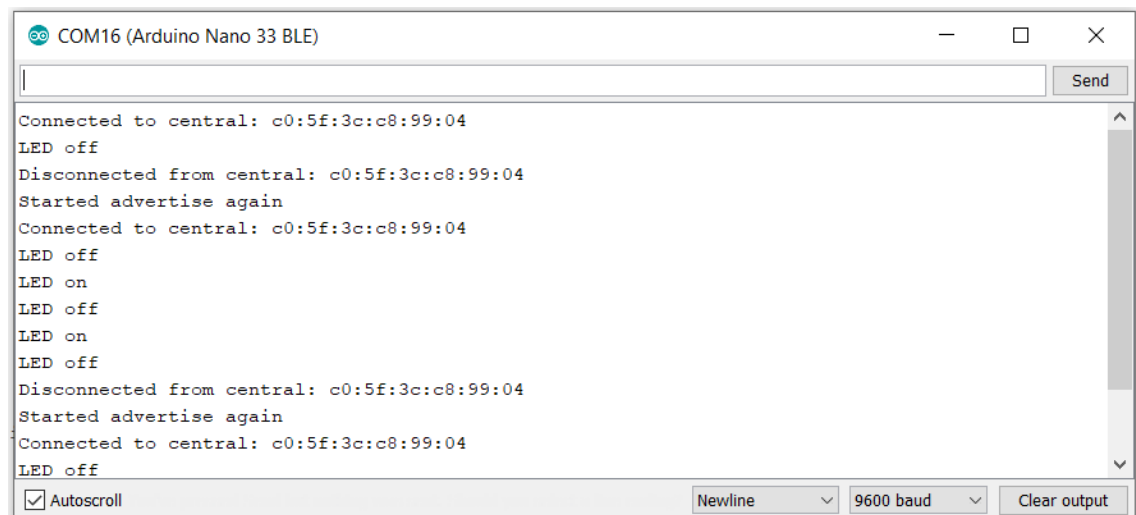
```

COM3
BLE Central - LED control
Found ee:80:51:8:34:c1 'LED' 19b10000-e8f2-537e-4f6c-d104768a1214
Connecting ...
Connected
Discovering attributes ...
Attributes discovered
button released
button pressed
button released
button pressed
button released
Peripheral disconnected
Autoscroll Show timestamp Newline 9600 baud Clear output

```

Fig. 9 Program 1 BLE Central device Serial Monitor output

The Program 1 Serial Monitor output is in Figure 9. Following the output in Figure 9 first the central device is started, and it finds a device and displays the MAC address of the device, advertised service LED, and service UUID. Establishing a connection and discovering attributes are relayed to the user. The next output is user-controlled after the first “button released” print, which informs the user that the button was not pushed down at the beginning. After the user pushes and releases the button, it is printed to the output. The connection is lost if the other device is turned off or too far out of range. The code starts scanning, trying to reconnect if reset is pushed or the program is started again.



```

COM16 (Arduino Nano 33 BLE)
Connected to central: c0:5f:3c:c8:99:04
LED off
Disconnected from central: c0:5f:3c:c8:99:04
Started advertise again
Connected to central: c0:5f:3c:c8:99:04
LED off
LED on
LED off
LED on
LED off
Disconnected from central: c0:5f:3c:c8:99:04
Started advertise again
Connected to central: c0:5f:3c:c8:99:04
LED off
Autoscroll Show timestamp Newline 9600 baud Clear output

```

Fig. 10 Program 2 BLE peripheral device Serial Monitor output

Figure 10 is peripheral device output which shows connection status and LED status. The peripheral device will start advertising again after losing the connection. Between starting advertising again and being connected to central, Program 1 is restarted with a reset button, whereas Program 2 continues operation by advertising and establishing a new connection when another device is found. LED status is updated when the command is received from the central device.

6.2 Accelerometer program to measure acceleration

In the following programs, the intention is to go through new features, focusing on specific areas instead of the basics. The purpose of the sensor on the car tire is to measure speed, velocity, or acceleration. The Arduino Nano 33 BLE Sense board has the LSM9DS1 chip with a 9-axis IMU unit. The chip has a 3-axis accelerometer, 3-axis gyroscope, and 3-axis magnetometer. Velocity can be calculated from accelerometer data. In Figure 11, the acceleration data is demonstrated without modifications. Sensor output is x, y, and z axis acceleration that must be modified to get the speed. The sensor is rotating in the car tire, so acceleration from that movement is used to measure the tire speed forward.

$$a = \frac{\Delta v}{\Delta t} \quad (1)$$

$$\Delta v = a \times \Delta t \quad (2)$$

In formula 1, acceleration is calculated from the difference in speed divided by the difference in time. If acceleration is already known, flipping acceleration to the other side allows measuring the speed. [37][38] The formulas might turn out to be useful depending on what information and values are exactly requested from the car tire. For now, acceleration is the only value wanted in the research. We are more interested in transferring that data since the final use case is unknown.

```

COM7
Peripheral disconnected
Found dd:34:d5:4d:21:5a 'Accelerometer' ae45cfc5-152f-487e-afb5-3f30072a0f90
Connecting ...
Connected
Discovering attributes ...
Attributes discovered
Choose x, y, z or all:
y
How many values enter number:
4
Y value: -0.05
Y value: -0.05
Y value: -0.05
Y value: -0.05
Choose x, y, z or all:
all
How many values enter number:
3
X value: 0.15 Y value: -0.05 Z value: 0.98
X value: 0.15 Y value: -0.05 Z value: 0.98
X value: 0.15 Y value: -0.05 Z value: 0.98
Choose x, y, z or all:

```

Fig. 11 User inputs *x, y, z, or all* to get requested values as an output

The peripheral device is the Sense board, which has a sensor to measure the acceleration. The Figure 11 Serial Monitor output is from the central device that requests the values. For example, if a user writes “y” from central, the *write* command is sending byte value to inform the peripheral which axis acceleration data is requested. The peripheral *read* command is used to get that data, and then corresponding values are written back for central to show the data to the user.

6.3 Mbed characteristics update

The function of the program was to update characteristics at the Mbed Studio platform. The test program had a clock service that offered hours, minutes, and seconds as characteristics. Updating these values to Serial Monitor would indicate that the connection is complete. This part had the most issues with establishing the connection.

Next, steps needed to be clarified as to how the connection is supposed to be established. Factory settings have set the MAC addresses of both boards the same. This was discovered by using the phone app nRFConnect, which shows the device address.

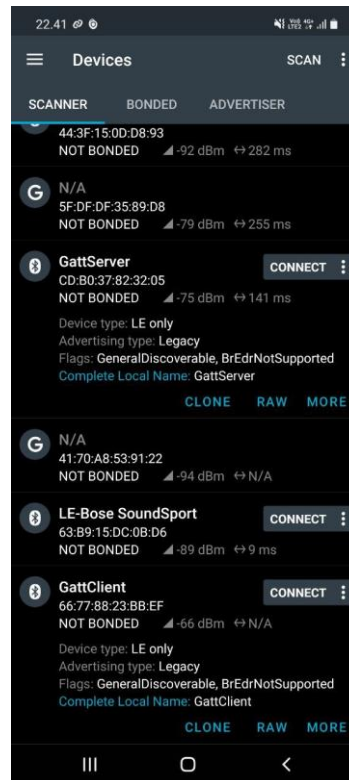


Fig. 12 Phone app nRFConnect showing GattServer and GattClient

Only Artemis DK supports the Mbed Studio platform from the boards used in this project. Changing the board was not possible, so the MAC address had to be changed since it is required to be unique. Searching solutions lead to modifying file `hci_drv_apollo3.c` in the Mbed operating system files, adding a definition for custom Bluetooth address

```
#define AM_CUSTOM_BDADDR
```

and then a few extra lines in the file to create new address numbers. In Figure 12, it is shown that GattServer and GattClient have different MAC addresses. Previously both boards had this 66:77:88:23:BB:EF factory address, and only one would show up at a time in the nRFConnect app even both were turned on. Now the GattServer code is modified to use a custom address which is shown in Figure 12. Still, the boards did not connect to each other, meaning something else is not working.

EventQueue class is used in the program to schedule events. Event points to the function that is executed when the corresponding event is first in the queue. The event execution priority is based only on time. [26] In the example program where the connection issue is present event queue is used. The event that calls the “onAdvertisingReport” function is the last event executed. The print command was useful for tracking events until the last event was found when something else happened that could not be tracked with the same method. Inside “onAdvertisingReport” function connect is called, which should put the next event in the queue. Afterwards the

“onConnectionComplete” function would indicate that the connection is successfully established. This step is never executed in the program, and the “onConnectionComplete” function is not reached.

```
error = _ble.gap().connect(  
    event.getPeerAddressType(),  
    event.getPeerAddress(),  
    connection_params  
);
```

Without deep programming knowledge, the author cannot create the whole program using something other than EventQueue in a new environment. The process of trying to remove EventQueue class and execute the code step-by-step to the connection function failed. EventQueue and BLE library are connected to each other in the example inflicting BLE features to break in the code while removing EventQueue from the program code. The BLE library is also included in the example, but other BLE libraries were not found for Mbed Studio. Without the BLE library programming would be even harder.

The result is out of reach since the exact reason for the issue could not be pinpointed. The environment and program managed to connect one BLE board and phone, but testing with two boards can be something the developers have not tried before. The goal was to develop this program code further with a working environment. Unfortunately, something on the way is not working. For this reason, the whole research direction changed more to a comparison between environments and boards rather than developing the application for BLE boards.

7. RESULTS AND COMPARISON

The research found differences between the tested development environments. As a result, some of the researched environments and boards stand out as more suitable for this project. However, all the studied development environments performed well in their specific areas, which might be important for other projects.

7.1 Comparison of development environments

The learning curve, compiling speed, and overall getting everything working with existing information were the main differences between the environments. Preferences and more suitable functionalities were found during the research. These results should help to choose recommended environments and board combinations for different applications. For example, if a single one of the researched environments does not seem appropriate, in that case Nordic BLE development sites should be checked.

Arduino IDE would have been the most straightforward to use without previous knowledge since everything is done exactly as it is learned the first time. The board must be found in the board manager, and if it is not in there, it should be added by the user. Users can include SparkFun boards in Arduino IDE by manually adding a definition for the boards to be found in the board manager. Choosing the right bootloader can be done by trial and error as there are only a few options. Plug in the board and choose the correct port. Including needed libraries and testing example code from the library leads to the starting point. Arduino IDE does not require too many steps, and if something goes wrong, the problem is effortless to locate with Arduino IDE error messages.

Mbed Studio was fairly similar to set up and used just as Arduino IDE. Getting the first program to run in Mbed Studio still took more time than Arduino IDE. A totally new environment of course made it harder. Not enough available information and a slightly more complicated project setup took the extra time. Mbed Studio creates a new copy of Mbed OS for every new program but adding programs from a separate source might not include Mbed OS. In the beginning, the most important thing was to add Mbed OS for every program in the Mbed Studio. Linking program to use a shared instance of Mbed OS saves disk space. Otherwise, every new program uses 1 GB of memory as a default, a mistake realized when a few test programs took 24 GB of disk space. [26]

In the end, Mbed Studio cannot be recommended, at least with SparkFun boards. Solving connection hanging issues took way too much time, and even if the

problem was not about the environment or the boards, there was not enough information available from those. BLE and Mbed Studio are relatively new, so all kinds of references are needed in the research. Usually, it is possible to find some demo projects and examples, but such was not available this time. Even the examples on the product developer page did not work. It was hard to get into the environment when expecting that an example can be seen before getting deeper into developing. Development had to be started from scratch. The work began with fixing the example code without knowing if the problem was in the code since the environment and board can be the source of the issue in embedded systems. From the top level, it seemed that everything could be done in Mbed Studio, but for some unknown reason development to a direction where Artemis boards are supported in Mbed OS is on a break or not progressing at the moment.

Ampiq Apollo3 SDK has significant differences compared to previous environments. Ampiq Apollo3 SDK does not have IDE enabling to edit code in the preferred editor. This is an advantage for some developers, but for those without preference or knowledge about many editors it is disadvantageous. After setting up the editor, several further issues with the environment still exist. Several other tools are required to be installed and set up for a functioning environment. Installing the required tools caused issues, as noted in Chapter 5. The disadvantage of Ampiq Apollo3 SDK is setting up several prerequisites for the environment and then understanding the intended usage. Yet, using effort in the time-consuming configurations might end up being beneficial if one manages to set up a fast and functioning environment.

7.2 Suitable boards

Multiple different boards were tested out at the same time when different environments were tested. Specific boards had more information and guides for certain environments, leading to how they were first chosen. Later many boards were cross-tested with other environments. One takeaway is that some of them showed up in the target list but were not supported yet, or uploading code was not directly possible. All the different boards used on the project are listed in Figure 13.



Fig. 13 Boards from left to right: Arduino Nano 33 BLE and Arduino Nano 33 BLE Sense, two Artemis Nano, two Edge and two Artemis DK boards.

Arduino IDE supports Arduino Nano 33 BLE boards without problems. Arduino IDE recognizes the COM port immediately after plugging in Arduino boards. With boards from other manufacturers, the support varies. The environment might recognize other boards, but that does not help if other support does not exist. Bootloader for Artemis DK was missing in Arduino IDE even after adding Artemis DK with the board manager. Drag-and-drop uploading code to the board enables Artemis DK to be used with Arduino IDE even though they are not used together in this project. Artemis Nano had support in Arduino IDE, probably because it is very similar to Arduino Nano. The Edge board was not tested in this environment since two different manufacturers' boards already worked and gave all functionalities needed. Also, Arduino boards are only used with Arduino IDE because other environments would not give any extra opportunities for these boards.

Arduinos are commonly seen as non-professional boards. Many times they do not come up in companies for professional use. Arduino is an overall well-functioning board but it is not particularly excellent in any specific area except the simplicity of development and getting started. Similarly, in this project, Arduino was almost suitable for the project, but the size of the board became an obstacle. Artemis Nano do not have extra components as Arduino, and even the tiny BLE module in Artemis Nano can be used alone. Size and the weight difference were some of the main reasons for the Artemis module being used in the final product.

Mbed Studio had all three SparkFun boards listed in board targets. All three SparkFun boards, Artemis DK, Nano, and Edge can be found by adding SFE (*SparkFun Electronics*) in front of the target name.

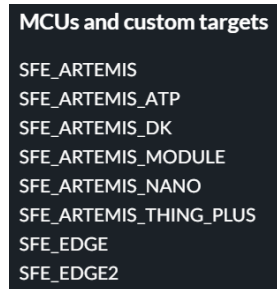


Fig. 14 Mbed Studio SFE target list

Even though several SparkFun boards were listed on the Figure 14 target list, the Mbed Studio did not recognize them all. As a result, only plugging in Artemis DK and choosing SFE_Artemis_DK from the list allowed the target to be connected.

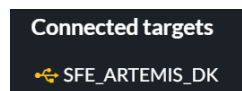


Fig. 15 Mbed Studio connected target

Figure 15 shows a yellow mark in front of the board name, which indicates that the board is connected and selected as an active target.

The Ampiq Apollo3 SDK Edge board was favored because it has an accelerometer included and a suitable example code to test it. Other SparkFun boards lacked examples which led to the Edge board being more convenient to develop. The general guide to using the Ampiq Apollo3 SDK environment featured the Edge board as the first device to be programmed with.

The name Artemis Development Kit indicates that the board is used for development. The size of the board is larger than many applications need because the board layout has many pins and areas for different developing directions. Some extra features are consuming electricity even when they are not used. Leaking energy is unacceptable for low power applications where all the energy consumption matters. The board also includes debugging spots for measuring voltage and other variables, which are excellent for testing but not for the actual application.

Overall, the most suitable board for the project is the Artemis Nano. Necessary features such as BLE connectivity and low power features were present, and the board does not include unnecessary characteristics that would waste electricity. Artemis Nano had the most advantages in size, number of features, and developing possibilities compared to other tested boards. Artemis Nano was the tiniest one of the boards and could be made even tinier by separating the BLE module and adding a separate accelerometer sensor. Tiny size and low weight are preferred when the device is installed

inside a car tire. Artemis Nano was tested with all the environments used in the project, and it functioned in the majority of them without any problems.

7.3 Use BLE for data transfer

BLE packet size is limited to 255 bytes, and 247 bytes is recommended for BLE [39]. When the amount of data is larger, the data is split with the “onAttMtuChange” function. EventHandler calls this function. Payload size is limited to legacy advertising max size before Bluetooth 5; versions of Bluetooth 5 and later allow larger payload sizes. Keeping compatibility with older versions of Bluetooth, it is suggested to test whether the device supports a larger payload or only a legacy size. [26]

Privacy features need to be enabled and included separately to have secure data transfer [26]. Otherwise, any BLE device can interfere with data transfer and see all the information on the board. The device in the project is apart from the main system, and the data do not contain sensitive information. Security features must be enabled in the future, but they are not the main concern for now.

7.4 Applications under development and future possibilities

The research information leads to future applications that benefit from energy harvester and BLE data transfer. Specific application needs are an existing energy source, isolated location, and information transferred wirelessly. Also, it may be one of those needs when the energy harvester or BLE technology turns out to be useful. This chapter gives considerations for other applications that are not introduced in the research, but the technology introduced can be used as pre-information. It is especially important to take a broad-minded view with rather new technology that has possibilities even though it is not commonly used.

BLE can replace multiple other wireless technology applications or be an alternative option. Wireless applications have been around for many years now, but energy harvesters as a power source represent a quite new technology opening new inventions. Low power consumption features should be used to benefit from BLE in particular. The device that is connected to the mains could also use BLE technology. Saving energy in any application should be beneficial to the user and to nature. For example, saving energy with the following everyday example. Wireless switch sockets are connected to a wall plug where remote control sends a signal to that switch to turn on or off the connected device. These switches could use BLE to save energy since they are consuming electricity all the time. If this is not convincing, the device roles can be

changed and do a little modification. The switch socket is now alone connected to a wall plug. A BLE transmitter is the switch and connected to other devices in the house, for example, an aquarium lamp. The timer controls the switch and turns the lights off at night. The lamp needs a wireless receiver, and the wall plug has a wireless transmitter connected. The BLE transmitter at the wall can sleep the whole day and keep on tracking the time to wake up the device when it is time to switch the lights on. The device connected to a system also needs a BLE receiver that can use a battery, so BLE technology can significantly save electricity, making the battery last for longer or consume less electricity when plugged into a wall. The whole application might be unnecessary but serves as an example.

The energy harvester was tested in the project, shaking it by hand. The result was quite unstable output. Still, it could be an extra feature added to the remote control that shaking the remote-control charges the battery, so it lasts longer, or possibly an empty battery can be charged, making the device momentarily functional. Applications turned on momentarily can utilize this type of energy harvesting in places such as a home where energy sources are available only at certain times when a human does some activities independently or with other devices. Combining this remote control with BLE low energy features improves the performance. With BLE, less energy is needed to power up the device. The project system receives data wirelessly and sends commands to choose the data received. The commands trigger the other device controls. BLE is suitable for close-range, remotely controlled applications.

The project vehicle tire sensor can be developed for further use. One sensor measures multiple characteristics such as tire temperature and pressure. Acceleration measurements, together with speed measurements, can provide more accurate vehicle speed. [40] Later BLE use can be broadened with more sensors and commands. Newer cars have monitors in the center console with a built-in BLE receiver to display the information received from sensors. The driver can add custom lights to the car and connect them to the system with a BLE connection, allowing control of the custom lights from the original car display. Monitoring the car display is convenient since the phone should not be used while driving. When a car is stationary, the connected phone can control the devices. The lights were only one example of added custom objects.

BLE can be used to monitor machines, living things, or hazardous substances. For example, factories can monitor their machines, or farmers can monitor tractors, or any machine that needs monitoring. BLE can be built into the monitored device or added later to have wireless monitoring available. Other wireless protocols might prove to be more suitable for mesh networking but BLE can be used at least on a small scale if the low

energy aspect is an advantage, for example, monitoring a well or a tank that has not been checked inside for years, but the information about surface level would be helpful. A BLE device can be powered by battery for many years and report the tank surface level wirelessly. If the tank is filled with a hazardous substance, monitoring is running all the time and consuming electricity, and at some point, the battery will run out. A less critical application example is a fuel tank that indicates the surface level only when it is near empty, so the user has time to order a refill and never totally runs out of fuel. With this kind of consumer product, when tanks are monitored and checking is done from time to time, BLE device features can be utilized.

A mini BLE beacon connected to humans could be used to extend and strengthen a network, schedule, monitor daily activities, alert about natural disasters, locate people and use it in commercial purposes. The beacon can also be connected to animals. First of all, the reason especially in this context that the BLE beacon is connected to a human is that humans or animals can be the energy source for energy harvesters. This technology is in progress since wearable devices are not simple to design. They must be as indistinguishable as possible to the user and unnoticeable to people around while still tolerating high stress in daily use. Harvesting energy with a kinetic energy harvester from a human without interrupting movement could prove hard to execute. The harvester could be built in a way that recalls the external exoskeleton. Covering the human body with solar panels might be as hard as a kinetic harvester. This is being researched to extend and strengthen the network to reach people in a natural disaster or for some other important reason. Satellites and phones are not totally certain to reach people. [41] Privacy restricts collecting information from human behavior, but if it is not an issue then locating people for commercial use permits advertising and locating shops by the information BLE beacons provide. [42]

Human smart shoes already have had some research done, but it is an interesting question if they will ever be in a store. They utilize the piezoelectric effect to harvest energy from walking or running. This way shoes, are battery-free, and housing for the system only thickens the bottom of the shoe. [43][44] A thick shoe bottom can go as a fashion item, making it truly indistinguishable wearable electronic. Smart shoes have potential and enable a way to test energy harvesters in wearable electronics.

8. CONCLUSIONS

The thesis focused on energy harvesters, Bluetooth Low Energy, and the systems around them. A prototype of the system was introduced, and similar examples of low-energy communication with battery-free systems were presented. In order to transfer data with the BLE protocol, certain requirements had to be met. A few example programs were included to explain connection establishment and find more about BLE communication features. After introducing development environments and boards, they were compared to find a suitable combination of the board and environment for the specific project.

Energy harvesting was used to explain the context and applications in the project. Energy efficiency was related to the thesis to be able to use an energy harvester as a power source. Different energy harvester technologies were explained to have a broader view of the operation. The research helped to choose convenient operation environments for certain harvester types. Understanding the theory behind an electromagnetic induction harvester shows its suitability for the project environment later in the prototype chapter. Vibration movement was the essential requirement for this type of harvester. Energy harvesters are often compared to batteries since they are usually supposed to replace battery devices. New applications were also possible with energy harvesters, but some issues like being an unstable power source influenced operation.

The differences between Bluetooth and Bluetooth Low Energy were first explained, and alternative options for low-energy wireless protocols were also introduced. BLE was compared to find out the sector where BLE performs better or worse than other protocols. BLE-specific features as GattServer and GattClient were explained to know which device receives data and sends data. These were needed in the coding phase to get the data with commands to the right device. The main reason for using BLE is the energy efficiency features. Setting up sleep modes and ways to lower energy consumption were indicated.

The background was given for setting up an environment to develop BLE programs. All the required software for each environment was listed. Some environments were more flexible and had more options. That was also a disadvantage when several options made the environment complex. Choices were made to find a working environment, and the advantages and disadvantages came up during the process. The most suitable board for each environment was found from guides and examples, but

other boards were also tested if they seemed to be suitable for the environment. Testing multiple options gave answers to optimal board and environment choice. As previously with environments and boards, alternative options were also used for testing, such as Nordic and Android.

In Chapter 6, testing of the programs was performed to test the board operation with example programs and create the program suitable for the project requirements. More information about the advertising and scanning process was in this chapter. Debugging programs started by running multiple examples, such as the Battery service. Successful testing setups were made to test the BLE connection and send data. The main portions of program code that affected BLE communication were shown in the chapter. The project prototype used a program to send data from the accelerometer sensor. An example program for accelerometer data transfer was included and explained. So far, the programs have been made with Arduino IDE, and the plan was to continue with a new environment, Mbed Studio, to develop the main program. Explaining the issues that came up in chapter Mbed characteristic update. Because of these issues, options were to develop the program with a different environment, start afresh with Mbed Studio or compare options by testing multiple approaches to develop programs.

In the last chapter, the comparison was taken further. The knowledge received from the comparison was used to choose situations suitable for a specific board and in which environment it most probably functions. A closer look was taken at Mbed Studio functionalities at the same time it was compared with other platforms. BLE development and energy harvester utilization can lead to new applications, and some examples were included in the last subchapter with future possibilities.

APPENDIX

```

2      /*
3         LED Control wirelessly triggered with push button
4
5         This example scans for BLE peripherals until one with the
6         advertised service
7         "19b10000-e8f2-537e-4f6c-d104768a1214" UUID is found. Once
8         discovered and connected,
9         it will remotely control the BLE Peripheral's LED, when the
10        button is pressed or released.
11
12        The circuit:
13        - SparkFun Apollo3 Edge
14
15        You can use it with another board that is compatible with this
16        library
17    */
18    #include <ArduinoBLE.h>
19    #include <am_mcu_apollo.h>
20    #include <am_bsp.h>
21    #include <am_util.h>
22
23    // variables for button
24    uint32_t pin14Val = 0;
25    uint32_t oldButtonState = 0;
26
27
28    void setup() {
29        Serial.begin(9600);
30        am_hal_gpio_pinconfig(AM_BSP_GPIO_BUTTON14,
31        g_AM_HAL_GPIO_INPUT); //Button config
32
33        am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_RED,
34        g_AM_HAL_GPIO_OUTPUT_12);
35        am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_BLUE,
36        g_AM_HAL_GPIO_OUTPUT_12);
37        am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_GREEN,
38        g_AM_HAL_GPIO_OUTPUT_12);
39        am_hal_gpio_pinconfig(AM_BSP_GPIO_LED_YELLOW,
40        g_AM_HAL_GPIO_OUTPUT_12);
41
42        //while (!Serial); //For debugging purposes start when Serial
43        monitor is opened
44
45        // initialize the BLE hardware
46        BLE.begin();
47
48        Serial.println("BLE Central - LED control");
49
50        // start scanning for peripherals

```

```

52     BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214");
53 }
54 void loop() {
55     // check if a peripheral has been discovered
56     BLEDevice peripheral = BLE.available();
57
58     if (peripheral) {
59         // discovered a peripheral, print out address, local name,
60         // and advertised service
61         Serial.print("Found ");
62         Serial.print(peripheral.address());
63         Serial.print(" ");
64         Serial.print(peripheral.localName());
65         Serial.print(" ");
66         Serial.print(peripheral.advertisedServiceUuid());
67         Serial.println();
68
69         if (peripheral.localName() != "LED") {
70             Serial.println("Error: Peripheral name is not correct");
71             return;
72         }
73
74         // stop scanning
75         BLE.stopScan();
76
77         // controlled function is called
78         controlled(peripheral);
79         // returns if connection is lost or cannot connect
80     }
81
82     void controlled(BLEDevice peripheral) {
83         // connect to the peripheral
84         Serial.println("Connecting ...");
85
86         if (peripheral.connect()) {
87             Serial.println("Connected");
88         } else {
89             Serial.println("Failed to connect!");
90             return;
91         }
92
93         // discover peripheral attributes
94         Serial.println("Discovering attributes ...");
95         if (peripheral.discoverAttributes()) {
96             Serial.println("Attributes discovered");
97         } else {
98             Serial.println("Attribute discovery failed!");
99             peripheral.disconnect();
100            return;
101        }
102
103        // retrieve the LED characteristic
104        BLECharacteristic ledCharacteristic =
105        peripheral.characteristic("19b10001-e8f2-537e-4f6c-
106        d104768a1214");
107
108        if (!ledCharacteristic) {
109            Serial.println("Peripheral does not have LED
110            characteristic!");

```

```

110     peripheral.disconnect();
111     return;
112 } else if (!ledCharacteristic.canWrite()) {
113     Serial.println("Peripheral does not have a writable LED
114 characteristic!");
115     peripheral.disconnect();
116     return;
117 }
118
119 while (peripheral.connected()) {
120     // while the peripheral is connected
121     am_hal_gpio_output_set(AM_BSP_GPIO_LED_RED);
122
123     // read the button pin
124     am_hal_gpio_state_read(AM_BSP_GPIO_BUTTON14,
125 AM_HAL_GPIO_INPUT_READ, &pin14Val);
126
127     if (oldButtonState != pin14Val) {
128         // button state changed
129         oldButtonState = pin14Val;
130
131         if (pin14Val) {
132             Serial.println("button released");
133             //Shuts down all Edge board LEDs to demonstrate that
134             button is released
135             am_hal_gpio_output_clear(AM_BSP_GPIO_LED_BLUE);
136             am_hal_gpio_output_clear(AM_BSP_GPIO_LED_GREEN);
137             am_hal_gpio_output_clear(AM_BSP_GPIO_LED_YELLOW);
138
139             // button is released, write 0x00 to turn
140             // the LED off at peripheral device
141             ledCharacteristic.writeValue((byte)0x00);
142         } else {
143             Serial.println("button pressed");
144             //Lights up all Edge board LEDs to demonstrate that
145             button is pressed
146             am_hal_gpio_output_set(AM_BSP_GPIO_LED_BLUE);
147             am_hal_gpio_output_set(AM_BSP_GPIO_LED_GREEN);
148             am_hal_gpio_output_set(AM_BSP_GPIO_LED_YELLOW);
149
150             // button is pressed, write 0x01 to turn on the LED at
151             peripheral device
152             ledCharacteristic.writeValue((byte)0x01);
153         }
154     }
155     am_hal_gpio_output_clear(AM_BSP_GPIO_LED_RED);
156     Serial.println("Peripheral disconnected");
157     return;
158 }
159
160

```

Program 1. Central LED control

```

2      /*
3         LED Control Peripheral
4
5         This example creates a BLE peripheral with service that contains a
6         characteristic to control an LED.
7
8         The circuit:
9         - Arduino Nano 33 BLE, or Arduino Nano 33 BLE Sense board.
10
11      */
12      #include <ArduinoBLE.h>
13
14      BLEService ledService("19B10000-E8F2-537E-4F6C-D104768A1214"); // BLE
15      LED Service
16
17      // BLE LED Switch Characteristic - custom 128-bit UUID, read and
18      // writable by central
19      BLEByteCharacteristic switchCharacteristic("19B10001-E8F2-537E-4F6C-
20      D104768A1214", BLERead | BLEWrite);
21
22      const int ledPin = LED_BUILTIN; // pin to config the LED
23
24      void setup() {
25          Serial.begin(9600);
26          BLE.begin();
27          //while (!Serial);
28
29          // set LED pin to output mode
30          pinMode(ledPin, OUTPUT);
31
32          // begin initialization
33          if (!BLE.begin()) {
34              Serial.println("starting BLE failed!");
35          }
36
37          // set advertised local name and service UUID:
38          BLE.setLocalName("LED");
39          BLE.setAdvertisedService(ledService);
40
41          // add the characteristic to the service
42          ledService.addCharacteristic(switchCharacteristic);
43          // add service
44          BLE.addService(ledService);
45
46          // set the initial value for the characteristic:
47          switchCharacteristic.writeValue(0);
48
49          // start advertising
50          BLE.advertise();
51      }
52
53      void loop() {

```

```

54 // listen for BLE peripherals to connect:
    BLEDevice central = BLE.central();
56
    // if a central is connected to peripheral:
58 if (central) {
    Serial.print("Connected to central: ");
60 // print the central's MAC address:
    Serial.println(central.address());
62
    // while the central is still connected to peripheral:
64 while (central.connected()) {
    // if the remote device wrote to the characteristic,
    // use the value to control the LED
66 if (switchCharacteristic.written()) {
68     if (switchCharacteristic.value()) { // value 0x01
        Serial.println("LED on");
70         digitalWrite(ledPin, HIGH); // will turn the LED on
        } else { // received 0x00 value
72             Serial.println("LED off");
            digitalWrite(ledPin, LOW); // will turn the LED off
74 or keep it off
        }
76     }
    }
78 // when the central disconnects, print it out:
    Serial.print("Disconnected from central: ");
80 Serial.println(central.address());
82
    Serial.println("Started advertise again");
    BLE.advertise();
84 }
}

```

Program 2. *Peripheral LED control*

```

2      /*
3         Transfer accelerometer sensor data wirelessly with BLE to central
4         device
5
6         This example creates a BLE peripheral with the accelerometer
7         service.
8
9         The circuit:
10        - Arduino Nano 33 BLE Sense board.
11
12        */
13
14        #include <ArduinoBLE.h>
15        #include <Arduino_LSM9DS1.h>
16
17        // BLE Accelerometer Service
18        //BLEService accelService("00000000-0000-1000-7450-
19        BE2E44B06B00");
20        BLEService accelService("ae45cfc5-152f-487e-afb5-3F30072a0f90");
21
22        //BLEFloatCharacteristic accelCharacteristic("19B10001-E8F2-537E-
23        4F6C-D104768A1214", BLERead | BLEWrite);
24        BLEFloatCharacteristic accelCharacteristic("19B10E01-E8F2-537E-
25        4F6C-8104768A1214", BLERead | BLEWrite);
26
27        void setup() {
28            Serial.begin(9600); // initialize serial communication
29            pinMode(LED_BUILTIN, OUTPUT); // initialize the built-in LED pin
30            to indicate when a central is connected
31
32            // begin initialization
33            if (!BLE.begin()) {
34                Serial.println("starting BLE failed!");
35
36                while (1);
37            }
38            if (!IMU.begin()) {
39                Serial.println("Failed to initialize IMU!");
40                while (1);
41            }
42
43            /* Set a local name for the BLE device
44               This name will appear in advertising packets
45               and can be used by remote devices to identify this BLE device
46               The name can be changed but maybe be truncated based on space
47               left in advertisement packet
48            */
49            BLE.setLocalName("Accelerometer");
50            BLE.setAdvertisedService(accelService); // add the service UUID
51
52            accelService.addCharacteristic(accelCharacteristic);
53            BLE.addService(accelService);
54
55            // set initial value for this characteristic
56            accelCharacteristic.writeValue(0);
57
58            /* Start advertising BLE. It will start continuously
59               transmitting BLE

```

```

        advertising packets and will be visible to remote BLE central
60 devices
        until it receives a new connection */
62
        // start advertising
64     BLE.advertise();

66     Serial.println("Bluetooth device active, waiting for
connections...");
68
        Serial.print("Accelerometer sample rate = ");
70     Serial.print(IMU.accelerationSampleRate());
        Serial.println(" Hz");
72     Serial.println();
        Serial.println("Acceleration in G's");
74     Serial.println("X\tY\tZ");
    }
76
    void loop() {
78
        // wait for a BLE central
80     BLEDevice central = BLE.central();

82     // if a central is connected to the peripheral:
    if (central) {
84         Serial.print("Connected to central: ");
            // print the central's BT address:
86         Serial.println(central.address());
            // turn on the LED to indicate the connection:
88         digitalWrite(LED_BUILTIN, HIGH);

90         byte change;
            float x, y, z;
92         while (central.connected()) {
            if (IMU.accelerationAvailable()) {
94                 IMU.readAcceleration(x, y, z);

96                 if (accelCharacteristic.written()) {
                    accelCharacteristic.readValue(change);
98
                    if (change == 0x01) {
100                     accelCharacteristic.writeValue(x);
                    } else if (change == 0x02) {
102                     accelCharacteristic.writeValue(y);
                    } else if (change == 0x03) {
104                     accelCharacteristic.writeValue(z);
                    }
106                 }
            }

108             //Printing for debugging purposes should be later removed
            Serial.print(x);
110             Serial.print('\t');
            Serial.print(y);
112             Serial.print('\t');
            Serial.println(z);
114         }
    }
116

        // when the central disconnects, turn off the LED:

```



```
118     digitalWrite(LED_BUILTIN, LOW);  
119     Serial.print("Disconnected from central: ");  
120     Serial.println(central.address());  
121   }  
122 }
```

Program 3. IMU

```

2      /*
3         Accelerometer data receiver
4
5         This example scans for BLE peripherals until one with the
6         advertised service
7         "ae45cfc5-152f-487e-afb5-3f30072a0f90" UUID is found.
8
9         The circuit:
10        Arduino Nano 33 BLE, or Arduino Nano 33 BLE Sense board.
11    */
12    #include <ArduinoBLE.h>
13
14    String serial;
15    int number = 0;
16
17    void setup() {
18        Serial.begin(9600);
19
20        // initialize the BLE hardware
21        BLE.begin();
22
23        Serial.println("BLE Central - LED control");
24
25        // start scanning for peripherals
26        BLE.scanForUuid("ae45cfc5-152f-487e-afb5-3f30072a0f90");
27        //BLE.scanForUuid("00000000-0000-1000-7450-BE2E44B06B00");
28    }
29
30    void loop() {
31        // check if a peripheral has been discovered
32        BLEDevice peripheral = BLE.available();
33        if (peripheral) {
34            // discovered a peripheral, print out address, local name,
35            // and advertised service
36            Serial.print("Found ");
37            Serial.print(peripheral.address());
38            Serial.print(" ");
39            Serial.print(peripheral.localName());
40            Serial.print(" ");
41            Serial.print(peripheral.advertisedServiceUuid());
42            Serial.println();
43
44            if (peripheral.localName() != "Accelerometer") {
45                Serial.println("No peripheral named Accelerometer");
46                return;
47            }
48
49            // stop scanning
50            BLE.stopScan();
51
52            controlled(peripheral);
53
54            // peripheral disconnected, start scanning again
55            //BLE.scanForUuid("19b10000-e8f2-537e-4f6c-d104768a1214");
56            //BLE.scanForUuid("00000000-0000-1000-7450-BE2E44B06B00");
57            BLE.scanForUuid("ae45cfc5-152f-487e-afb5-3f30072a0f90");
58        }

```



```
118         Serial.print("Z value: ");
119         Serial.println(z);
120     } else if (serval == "all") {
121         switchChar.writeValue(byte(0x01));
122         switchChar.readValue(&x, 4);
123         Serial.print("X value: ");
124         Serial.print(x);
125         switchChar.writeValue(byte(0x02));
126         switchChar.readValue(&y, 4);
127         Serial.print(" Y value: ");
128         Serial.print(y);
129         switchChar.writeValue(byte(0x03));
130         switchChar.readValue(&z, 4);
131         Serial.print(" Z value: ");
132         Serial.println(z);
133     }
134 }
135 Serial.println("Choose x, y, z or all: ");
136 }
137 }
138 }
139 Serial.println("Peripheral disconnected");
140 }
```

Program 4. *Accelerometer receiver*

REFERENCES

- [1] ENOMA Energy Harvesting homepage, <https://www.energyharvesting.fi/>.
- [2] H.S. Dhadwal, J. Rastegar, Energy harvesting : for low-power autonomous devices and systems, SPIE Press, Bellingham, Washington, 2017.
- [3] A.K. Sultania, C. Delgado, J. Famaey, Enabling low-latency bluetooth low energy on energy harvesting batteryless devices using wake-up radios, Sensors (Basel, Switzerland), Vol. 20, Iss. 18, 2020, pp. 1-19.
- [4] M. Chiu, Y. Chang, L. Yeh, C. Chung, Numerical Assessment of a One-Mass Spring-Based Electromagnetic Energy Harvester on a Vibrating Object, Archives of acoustics, Vol. 41, Iss. 1, 2016, pp. 119-131.
- [5] T. Kivimäki, J. Vanhala, A. Halme, J. Ruuskanen, T. Salminen, Distributing the Generation of Electricity to Extreme Level, Proceedings of the 4th International Conference on Power and Energy Applications.
- [6] S. Beeby, N. White, Energy Harvesting For Autonomous Systems, Artech House, Norwood, 2010.
- [7] A.C. Hewson, V. Zlatić, Properties and applications of thermoelectric materials : the search for new materials for thermoelectric devices, 2009.
- [8] A. Erturk, D.J. Inman, Piezoelectric energy harvesting, Wiley, Chichester, 2011.
- [9] K. Townsend, B. Sawyer, M.K. Loukides, K. Montgomery, D. Futato, R. Demarest, Getting started with bluetooth low energy, O'Reilly, Sebastopol, California, 2014.
- [10] G. Backman, B. Lawton, N. A. Morley, Magnetostrictive Energy Harvesting: Materials and Design Study, in: IEEE Transactions on Magnetics, 2019, pp. 1-6.
- [11] T. Ueno, Magnetostrictive vibrational power generator for battery-free IoT application, AIP advances, Vol. 9, Iss. 3, 2019, pp. 35018-5.
- [12] R. La Rosa, P. Livreri, C. Dehollain, M. Costanza, C. Trigona, An energy autonomous and battery-free measurement system for ambient light power with time domain readout, Measurement : journal of the International Measurement Confederation, Vol. 186, 2021, pp. 110158.
- [13] F. J. Dian, A. Yousefi, S. Lim, A practical study on Bluetooth Low Energy (BLE) throughput, 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), pp. 768-771.
- [14] M. Afaneh, Intro to Bluetooth Low Energy, Novel Bits, 2018.
- [15] J. Lee, M. Dong, Y. Sun, A preliminary study of low power wireless technologies: ZigBee and Bluetooth Low Energy, in: ICIEA, IEEE, 2015, pp. 135-139.

- [16] L. Wood, Bluetooth Low Energy IoT Market Report 2017 - BLE's Biggest Competition Comes from 802.15.4 and the Thread/Zigbee Ecosystem, NASDAQ OMX's News Release Distribution Channel, 2017.
- [17] M. B. Yassein, W. Mardini, A. Khalil, Smart homes automation using Z-wave protocol, 2016 International Conference on Engineering & MIS (ICEMIS), pp. 1-6.
- [18] RS Components, 11 Internet of Things (IoT) Protocols You Need to Know About, 2015, <https://www.rs-online.com/designspark/eleven-internet-of-things-iot-protocols-you-need-to-know-about>.
- [19] T. John, Comparison of Wireless Technologies: Bluetooth, WiFi, BLE, Zigbee, Z-Wave, 6LoWPAN, NFC, WiFi Direct, GSM, LTE, LoRa, NB-IoT, and LTE-M, <https://predictabledesigns.com/wireless-technologies-bluetooth-wifi-zigbee-gsm-lte-lora-nb-iot-lte-m/>.
- [20] Z. Yang, C.H. Chang, 6LoWPAN Overview and Implementations. pp. 357-361, https://scholar.google.com/scholar?hl=fi&as_sdt=0%2C5&q=6LoWPAN+Overview+and+Implementations.&btnG=.
- [21] S. Kaushik, An overview of technical aspect for WiFi networks technology, International Journal of Electronics and Computer Science Engineering (IJECSE), ISSN: 2277-1956), Vol. 1, Iss. 01, 2012, pp. 28-34.
- [22] K. Mekki, E. Bajic, F. Chaxel, F. Meyer, Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT, 2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), pp. 197-202.
- [23] Silicon Labs Admin, BLE master/slave, GATT client/server, and data RX/TX basics, 2021, https://community.silabs.com/s/article/ble-master-slave-gatt-client-server-and-data-rx-tx-basics?language=en_US.
- [24] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, D. Formica, Performance Evaluation of Bluetooth Low Energy: A Systematic Review, Sensors (Basel, Switzerland); Sensors (Basel), Vol. 17, Iss. 12, 2017, pp. 2898. Available (accessed ID: cdi_doaj_primary_oai_doaj_org_article_3a32a1146d544f81bdf7dad78ecac95f): .
- [25] S. Kamath, J. Lindh, Measuring bluetooth low energy power consumption, Texas instruments application note AN092, Dallas, 2010, .
- [26] Arm Mbed ARM Mbed Studio Documentation, <https://os.mbed.com/docs/mbed-studio/current/introduction/index.html>.
- [27] E.C. NATE Artemis Development with Arduino, <https://learn.sparkfun.com/tutorials/artemis-development-with-arduino>.
- [28] Low Noise, Wide Bandwidth, MEMS Accelerometer Datasheet, <https://www.analog.com/en/products/adxl1003.html#product-evaluationkit>.
- [29] Ampiq Micro Apollo3 Blue MCU Datasheet , https://cdn.sparkfun.com/assets/1/5/c/6/7/Apollo3-Blue-MCU-Datasheet_v0_15_0.pdf.

[30] I. Galili, D. Kaplan, Y. Lehavi, Teaching Faraday's law of electromagnetic induction in an introductory physics course, *American journal of physics*, Vol. 74, Iss. 4, 2006, pp. 337-343.

[31] J. Seo, K. Jhang, H. Lee, Y. Kim, Vibration energy harvesting technology for smart tire monitoring, *Journal of mechanical science and technology*, Vol. 33, Iss. 8, 2019, pp. 3725-3732.

[32] LIQUID SHOULDER Using SparkFun Edge Board with Ambiq Apollo3 SDK, <https://learn.sparkfun.com/tutorials/using-sparkfun-edge-board-with-ambiq-apollo3-sdk/bash-make-and-python----oh-my>.

[33] SparkFun Artemis Development Kit product information, <https://www.sparkfun.com/products/16828>.

[34] ARDUINO NANO 33 BLE SENSE product information, <https://store.arduino.cc/arduino-nano-33-ble-sense>.

[35] ARDUINO NANO product information, <https://store.arduino.cc/arduino-nano>.

[36] M. Afaneh How Bluetooth Low Energy Works, <https://www.novelbits.io/bluetooth-low-energy-advertisements-part-1/>.

[37] OpenStax College Angular Acceleration, web page, <https://courses.lumenlearning.com/physics/chapter/10-1-angular-acceleration/>.

[38] Catur Edi Widodo, Kusworo Adi, Monitoring for vehicle velocity and acceleration using an accelerometer, *International Journal of Innovative Research in Advanced Engineering*, IV, 20-23. doi: 10.26562/IJIRAE.2017.NVAE10086, 2017, <https://www.ijirae.com/volumes/Vol4/iss11/04.NVAE10086.pdf>.

[39] P. Bulić, G. Kojek, A. Biasizzo, Data Transmission Efficiency in Bluetooth Low Energy Versions, *Sensors (Basel, Switzerland); Sensors (Basel)*, Vol. 19, Iss. 17, 2019, pp. 3746.

[40] L. Wu, Experimental study on vehicle speed estimation using accelerometer and wheel speed measurements, 2011 Second International Conference on Mechanic Automation and Control Engineering, pp. 294-297.

[41] M. Murase, K. Tanaka, K. Naito, Prototype implementation of human management system with BLE beacon devices in natural disasters, 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC), pp. 1-2.

[42] J. Cerón D., D. López M., B.M. Eskofier, Human Activity Recognition Using Binary Sensors, BLE Beacons, an Intelligent Floor and Acceleration Data: A Machine Learning Approach, *Proceedings*, Vol. 2, Iss. 19, 2018, pp. 1265.

[43] J. Zhao, Z. You, A shoe-embedded piezoelectric energy harvester for wearable sensors, *Sensors (Basel, Switzerland); Sensors (Basel)*, Vol. 14, Iss. 7, 2014, pp. 12497-12510.

[44] H. Katsumura, T. Konishi, H. Okumura, T. Fukui, M. Katsu, T. Terada, T. Umegaki, I. Kanno, Development of piezoelectric vibration energy harvesters for battery-less

smart shoes, Journal of physics.Conference series; J.Phys.: Conf.Ser, Vol. 1052, Iss. 1, 2018, pp. 12060.