Eemeli Saarinen

Data Collection Tool for Movesense Sensors

Metropolia University of Applied Sciences

Bachelor of Engineering

Information and Communications Technology

Bachelor's Thesis

9 February 2024

Abstract

Author:	Eemeli Saarinen
Title:	Data Collection Tool for Movesense Sensors
Number of Pages:	36 pages + 2 appendices
Date:	9 February 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communications Technology
Professional Major:	Smart IoT Systems
Supervisors:	Sakari Lukkarinen, Project Manager
	Saana Vallius, Senior Lecturer

The aim of this project was to create a data collection tool for Movesense sensors capable of connecting to a sensor and subscribing to a data stream from a laptop. The tool was to be suitable for tracking body and limb movement and orientation. The project was done for Metropolia University of Applied Sciences. The tool was intended to be used on a project course where second-year Health Technology students learn to process, analyze, and visualize data using Python. As such, the tool was to be done using Python as it was going to be used with Jupyter Notebooks. Another goal was to investigate ways to simultaneously connect multiple sensors to the laptop and synchronize them.

A custom application was developed for the Movesense sensor to enable communication between the sensor and the laptop using Bluetooth Low Energy GATT services. An asynchronous Movesense Python library suited for collecting sensor data was developed for the project. The library can be used to find a nearby Movesense sensor, connect to it, and subscribe to sensor data streams. Connecting to multiple sensors simultaneously is also possible.

Validation testing was conducted to test the functionality and usability of the tool. The testing was conducted by eight test groups. Each group consisted of three second-year students specializing in Health Technology. The test comprised of four test cases. The validation test concluded that the tool was simple and intuitive to use. The testing provided valuable information about the usability of the tool. Furthermore, it brought up previously unnoticed errors that were then fixed.

The main goals of this thesis were accomplished successfully. The data collection tool contains all the required functionalities. Connecting to multiple sensors was also accomplished. However, the synchronization of the sensors requires further development.

Keywords: Movesense, BLE, GATT, sensor, Python, Jupyter

Tiivistelmä

Tekijä:	Eemeli Saarinen
Otsikko:	Tiedonkeruutyökalu Movesense-antureille
Sivumäärä:	36 sivua + 2 liitettä
Aika:	9.2.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Älykkäät IoT-järjestelmät
Ohjaajat:	Projektipäällikkö Sakari Lukkarinen Lehtori Saana Vallius

Tämän insinöörityön tavoite oli luoda tiedonkeruutyökalu Movesense-antureille, jotka keräävät mittaustietoja kannettavalle tietokoneelle. Työkalun tuli soveltua kehon ja raajojen liikkeen ja suunnan mittaamiseen. Projekti tehtiin Metropolia Ammattikorkeakoululle. Työkalu oli tarkoitettu käytettäväksi toisen vuoden projektikurssilla, jossa opitaan käsittelemään, analysoimaan ja visualisoimaan mittaustietoja käyttäen Python-ohjelmointikieltä. Tämän takia työkalu tuli tehdä Pythonilla, jotta sitä voidaan käyttää Jupyter Notebookissa. Toinen tavoite projektille oli tutkia useampaan Movesense-anturiin yhdistämistä samanaikaisesti ja sitä, kuinka niistä kerätty data voidaan synkronoida.

Movesense-antureille luotiin mukautettu sovellus, jolla saatiin anturi keskustelemaan kannettavan tietokoneen kanssa käyttäen Bluetooth Low Energy GATT -palveluja. Projektissa kehitettiin myös Movesense-anturille asynkroninen Python-kirjasto, joka soveltuu tiedonkeruuseen. Kyseinen kirjasto pystyi löytämään lähellä olevan anturin osoitteen, yhdistämään siihen ja keräämään mittaustietoja liikeantureista. Kirjasto kykeni myös keräämään mittaustietoja useasta Movesense-anturista samanaikaisesti.

Työkalun toimivuutta ja käytettävyyttä testattiin validointitestillä. Testauksen suoritti kahdeksan kolmen hengen testiryhmää. Ryhmät koostuivat toisen vuoden Terveysteknologian erikoisalan opiskelijasta. Testi koostui neljästä testitapauksesta. Validointitestissä todettiin, että työkalu oli yksinkertainen ja intuitiivinen käyttää. Testaus tuotti hyödyllistä tietoa työkalun käytettävyydestä. Lisäksi työkalusta löytyi aiemmin huomaamatta jääneitä virheitä, jotka saatiin korjattua.

Insinöörityön päätavoitteet saavutettiin onnistuneesti. Tiedonkeruutyökalu sisältää halutut toiminnot. Usean Movesense-anturin käyttö samanaikaisesti onnistuttiin toteuttamaan, mutta niiden synkronointi vaatii vielä kehitystä.

Avainsanat: Movesense, BLE, GATT, anturi, Python, Jupyter

Contents

List of Abbreviations

AFE:	Analog Front-End
API:	Application Programming Interface
ATT:	Attribute Protocol
BLE:	Bluetooth Low Energy or Bluetooth Smart
CPU:	Central Processing Unit
CSV:	Comma-Separated Values
DFU:	Device Firmware Update
ECG:	Electrocardiogram
FIFO:	First-In First-Out
GAP:	Generic Access Profile
GATT:	Generic Attributes Profile
HCI:	Host Controller Interface
HTTP:	Hypertext Transfer Protocol
IMU:	Inertial Measurement Unit
ISM:	Industrial, Scientific, and Medical
LED:	Light-Emitting Diode
MCU:	Microcontroller Unit

OS:	Operating System
OTA:	Over-The-Air
RAM:	Random Access Memory
REST:	Representational State Transfer
SIG:	Special Interest Group
UART:	Universal Asynchronous Receiver/Transmitter
UUID:	Universally Unique Identifier
YAML:	YAML Ain't Markup Language

1 Introduction

The goal of this final year project was to create a tool that can connect to a Movesense sensor and collect data from the Movesense sensor over Bluetooth Low Energy (BLE). The project was carried out for the School of Information and Communication Technology at Metropolia University of Applied Sciences. The tool was intended to be used on a project course where second-year Health Technology students learn to process, analyze, and visualize data using the Python programming language. During the project, Movesense sensors were used to track body movement.

The Movesense sensor is a small wearable BLE device. It can be used for measuring motion, heart rate and temperature. Movesense provides open-source software libraries for developing mobile and device applications. Movesense has a Showcase App mobile application for Android and iOS devices which can be used to collect data from the Movesense sensor. The device library contains sample applications for the Movesense sensor. The applications showcase different features of the sensor software. These sample applications can be used as a starting point for developing custom applications.

Previously the measurement data was collected using a mobile application. The data was then moved to the cloud to be used on the laptop. This method was inconvenient as a mobile application added an unnecessary step between measuring data and using it. There were also issues with the application. It saved the data using wrong decimal formatting and used different headers in CSV files for the same measurement data.

As a result, the aim of this project was to build a tested, documented, and tailored Movesense data collection tool. The tool was to be suitable for tracking the movement and orientation of the body and limbs.

Another goal was to remove the phone from the data collection process. Instead of connecting to the phone, the Movesense sensor was connected directly to a

laptop using BLE. The data was collected directly on the laptop and saved to a cloud service or to a local directory. There was need to transfer the data to the laptop every time new measurements are needed.

The tool was created using Python as the students will be using it with a Jupyter Notebook. This allowed the students to process, analyze and visualize the sensor data right after collecting it from the sensor.

Connecting to multiple Movesense sensors simultaneously and whether they can be synchronized was also investigated. The power consumption and how often data is then sent from the sensor should be studied as well.

2 Project Specifications

To accomplish the objectives for the project, the data collection tool was to be able to find the Movesense sensor. Once the sensor has been found, the tool must establish a connection to the sensor. Finally, the tool had to subscribe to the data stream of the Movesense sensor. The tool was to work inside a Jupyter Notebook. The data transfer was to be done using the BLE protocol.

The project was developed using Python programming language. The project used Python Bluetooth library called Bleak (Bluetooth Low Energy platform Agnostic Klient) to establish connection between a Movesense sensor and a laptop. Bleak acted as a Bluetooth Low Energy (BLE) Generic Attributes Profile (GATT) client for connecting to GATT services on BLE devices [1].

By default, the standard Movesense application does not have a way to communicate without a mobile application. Movesense has alternative communication methods which was used to get around this issue. To meet the requirements of this project, Movesense had to use an application capable of communicating with the Bleak GATT client.

The intended users of the data collection tool were teachers and students who are familiar with the Python language. On the course, the students use Jupyter Notebooks to measure, analyse, and visualize data. The Python library for the data collection was designed be intuitive and simple to use. Furthermore, the project was to include sample Notebooks to showcase how the tool was used.

In addition to creating the Python library, the objective was to explore connecting to multiple Movesense sensors simultaneously and to synchronize the incoming data streams. Another objective was to investigate the power consumption of the Movesense sensor and how often the data is sent from the sensor.

3 Background

This section contains background information about the Movesense sensor and the Bluetooth Low Energy protocol. Section 4.1 discusses the Movesense sensor and its hardware, the Whiteboard framework, power consumption, communication methods, and how the device firmware is updated. Section 4.2 goes over the BLE, and the protocols used in this project.

3.1 Movesense

The Movesense sensor is a small and versatile BLE device containing multiple sensors. It is designed to be used for health, sports, and wellbeing applications. The case of the device is constructed to be shock-proof and water-resistant up to 30 meters, making it suitable for all types of sports. Figure 1 shows the Movesense sensor from both sides. The device has a low profile and can be attached to gear or accessories using the two metal studs on the back side of the device. The studs are also used for heart rate and ECG measurements. The Movesense sensor weighs only 9.4 grams with the battery, making it very lightweight. [2.]

Figure 1. Front and back side of Movesense sensor [3].

There are currently three Movesense sensor models: Active, Medical and Flash. Movesense Active is designed to be used for sports and other activities. It currently has two variants, HR2 and HR+ [4]. HR+ has all the features of HR2 with the addition of a temperature sensor and compatibility with the Movesense Smart Connector. The Smart Connector can be used for context identification. [5.] There are also some older variants that are no longer sold. Movesense Medical focuses on healthcare applications. It is equipped with class IIa medically certified ECG and a motion sensor [6]. Lastly, the Movesense Flash is made for long-term data collection without the need to be connected to a client. It has 128MB on-board memory compared to the 3Mbit EEPROM memory of the other models. [7.]

The Movesense sensor has open-source developer resources and an Application Programming Interface (API) for developers and manufacturers to build custom in-device and mobile applications for their needs. There are two main repositories: device library and mobile library. The device library contains sample applications, tools, the core library, and other recourses for developers. The mobile library contains the Showcase Apps and mobile libraries for Android and iOS platforms.

3.1.1 Sensor Hardware

The Movesense device was designed using Suunto design and development guidelines and they are designed and manufactured in Finland. The devices are designed to be durable and robust to be able to handle all kinds of activities. [2.]

The Movesense sensor is powered by a CR2025 lithium coin cell battery. It can provide up to months of operating time depending on the application. At the core of the device there is an nRF52832 microcontroller unit (MCU) by Nordic Semiconductors. The nRF52832 has a 32-bit ARM Cortex-M4 core. The core handles both the Movesense platform and the Bluetooth connectivity for the device. The MCU has 65kB of on-chip Random Access Memory (RAM) and 512kB of on-chip Flash memory that are shared with the application and the

Movesense operating system (OS). Both Bluetooth 4.0 and 5.0 are supported by the nRF52832 chip. It also has 3Mbit of non-volatile external memory (EEPROM). [2.]

As Movesense strives for low power consumption and long battery life, the selected components for the device have ultra-low power consumption [2].

Movesense has a 9-axis Inertial Measurement Unit (IMU) comprised of a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer [2]. There is a hardware limitation that makes it impossible to use the gyroscope or the magnetometer without powering on the accelerometer as it is responsible for the IMU data transfers [8]. The IMU can be used for tracking the 3D orientation and movement of a body.

The 3-axis accelerometer measures linear acceleration and forces in three dimensions. The Movesense uses m/s^2 as the unit for accelerometer data. The accelerometer supports sampling frequencies from 13Hz to 1666Hz, each rate roughly doubling [9]. It is recommended to use the lowest possible frequency needed for the use case since the central processing unit (CPU) load and power consumption increase with higher frequencies [8]. The accelerometer can be used as a wake-up method for the sensor. The sensor can be configured to wake up from any movement, a single tap, a double tap, or a free fall. The force needed for waking up can also be configured [10]. The accelerometer has four configurable ranges. The ranges are listed in Table 1.

The 3-axis gyroscope measures angular velocity in three dimensions. It uses degrees per second (dps) as the measurement unit. As the accelerometer and gyroscope are part of the same component, they share the same sampling frequencies [2]. Also, the measurement range for the gyroscope can be configured as well and the range options can be found in the Table 1.

The 3-axis magnetometer measures the position of the sensor relative to the magnetic field of Earth in three dimensions. It shares the sampling frequencies with the accelerometer and the gyroscope [2].

The measurement ranges for the accelerometer, the gyroscope and the magnetometer are listed in Table 1 [2].

Table 1. Supported measurement ranges for sensors [2].

Sensor	**Range**
Accelerometer	±2/±4/±8/±16g full scale
Gyroscope	±125/±245/±500/±1000/±2000°/s full scale
Magnetometer	±49 gauss full scale
Temperature	accuracy <±0.5°C, 0°C to +65°C

Capturing ECG signals and calculating heart rate with the Movesense device is possible using the analog front-end (AFE). The AFE is used through the studs at the back of the sensor. The Movesense sensors use single-channel ECG. The HR models measure non-medical ECG and the Medical model measures medical ECG. The difference is due to the medical approvals applied for the devices. The supported sampling frequencies for the ECG are 125Hz, 128Hz, 200Hz, 250Hz, 500Hz and 512Hz. When sampling the ECG data over BLE, sampling rates above 256Hz should be avoided. The reason for this is the First-In, First-out (FIFO) buffer of the analog front-end that holds up to 32 samples. High frequencies over BLE can cause the FIFO to overflow leading to data loss. High-pass and low-pass filters can be configured for the ECG. Heart rate can be measured using the API or using the BLE standard heart rate service (HRS). It is measured as beats per minute, and it includes the RR intervals for the measurements. [9.]

Some Movesense models also include a temperature sensor for measuring internal temperatures of the device. The accuracy of the temperature measurement is +-0.5 °C and the device can measure temperatures between 0 °C and +65 °C. [9.]

3.1.2 Whiteboard and API

Movesense uses their own REST-like communication framework called Whiteboard [11]. It is an internal framework created by Suunto for using a REST-like interface with embedded devices [12]. Whiteboard is a fully asynchronous library that handles both internal and external communication for the Movesense sensor. GATT communication is an exception and is not handled by Whiteboard. Like REST, Whiteboard has a client-service architecture. Clients use the provided services by sending asynchronous requests. Services send asynchronous responses and provide the requested resources to the clients. [13] Clients can use the Whiteboard API which includes the commonly used GET, PUT, POST and DELETE request types. Whiteboard includes an additional subscription feature which allows clients to subscribe to continuous data from the sensor. Data is normally delivered with continuous notifications. Subscriptions can be terminated using an unsubscribe request. [11.]

The Movesense REST API is divided into eight sections listed in Table 2. Each section is composed of multiple modules. Modules are defined in YAML files and are defined using the Swagger 2.0 specification. It is used by most Internet REST services. It is possible to define custom APIs for applications. [14.]

Table 2. The Movesense REST API sections [14].

Resource	Description
/Meas	Sensor information, configuration, and data
/Mem	Data memory access for DataLogger and Logbook
/Comm	Communication protocols: BLE and 1Wire
/Component	Low-level component features: LED, EEPROM, chip specific features
/System	System features: Mode, Settings, Energy, Memory, States
/UI	User interface
/Misc	Everything that does not fit elsewhere
/Whiteboard	Whiteboard services

The first line of the YAML file starts with defining the Swagger version. The structure of the API is defined by the version used. The Movesense API uses four sections from the Swagger 2.0 specification, info, paths, parameters, and definitions. The information section contains basic information about the API. Listing 1 shows the info section from the IMU API.

```
swagger: '2.0'

info:
  version: NA
  title: IMU (Inertial Motion Unit) - Movesense-API
  description: |
    This file defines interface for the IMU API's.
  x-api-type: public
  x-api-required: true
```

Listing 1. Information section from the IMU API file [15].

API endpoints are defined in the paths section. Each path specifies the supported operations of the endpoint such as GET or POST. The paths can contain parameters as is the case for subscriptions. The subscription path has a SampleRate parameter which tells the sensor how often the data should be sampled and sent to the client. Paths section from the IMU API can be seen in Listing 2.

```
paths:
…
  /Meas/IMU9/{SampleRate}:
    parameters:
       - $ref: '#/parameters/SampleRate'

  /Meas/IMU9/{SampleRate}/Subscription:
    parameters:
       - $ref: '#/parameters/SampleRate'
    post:
      description: |
        Subscribe to periodic 9-axis IMU measurements.
      responses:
        200:
          description: Operation completed successfully
        501:
          description: Non-supported sample rate
        x-notification:
          description: New measurements
          schema:
            $ref: '#/definitions/IMU9Data'
    delete:
      description: |
        Unsubscribe from periodic 9-axis IMU values.
      responses:
        200:
          description: Operation completed successfully
```

Listing 2. Path from the IMU9 Subscription endpoint [15].

The Parameters section defines parameters used by paths. A parameter defines type and format of the parameter. Parameters are mostly used in sensor APIs. Listing 3 shows the SampleRate parameter from the IMU API.

```
parameters:
  SampleRate:
    name: SampleRate
    in: path
    required: true
    type: integer
    format: int32
```

Listing 3. Sample rate parameter used by the IMU subscription endpoints [15].

The Definitions section contains definitions for data and resources used by endpoints. Definitions contain information such as data types, formats, and units of the data. Listing 4 shows an example definition from the IMU API.

```
definitions:
…
  IMU9Data:
    required:
      - Timestamp
      - ArrayAcc
      - ArrayGyro
      - ArrayMagn
    properties:
      Timestamp:
        description: Local timestamp of first measurement.
        type: integer
        format: uint32
        x-unit: millisecond
      ArrayAcc:
        description: Measured acceleration values (3D) in array.
        type: array
        x-unit: m/s^2
        items:
          $ref:
'http://localhost:9000/builtinTypes.yaml#/definitions/FloatVector3D'
      ArrayGyro:
        description: Measured angular velocity values (3D) in array.
        type: array
        x-unit: dps (degree per second)
        items:
          $ref:
'http://localhost:9000/builtinTypes.yaml#/definitions/FloatVector3D'
      ArrayMagn:
        description: Measured magnetic field values (3D) in array.
        type: array
        items:
          $ref:
'http://localhost:9000/builtinTypes.yaml#/definitions/FloatVector3D'
```

Listing 4. IMU9 data structure definition from the IMU API.[15]

Application specific APIs are declared in the app_root.yaml file. The file is in the root folder of the application. The APIs defined by the application can be found in the wbresources folder. [14.]

3.1.3 Power Consumption

The main communication method for Movesense sensor is Bluetooth Low Energy. It also has the largest power consumption out of all the peripherals of the device. For this reason, BLE advertising should be turned off when possible. [8.] Bluetooth should not be connected longer than needed. On demand

connections should be preferred instead [16]. The CPU frequency is automatically reduced by the Movesense technology in response to calculation demand. It is possible to get as low as 10uA current consumption by disconnecting Bluetooth and unsubscribing all providers. [8.]

Movesense has a Full Power Off mode which should be used when the device is not in use. There are two methods that can be configured to wake up the device from the Full Power Off mode. The first method is movement wake up that uses an accelerometer to wake up the device based on the configured movement. The second method is heart rate wakeup which uses the studs at the bottom of the device. When using the Full Power Off mode, system time will reset [16].

After BLE, the sensors have the second largest power consumption of the Movesense device. [8]. The consumption depends mostly on the sampling frequency. Movesense recommends avoiding higher frequencies than necessary. For accelerometers, the power consumption significantly increases after 52 Hz. [16.] This applies to the gyroscope and the magnetometer since the accelerometer is responsible for transferring data from all of three sensors [8].

3.1.4 Communication Methods

The main communication method for connecting to a Movesense sensor is an Android/iOS mobile application. Movesense also offers alternative methods for communicating with the device. The device has a built-in heart rate service (HRS), Nordic UART services, and a custom GATT service. Furthermore, Movesense offers an option to embed data into BLE advertisement packets. [14.]

UART is used for debugging the sensor using a Movesense programming jig [8]. UART could not be used in this project as it is not a wireless communication method.

Embedding data to BLE advertising packets can be used for broadcast-oriented applications where other devices may discover the Movesense sensor and collect data without pairing with the device. The method is very limited and does not fit the requirements of this project. This is because the advertising packets can contain up to 27 bytes of data and can be sent around 5 times per second. The minimum size of an IMU9 data packet is 40 bytes which are sent 13 times per second, which means the embedding method cannot send data fast enough. Furthermore, the goal is to have the Movesense sensor connected to the laptop using BLE. [14.]

3.1.5 Device Firmware Update

The Movesense device is updated using a Device Firmware Update (DFU). This process happens over-the-air (OTA) by sending a DFU package to a Movesense sensor. The package contains all relevant data for updating the device application. There is also an additional DFU package type for updating the device firmware from an older to a newer version (e.g. 1.9.x to 2.x) This package type contains the BLE stack and bootloader in addition to the data contained in a regular DFU package. [17.]

The DFU can be performed using the Movesense Showcase App or nRF Connect app from Semiconductor. The device goes to DFU mode when updating the device. During the DFU mode, the device is advertised by the name “DfuTarg”. If something interrupts the update process, the device will stay in the DFU mode until the device is reset. The reset can be done by removing the battery. If a broken application is installed onto a Movesense device, the device can be put into a DFU Recovery Mode. [17.]

3.2 Bluetooth Low Energy

BLE is a low-power wireless technology. It is a new technology rather than an upgrade to the Bluetooth classic. As such, it is not compatible with the Bluetooth classic. BLE was introduced with the Bluetooth 4.0 specification, and it is

intended for low-power Internet of Things applications such as sensors. It specifically targets battery powered devices that need to run for long periods without changing the battery. Like Bluetooth and Wi-Fi, BLE operates on the 2.4 GHz ISM band. [18.]

BLE achieves its low power by using low data transfer speeds to send small amounts of data. The radio is turned off whenever it is not needed. The data transfer speeds differ based on the Bluetooth version used. [18.]

The range of the BLE can reach up to 50 meters when there are no obstacles between the devices. When there are walls and obstacles the range drops significantly. Bluetooth 5.0 specification introduces a long-range feature which increases the range up to 800 meters with no obstacles. However, the transfer speeds are much lower because of it. [18.]

The BLE Stack illustrated in Figure 2 displays the layers and functional modules which are distributed between two architectural blocks known as the host and the controller. Between the two blocks there is a Host Controller Interface (HCI), a logical interface, which defines the communication between the two blocks. [19.]

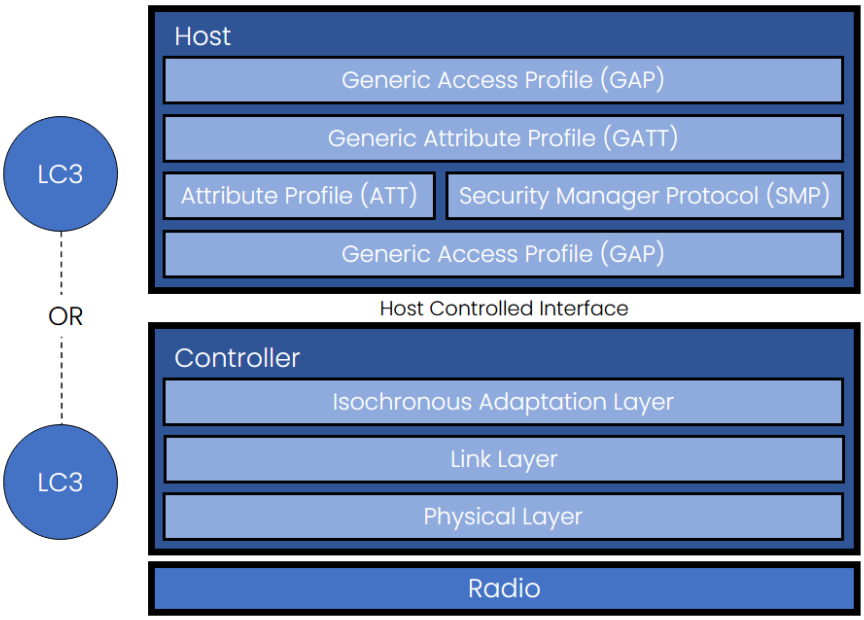

Figure 2. Layers and functional modules of BLE Stack distributed between the host and the controller architectural blocks.

The controller and the host operate as distinct logical containers. As such, they can be implemented in different physical components from different manufacturers. The host and the controller can be compared to an operating system and a system on a chip. [19.]

The HCI is a logical component which can be put into practice in multiple ways. However, the logical interface will always stay the same [19].

The device discovery and connection are defined by the Generic Access Profile (GAP). The Generic Attribute Profile and the Attribute Protocol are discussed in more detail below. Security Manager Protocol (SMP) is utilized for security procedures. [19.]

3.2.1 Attribute Protocol

GATT specifies in detail in how all user and profile data is exchanged over the BLE connection. It uses the Attribute Protocol (ATT), a generic data protocol as its transport protocol. ATT acts as the underlying infrastructure for GATT. It defines how data is exposed by servers to clients. It also defines how data is structured. ATT uses the term Attribute to refer to all data exposed by the server. [20.] Figure 3 demonstrates a GATT client requesting data from a GATT server and the server responding to the requests.

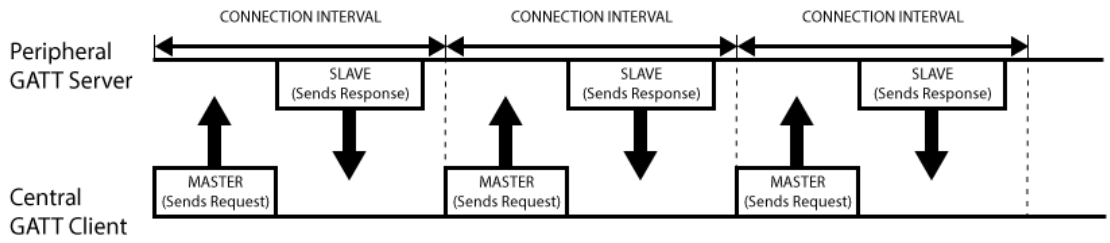

Figure 3. GATT transactions between a GATT server and a client [21].

The Attribute Structure has four elements: a handle, a Universally Unique Identifier (UUID), a value and permissions. The handle is a 16-bit unique identifier that the client uses to reference an attribute on a server. It handles a range hexadecimal values from 0x0001 to 0xFFFF. The UUID is a unique identifier which can be assumed to be globally unique. Bluetooth SIG uses 16-bit UUID numbers whereas custom UUIDs are 128 bits long. The SIG-adopted UUIDs use a common 128-bit UUID as a base. The 16-bit UUID replaces four digits from the base starting from the 5th digit. Custom UUIDs cannot conflict with SIG-adopted UUIDs. The value of the attribute holds data exposed by the server. It has a variable length, and it is formatted based on the attribute type. The attribute type is defined by the UUID. Service declaration and characteristic declaration are examples of attribute types. Permissions define read and write access and if the attribute can be notified or indicated. It also defines security levels for each of the forementioned operations. [20.]

ATT has two roles: a server and a client. A server exposes data to clients and sends responses, notifications, and indications to those clients. A client on the other hand communicates with the server by sending requests and commands and receives data from the server. ATT uses six types for packets for the communication listed below. [20.]

- Commands (no response required)
- Requests (response required)
- Responses (response to a request)
- Notifications (no response required)
- Indications (response required)
- Confirmations (response to an indication)

The Attributes have two main data operation types: reads and writes. Since every read expects a response, all reads are requests. All writes do not require a response meaning they can either be commands or requests. [20.]

3.2.2 Generic Attribute Profile

GATT describes how services and characteristics are formatted and how attributes are interacted with. These include procedures such as service discovery, characteristic reads, writes, notifications and indications. GATT shares the same roles with ATT. The roles are established per transaction rather than being predetermined for each device. For instance, when the indication is sent by the server, the client will respond with a confirmation. In this case, the server acts as the client. As such, the BLE device can act as a GATT server and a client simultaneously. [20.]

3.2.3 Services

Service is a term used for a group of one or more attributes. The purpose is to group related attributes together. Services have characteristic and non-characteristic attributes. Characteristic attributes contain values such as sensor

data. Non-characteristic attributes help structure of the data in the service. The structure of a GATT server is shown in Figure 4. [20.]

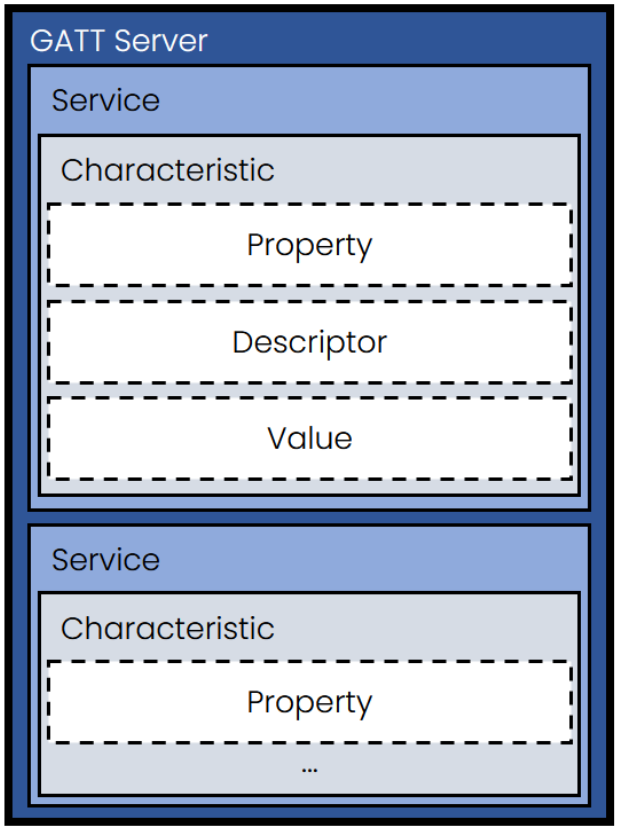

Figure 4. Service hierarchy of a GATT server and services.

The GATT server contains services with characteristics. Each service can have multiple characteristics. The characteristics are discussed in more detail in the next section.

3.2.4 Characteristics

A characteristic is a container which holds data or where data is written to. Characteristics can also include other attributes like properties or descriptors. Properties define how the value can be utilized. Descriptors contain information about the contained value like a user description, format, or a unit. [20.]

4 Implementation

Figure 5 showcases the system diagram. The diagram shows how the Movesense sensor was originally used. The Movesense sensor had to be connected to the phone where the Movesense Showcase App was used to record the sensor data. The sensor data had to be manually transferred to the cloud or directly to the laptop. The Movesense Showcase App saves the data into CSV files. However, it does not use unified headers for the same sensor measurements. The CSV files also use a wrong decimal point localization.

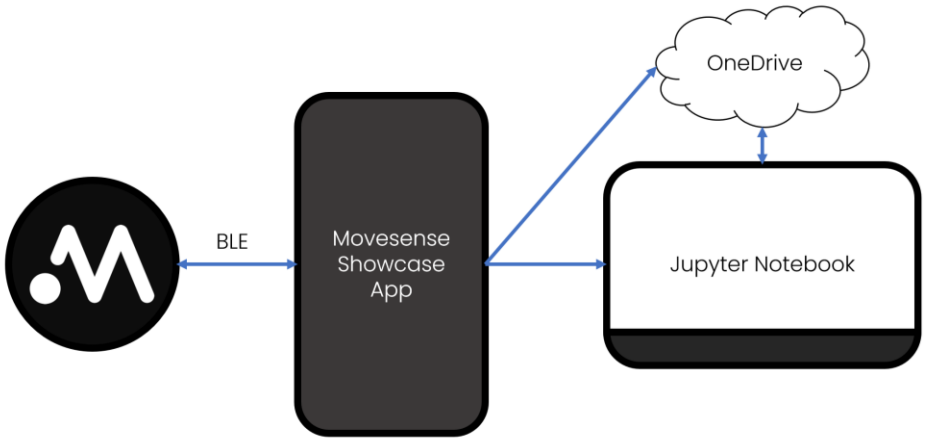

Figure 5. System diagram of the original way of collecting data from a Movesense sensor.

Figure 6 shows the system diagram of the intended solution. In this diagram, the mobile phone is no longer part of the measuring process. The Movesense sensor communicates with the laptop directly using GATT. The measured data can be automatically saved to the cloud or to the laptop. The solution fixes the problems mentioned in the previous paragraph.

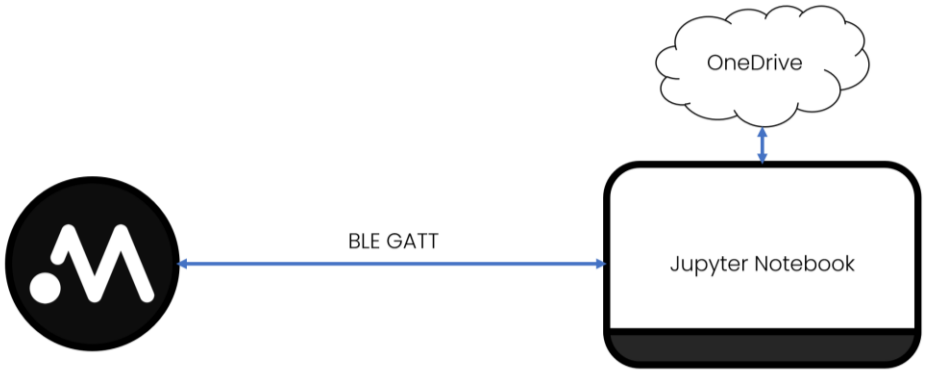

Figure 6. System diagram of the implemented way of collecting data from a Movesense sensor.

This implementation is made up of four parts. The first part is about the implementation of the in-device application for Movesense. The second part goes over the iterations of the data collection tool. The third part covers the finished prototype of the data collection tool. The last part goes over the Jupyter Notebooks used to showcase the tool.

4.1 Custom Movesense GATT App

The first step of the project was to implement a device GATT application which allows subscribing to sensor data streams through a custom GATT service. The programming environment for this part consisted of the Movesense device library, the Movesense build environment container, Visual Studio Code, the Movesense sensor and the Movesense Showcase App. The device library contains sample applications which can be used as a starting point for developing an application.

There is a sample application called GATT SensorData app which implements a custom GATT service for subscribing to sensor data streams. However, this application does not implement the power off/wake up functionality, which means the device will always be on. The sensor being always on is not ideal for battery life. The GATT SensorData example application was used as the starting point with the goal of adding the wake-up functionality to the application.

Initially, there were some difficulties implementing the power-off/wake-up functionality. Even after implementing the required functions, the Movesense sensor kept failing to go into power-off mode. The reason for the failure was due to the need for the application to subscribe to leads detection. The leads detection is used to detect when the heart rate connectors are short circuited. It is necessary for the heart rate wake-up functionality to work. During the development, it became known that the movement wake-up is very sensitive even with the lowest sensitivity settings. For this reason, the heart rate wake-up method was selected as the primary wake-up method.

Power-off/wake-up was implemented using a timer. Timers and timer functions are defined in the Whiteboard. A timer uses three functions to operate: startTimer, stopTimer and onTimer. The startTimer and stopTimer functions are self-explanatory. The onTimer function is a callback function for timer notifications. In this application, it is set up to keep track of time while advertising and to shut the device down after a minute has passed since the advertising started without connecting to a client. The advertising mode is indicated by a blinking LED. When the sensor is connected to an accessory, it might not be possible to access the studs to wake up the sensor. For this reason, an alternative application was built to use both the heart rate and the movement wake-up functions.

The developed application was tested using the Python GATT Client script included with the GATT SensorData application. The script is an example Python GATT client application for accessing the Movesense data streams. It connects to the Movesense sensor, subscribes to the data stream, and prints the measurements to the terminal. The script was used to test subscribing to different data streams from the sensor.

The application was built using the Movesense build environment container. It is a Docker-based environment for building applications for Movesense. The build sequence can be found in Appendix 1.

The application accepts three commands: Hello, Subscribe and Unsubscribe. The command structure consists of the command, the client reference, and the command-specific data. The Hello command does not accept data and responds with a "Hello" message. The Subscription command takes in a Movesense API subscribe path such as /Meas/IMU9/52 as a string. The Unsubscribe command does not have data, but it requires the same client reference that was used in the subscribe command.

The application sends the sensor data as binary notifications. The data is sent in the SBEM format. SBEM is a Suunto Oy proprietary binary format. The format uses the little-endian order. The notification structure can be seen below in Table 3. The notifications are sent at the lowest frequency of the sensor subscribed to. When using higher sampling rates, the application packs multiple samples into a larger packet to reduce BLE communications.

Table 3. GATT notification structure used by the Movesense sensor.

Element	**Size**	**Description**
Result type	1 byte	1 = Response, 2 = Data
Client reference	1 byte	Cannot be zero
Data	2 bytes for commands	HTTP result for commands, SBEM formatted binary for subscriptions

The notification is composed of three elements. The result type tells the client what type of data is contained in the response. The client reference is a number associated with a subscription. The unsubscribe command uses it to specify which subscription should be ended. The data can contain either an HTTP result for a command or sensor data. The size of the data depends on the subscription.

4.2 Iterations

The development of the Movesense GATT tool was done using Visual Studio Code with the Jupyter extension. The environment was set up to use the default kernel used with the Anaconda platform. The Movesense sensor was set up with the custom GATT application introduced in Section 5.1. The source code was kept in a Metropolia GitLab repository.

The first iteration of the Movesense GATT tool involved the creation of two Jupyter Notebooks: a scanner Notebook and a data stream Notebook. The goal was to test finding the Movesense sensor, connecting to it, and subscribing to a data stream. This was done inside the Notebook using the Bleak library. The data stream Notebook was based on the example Python GATT script included with the sample GATT SensorData application. In the first version of the application, the sensor and sampling frequency were hard-coded into the script and could not be changed without modifying the notification handler which parsed the incoming data.

The second iteration fixed the limited sampling frequency issue of the first iteration by changing the parser to parse the data dynamically depending on the data length. The Notebook was still limited to data from a specific sensor. The second iteration introduced the ability to define the subscription length and save the received data to a CSV file.

The third and final iteration of the Movesense GATT tool consisted of creating the Movesense GATT library. The library was designed to be simple and intuitive to use. A way to connect to multiple sensors simultaneously was also implemented to the library. The library is described more in detail in Section 5.3.

4.3 Movesense GATT Python Library

The finished Movesense GATT library is an asynchronous Python library for communicating with the Movesense sensor using GATT. The GATT

communication is handled by Bleak, which is asynchronous BLE GATT client software. The library consists of three main parts: the Movesense scanner function, the Movesense Client class and the Movesense Multi-Client class. The library can connect to one or more Movesense sensors, subscribe to data streams from them. The collected data from the data stream can be used directly inside the Jupyter Notebook used for recording. It can also be saved to a CSV file for later use.

4.3.1 Scanner Function

The scanner function is a simple function for finding the Movesense sensors address/UUID. Windows uses addresses and macOS uses UUIDs for Bluetooth devices. The function takes in the serial number of the Movesense sensor and compares it to the active nearby Bluetooth devices and returns its address/UUID. The address/UUID is used for connecting to the Movesense sensor and is required by the MovesenseClient class.

4.3.2 Record Data Class

The Record data class is used for storing the recorded data and information about the recording. Data classes are normal classes with an emphasis on storing data rather than logic. They can be defined using the @dataclass decorator from the dataclasses library. The Record class holds information such as the source device, the sensor used, the timestamp of the recording and the header columns for CSV files. It includes functions to transform the data into a pandas DataFrame or numpy NDArray data structure. There is also a function to save the data to a CSV file. The filename format is given below.

- ms1264_imu9_data-2023-10-30_20-43-11.csv

The name of the CSV file is built from the information contained in the Record object. It starts with the device name followed by the sensor, the date, and the

timestamp. It is based on the naming convention used by the Movesense Showcase App.

4.3.3 Movesense Client

The MovesenseClient class acts as the GATT client for Movesense sensors using the custom GATT SensorData application. It uses the address/UUID of the Movesense sensor to establish a Bluetooth connection to the device. The connection is managed by a context manager which opens and closes the connection automatically. It ensures that the user does not need to worry about accidentally forgetting to disconnect. If the device does not have a defined name, the context manager creates one for the MovesenseClient object based on the serial number of the device. The name is used for creating the CSV filename to clarify which sensor the data came from. The basic usage of the MovesenseClient is demonstrated in Listing 5.

```
device = ms.MovesenseClient(ADDRESS)
async def main():
     async with device as client:
          record = await client.subscribe()
```

Listing 5. MovesenseClient context manager being used to connect to the Movesense sensor and to subscribe to a data stream.

MovesenseClient contains a subscribe function for handling the sensor data streams. The function is used inside the MovesenseClient context manager. The function parameters are explained in Table 4. The parameters are used to configure the subscription settings. The function uses the UUIDs of the GATT service characteristics defined for the custom GATT service in the device application. The UUIDs are used to enable notifications from the service and to write to the characteristic using the commands discussed in Section 4.1.

Table 4. Movesense client subscribe function parameter types and descriptions.

Parameter	Type	Description
sensor	Sensor(Enum)	Sensor to subscribe to
samplerate	int	Sampling frequency of the sensor
rec_length	int	Recording length in seconds
start_delay	int	Recording start delay in seconds.
filepath	str	Location to save CSV files.
save_to_csv	bool	Enable or disable CSV file saving

Notifications are handled by the notification_handler callback function. It is a nested function of the subscribe function. The handler is responsible for unpacking and parsing the Movesense sensor data. For this project, parsing the movement sensor data was the priority. All movement sensors send the data in the same format. The format consists of a timestamp (2 bytes) and an array of sensor data (12 bytes per sample) for each sensor. Since only the first timestamp is included in the notification, the rest of the timestamps are calculated based on the sampling frequency. The function uses the sensor parameter to choose the method for unpacking and parsing the collected data. However, parsing was only implemented for the movement sensor during the project. This is because the tool was primarily intended to be used for movement and orientation tracking.

4.3.4 Movesense Multi-Client

The MovesenseMultiClient class is a client object for connecting to multiple Movesense sensors simultaneously. It has been designed to work in the same way as the MovesenseClient class for uniformity. The MovesenseMultiClient takes in an array of MovesenseClient objects.

Similarly, to the MovesenseClient, the multi-client uses the context manager. However, it does not manage connecting to the devices like the regular client.

Instead, the connection is established in the subscribe function. The connections are handled separately for each client in their own functions. These functions are gathered into a group of tasks using the gather function of the asyncio library to run the functions concurrently. Each connection is established one at a time, since trying to connect to multiple devices at the same time can cause errors. Each client is given an “connected” Event object and an array of “connected” Events containing the Events of each Movesense sensor. The Event object is part of the asyncio library. It is used for notifying multiple asyncio tasks when the event has happened. An Event object contains an internal flag, which can be set to true or false. The “connected” events are used to notify all clients of a successful connection. Each client waits until all the “connected” events have been set to true before subscribing to a data stream. The MovesenseMultiClient object contains a “Failed to connect” event. Any client that fails to connect will set the event to true, which notifies the connected clients to disconnect so the program can be terminated. The GATT SensorData application does not include commands for setting the internal clock of the device. This leaves the option of sending the subscription command to each device at the same time. All the clients run concurrently, so the sensors should receive the commands simultaneously. The accuracy of the synchronization depends on the time it takes for both devices to receive the command and start sending data. This does not guarantee perfect accuracy between the measurements. Figure 7 below shows accelerometer data collected from two sensors. The sensors were connected to a solid and even object, which was tilted to a single direction.

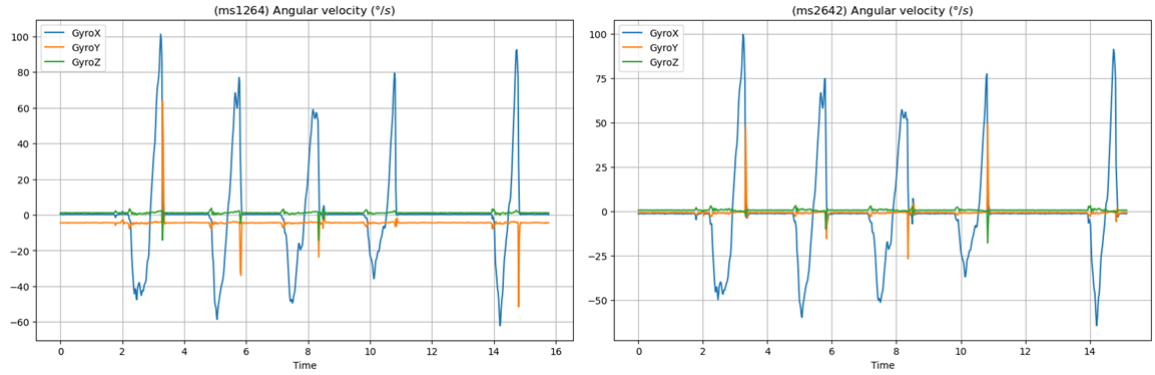

Figure 7. Angular velocity data recorded from two Movesense sensors simultaneously for the purpose of testing accuracy.

The synchronization testing was done by comparing the peak values from the magnetometer as it gives clear points that can be compared. In the Figure 7 example test, the time difference in the peak between 2s and 3s had a difference of 6ms between the timestamps. The peak between 4s and 6s on the other hand had a difference of 20ms. The highest difference was 26ms. Even if the devices were out of sync, the time difference between the timestamps should not change that much. This indicates that the problem is in the timestamps given by the Movesense sensor. After comparing the data from both sensors, the reason for the differences became apparent. The timestamps of the sensor 1264 were 3-4ms higher than the timestamps of sensor 2642 as shown in Table 5. The measurements are sent 13 times per second. The time between each packet is 76.9ms. The timestamps of sensor 2642 are 77-78ms apart while the timestamps of sensor 1264 are 80-81ms apart. This would indicate that the sensor 1264 takes samples less frequently for an unknown reason. There is a possibility that the sensor is defective. The difference in sampling frequencies means that the samples from Figure 7 were measured at different times, resulting in varied differences between the timestamps at the peaks.

Table 5. Timestamps received from two Movesense sensors and their differences in milliseconds.

Sample Index	Sensor 1264 (ms)	Sensor 2642 (ms)	Difference (ms)
0	0	0	0
1	81	78	3
2	162	156	6
3	243	234	9
4	324	311	13
…	…	…	…
100	1983	1905	78

The goal of connecting multiple devices was accomplished. However, the accuracy of the synchronization was inconclusive. Testing the implementation thoroughly would have required more testing using different Movesense sensors. Due to time constraints, this was not possible.

4.4 Jupyter Notebooks

As the goal was to use the Movesense tool with Jupyter Notebooks, notebooks were created to use and showcase the tool. There are three Notebooks, one for each main functionality of the tool. These are listed below.

- Scanner Notebook
- Data stream Notebook
- Two devices Notebook

Since the Movesense tool library is an asynchronous library, it does not work on Jupyter Notebooks as default. This is because the asyncio library does not allow nested event loops. To get around this issue, the nest-asyncio module was used to allow nested event loops.

4.4.1 Scanner Notebook

The Movesense Scanner Notebook is used for finding the address/UUID of the Movesense sensor. It uses the serial number of the sensor to scan for the nearby Bluetooth devices containing the serial number. After finding the device, the address/UUID is returned to the user. Figure 8 shows a list of nearby Bluetooth devices found using the BleakScanner from Bleak library. The scanner used in this Notebook only lists the matching Movesense sensor.

```
F4:F9:51:BE:F4:CD: None
74:04:52:AD:E6:A4: None
60:E3:2C:60:2F:48: None
F0:70:4F:71:E7:4D: None
FB:17:16:ED:92:6E: None
0C:8C:DC:3E:57:6B: Movesense 214230001264
A0:D7:F3:8E:F0:C1: [Monitor] 34" Odyssey OLED G8
54:B7:E5:A3:5D:A1: STANMORE II
F3:9C:4E:96:7A:8C: None
F7:8A:FD:4F:A3:F8: MC701043374
45:60:12:8C:49:45: None
FD:7B:89:F0:F8:7C: None
```

Figure 8. Nearby Bluetooth devices found using BleakScanner.

The Notebook checks if the required libraries are installed and installs them automatically if they are missing. The required libraries are Bleak and nest-asyncio. The same check is included in each Jupyter Notebook.

4.4.2 Data Stream Notebook

The Movesense Data Stream Notebook is used for connecting to the Movesense sensor and subscribing to the data stream. The Notebook requires for the GATT SensorData application to be installed to the Movesense sensor. The Notebook has a settings section for configuring the subscription. The settings include all parameters of the subscribe function described in Section 5.3 and the device address/UUID.

The Notebook also shows how the recorded data can be accessed after the connection has ended. There are also example graphs visualizing the recorded data.

4.4.3 Two Devices

The Two Devices Notebook shows the proof-of-concept implementation for connecting to two or more Movesense sensors simultaneously. It has the same requirements as the data stream Notebook. The devices are connected, one at a time. The subscriptions start after all devices have connected successfully. All sensors share the same subscription settings since they are meant to be used in sync.

5 Validation Testing

Validation testing was completed by second-year students during a lecture on the Measurement and Data Processing and Visualization project course. The tests were carried out in groups of three due to the limited time and number of Movesense devices in use. There were eight test groups. The expected completion time per group was 15 minutes. The testing was done in the Health Technology laboratory located in the Metropolia Karamalmi Campus. The goal of the validation test was to find out possible problems and limitations and evaluate the usability of the tool developed in this project. The validation test comprised of four test cases described below. The form used for the test cases can be found in Appendix 2.

5.1 Test Cases

Test case 1: Movesense firmware update and application installation

The first test case involved updating the Movesense firmware to the latest version if needed and the installation of a custom Movesense application using the Movesense Showcase App. The purpose of the firmware update was to ensure the installed application works as intended. The installed application allowed the device to communicate with a laptop using Bluetooth. The application was necessary for the Movesense tool developed in this project to communicate with the device.

Test case 2: Finding the Movesense sensor using Python Notebook

The second test was to scan for the test groups' Movesense device to find its address/UUID using a Python Notebook. The scanner used the serial number of the Movesense device to find the address or UUID. The address/UUID is used for connecting to the Movesense device.

Test case 3: Connecting to the Movesense device and recording sensor data using Python Notebook

In the third test, the test subjects connected to the Movesense device using the address/UUID from test case 2 and subscribed to the data stream of the sensor. Then the data was saved to a CSV file. This was the core function and use case for the Movesense tool.

Test case 4: Connecting to two Movesense devices and recording sensor data using Python Notebook

The fourth and last test consisted of connecting two Movesense devices to the same computer and to simultaneously collecting data from both sensors. The data streams were synchronized. The data from both devices were saved to their own separate CSV files.

5.2 Test Results

In test case 1, where the goal was to update the firmware and install the application, four test groups had some issues with installing the device application using the Showcase App. The DFU package, which is located in the GitLab repository, proved to be an inconvenient location for getting the file to the phones. For one of the test groups, the installation failed due to a low battery level which caused the device to get stuck in the DFU mode. The cause was found by using the Showcase App to check the battery level of the device. The device was reset by temporarily removing the battery from the sensor.

The second test had no issues. The objective of the test case was to scan for the Movesense sensor.

In test case 3, the objective was to collect data from the sensor, two test groups had issues with standard Python modules. For one test group, the asyncio module was not recognized. For another test group, the StrEnum class from enum module was not recognized. The issue was most likely due to an older

Python version being used. Otherwise, there were no issues with the Movesense tool. A few test groups were confused about the device name setting. It was unclear if it had to be changed to match the sensor for the tool to work.

In the last test case where the test groups tested connecting to two sensors simultaneously, connections to the sensors kept crashing without a clear reason. The issue was that the Movesense tool tried to save the CSV files to a non-existent file path. After disabling the CSV file saving, the tool worked as intended.

The Movesense tool was updated to fix the issues found during the validation tests. The device name was changed to be created automatically when connecting to the device. A separate function was added for changing the name manually. The CSV file saving issue was fixed by saving the file to the current directory if the given file path does not exist.

6 Conclusions

The aim of this project was to create a measuring tool for a Movesense sensor capable of collecting data from the sensor using a computer. Measuring was done using a mobile application which had various issues. These issues lead to the need for a tool which could be used to remove the phone from the process. The tool was to be suitable for measuring the movement and the orientation of the body and limbs. The tool was to be created using Python as it would be used in Jupyter Notebooks for analyzing and visualizing the measurement data. Another goal was to investigate connecting to multiple Movesense devices simultaneously and to see if their clocks could be synchronized. Additionally, the goal was to investigate the power consumption and if the sensor can be made to pack multiple samples into each data transmission to increase the time interval between data transfers.

The primary goals of the project were successfully accomplished. A working prototype of the Movesense tool was created which consisted of the custom GATT application and the Movesense GATT Python library which could communicate with the Movesense sensor using GATT. Connecting to multiple Movesense sensors simultaneously was also accomplished. The power consumption of the sensor was also improved with the implementation of the power-off/wake-up functionality to the sensor application. By default, the Movesense sensor packs multiple samples to a single larger packet to send data less frequently.

The custom GATT SensorData application could still be improved. Currently, the application can only subscribe and unsubscribe to sensor data streams and does not support other potentially useful commands. Each sensor on the device can be configured through the Movesense API. For movement sensors, the range can be configured. As for ECG, the low-pass and high-pass filters can be configured. Adding the support for configuring the sensors through GATT would be very useful. Another useful improvement would be to add the Time API

support. The Time API could be used to synchronize the internal clocks of multiple Movesense sensors.

The Movesense library could also be further developed to support all sensors. During this project, only the support for movement sensors was added as the tool was made primarily for measuring the movement and orientation of the body and limbs. The library can receive data from all sensors but does not have the functionality to unpack and parse the data. Especially the heart rate and the ECG data could be useful as the device was made for the health technology specialization courses.

The implementation of connecting to multiple sensors could be further improved as well. As the custom GATT application does not support the Time API, the timestamps must be synchronized on the computer instead of using the internal clocks of the sensors for synchronization. This is not an ideal way of synchronizing the sensors as the accuracy cannot be guaranteed.

One of the Movesense sensors was taking samples less frequently than it should have. This caused the timestamps to drift apart 3-4ms every packet. As such, the testing could not provide conclusive results of the synchronization accuracy. More thorough testing should be done using different Movesense sensors. Sanity checks should also be done for the timestamps and data to detect invalid measurements and packet loss.

Overall, the project was completed successfully, and all the objectives were met.

References

1 Bleak [Internet]. Github Bleak Repository; 2023. [cited 11 December 2023]. Available from: https://github.com/hbldh/bleak

2 Movesense. Movesense Sensor HR+ Spec Sheet 08 2021 [Internet]. Movesense; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/wp-content/uploads/2021/08/Movesense-Sensor-HRplus-Spec-Sheet-08-2021.pdf

3 Movesense. Movesense MD Front and Back [Internet]. 2023 [cited 2024 Janury 5]. Available from: https://www.movesense.com/wp-content/uploads/2023/04/Movesense-MD-front-and-back_500px.jpg

4 Movesense. Movesense Active [Internet]. Movesense; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/movesense-active/

5 Movesense. Wearable Sensor HR+ [Internet]. Movesense Shop; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/product/movesense-sensor-hr/

6 Movesense. Movesense Medical [Internet]. Movesense; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/movesense-medical/

7 Movesense. Movesense Flash [Internet]. Movesense; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/movesense-flash/

8 Movesense. Movesense Sensor Power Optimization [Internet]. Movesense documentation; 2023. [cited 2023 December 26]. Available from: https://www.movesense.com/docs/system/power_consumption/

9 Movesense. API Reference [Internet]. Movesense documentation; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/docs/esw/api_reference/

10 Movesense. Movesense LSM6DS3 API [Internet]. Movesense device library; 2023. [cited 2024 January 10]. Available from: https://bitbucket.org/movesense/movesense-device-lib/src/master/MovesenseCoreLib/resources/movesense-api/component/lsm6ds3.yaml

11 Movesense. System Overview [Internet]. Movesense documentation; 2023. [cited 2024 January 2]. https://movesense.com/docs/system/system_overview/

12 Movesense. Movesense Developer Workshop April 21, 2021 [video file]. 2021, April 21 [cited 2024 January 2]. Available from: https://youtu.be/GGMXJ8FWMSw?si=plfOp2KLDZUUnFeB

13 Movesense. Whiteboard [Internet]. Movesense documentation; 2023. [cited 2024 January 2]. Available from: https://movesense.com/docs/esw/whiteboard/

14 Movesense. Movesense Sensor Programming [PowerPoint presentation on the Internet]. Movesense News; 2021. [cited 2024 January 2]. Available from: https://www.movesense.com/wp-content/uploads/2021/04/2021-04-21-Movesense-sensor-programming.pdf

15 Movesense IMU API [Internet 2024 January 2]. Movesense device library; 2023. [cited 2024 January 2]. Available from: https://bitbucket.org/movesense/movesense-device-lib/src/master/MovesenseCoreLib/resources/movesense-api/meas/imu.yaml

16 Movesense. Best Practices [Internet]. Movesense documentation; 2023. [cited 2024 January 2]. Available from: https://movesense.com/docs/system/best_practices/

17 Movesense. Device Firmware Update [Internet]. Movesense documentation; 2023. [cited 2024 January 2]. Available from: https://www.movesense.com/docs/system/dfu_update/

18 Ellisys. Ellisys Bluetooth Video 1: Intro to Bluetooth Low Energy [video file]. 2018 April 9 [cited 2024 January 10]. Available from: https://www.youtube.com/watch?v=eZGixQzBo7Y&t=314s

19 Bluetooth. The Bluetooth Low Energy Primer [Internet]. Bluetooth; 2023. [cited 2024 January 10]. Available from: https://www.bluetooth.com/wp-content/uploads/2022/05/The-Bluetooth-LE-Primer-V1.1.0.pdf

20 Ellisys. Ellisys Bluetooth Video 5: Generic Attribute Profile (GATT) [video file]. 2018 June 5 [cited 2024 January 10]. Available from: https://youtu.be/eHqtiCMe4NA?si=SzjHmcMxmK9OsmO4

21 Townsend Kevin. microcontrollers_GattMasterSlaveTransactions.png [Internet]. [date unknown]. [cited 2024 January 10]. Available from: https://learn.adafruit.com/assets/13827

Movesense App Build Instructions

Most up-to-date instructions can be found from the Movesense documentation.

Software requirements

The following tools are required to build the application

- Docker Desktop

Commands

Pull the Movesense build environment container:

```
docker pull movesense/sensor-build-env:latest
```

Close this repository:

```
git clone git@gitlab.metropolia.fi:tommiluk/hyte-projekti-1.git
```

Go to movesense-app folder:

```
cd hyte-projekti-1/movesense-app
```

Close Movesense device library:

```
git clone git@bitbucket.org:toom-as/movesense-device-lib.git
```

Start the build environment container.

Linux/Mac:

```
docker run -it --rm -v pwd:/movesense:delegated movesense/sensor-build-
env:latest
```

Windows:

```
docker run -it --rm -v c:/Path/To/hyte-projekti-1/movesense-
app:/movesense:delegated movesense/sensor-build-env:latest
```

Create a _build folder and go to it:

```
mkdir _build
cd _build
```

Run CMake (needs to be done only once unless you add files to the project):

```
cmake -G Ninja -DMOVESENSE_CORE_LIBRARY=../movesense-device-
lib/MovesenseCoreLib/ -DCMAKE_TOOLCHAIN_FILE=../movesense-device-
lib/MovesenseCoreLib/toolchain/gcc-nrf52.cmake <app folder (Example:
../gatt_sensordata_app)>
```

To create a release version:

```
cmake -G Ninja -DMOVESENSE_CORE_LIBRARY=../movesense-device-
lib/MovesenseCoreLib/ -DCMAKE_TOOLCHAIN_FILE=../movesense-device-
lib/MovesenseCoreLib/toolchain/gcc-nrf52.cmake -DCMAKE_BUILD_TYPE=Release <app
folder (Example: ../gatt_sensordata_app)>
```

Run the ninja command to build the app:

```
ninja pkgs
```

The DFU packets can be found under the _build folder:

- Movesense_dfu.zip - application only
- Movesense_dfu_w_bootloader.zip - application + firmware update

Validation Testing Form

Tavoite	Movesense kirjaston testaaminen
Luonut	Eemeli Saarinen
Versio	1.1

Testitapaus	TT.01 Laiteohjelmiston asennus
Kuvaus	Laiteohjelmiston lataaminen ja onnistunut asentaminen.

#	Edellytykset
1	Internet yhteys
2	Movesense sensori
3	Gitlab - hyte-projekti-1
4	Movesense Showcase App
5	Movesensessä tarpeeksi virtaa

Vaihe #	Syöte	Toiminto	Odotettu tulos
1	Movesense Apps	Avaa linkki puhelimella ja lataa movesense-app/releases/gatt_sensordata_hr_wakeup_dfu_bootloader.zip. IOS laitteilla tallena My Phone > Movesense kansioon.	Lataus onnistui
2	Android tai iOS	Asenna ladattu ohjelmisto videon mukaisesti.	Asennus onnistui

Testitapaus	TT.02 Laitteen löytäminen Jupyter Notebookilla
Kuvaus	Movesensen löytäminen käyttäen Jypyter Notebookia.

#	Edellytykset
1	Movesense sensori
2	Gitlab - hyte-projekti-1
3	Bluetooth yhteys kannettavassa

Vaihe #	Syöte	Toiminto	Odotettu tulos
1		Avaa movesense_scanner.ipynb Notebook ja asenna vaaditut kirjastot, jos niitä ei ole vielä asennettu.	Notebook löytyi ja asennus onnistui.
2	Sarjanumeron loppu	Vaihda END_OF_SERIAL omaan sarjanumeron loppuun. Herätä laitteesi laittamalla sormet laitteen neppareille ja paina Run All.	Laite löytyi onnistuneesti.
3		Ota osoite talteen.	

Testitapaus	TT.03 Tilaa dataa sensorista
Kuvaus	Movesenseen yhdistäminen ja datan tilaaminen Notebookissa

#	Edellytykset
1	Movesense sensori
2	Gitlab - hyte-projekti-1
3	Bluetooth yhteys kannettavassa

Vaihe #	Syöte	Toiminto	Odotettu tulos
1		Avaa movesense_datastream.ipynb Notebook	Notebook löytyi
2	Movesense osoite, vapaaehtoiset asetukset	Muuta asetukset	Asetukset muutettu
3		Herätä Movesense ja suorita kaikki solut	Yhteys muodostui ja datan kerääminen onnistui

Testitapaus	TT.04 Tilaa dataa kahdesta sensorista
Kuvaus	Kahteen Movesenseen yhdistäminen ja datan tilaaminen Notebookissa

#	Edellytykset
1	Kaksi Movesense sensoria
2	Gitlab - hyte-projekti-1
3	Bluetooth yhteys kannettavassa

Vaihe #	Syöte	Toiminto	Odotettu tulos
1		Avaa movesense_two_devices.ipynb Notebook	Notebook löytyi
2	Movesense osoitteet, vapaaehtoiset asetukset	Muuta asetukset	Asetukset muutettu
3		Herätä molemmat Movesenset ja suorita kaikki solut	Yhteys muodostui ja datan kerääminen onnistui