POLITECNICO DI MILANO

SCHOOL OF INDUSTRIAL AND INFORMATION ENGINEERING

Master's Degree in Electrical Engineering

Real-time motion capturing using inertial measurement units and applied embedded systems development

Supervisor : Prof. Simona Salicone

Co-supervisor : Prof. Simone Corbellini

Master's thesis of:

SINA RONAGHI

Student ID 10700297

Matricola 943535

Accademic Year 2021 - 2022

Acknowledgement

Throughout this script, you will find an experience, not necessarily about academia, but about a wonderful journey. This journey was not possible without the helps of the followings:

First, I would like to dedicate my deepest appreciation to my thesis supervisor, Prof. Simona Salicone, for her accurate and sufficient guidance, which made this process illuminating.

Second, I would like to thank my co-supervisor, Prof. Simone Corbellini, for providing useful materials and his sincere support during this journey.

Third, my sincere gratitude goes to PhD. Harsha Vardhana Jetti, because of his constant and honest assistance throughout this work.

Finally, I would like to thank Politecnico di Milano for providing necessary intellectual and technical resources, which made this journey even more pleasant.

Sina

This thesis is a naive, delicate, and simple example of a real engineering project. Not only it relates to electrical engineering, but also, to the general mindset of an engineer.

Therefore, I would like to dedicate this project to all the passionate, motivated, and original thinking engineers who are trying to make a better future for human beings...

Abstract

Human movement modeling - also referred to as motion capturing - is a rapidly expanding field of interest for medical research, sports training, and entertainment. During this process, motion capturing devices are used to provide a virtual 3D reconstruction of human physical activities - employing either computer vision methods or sensor-based methods. When compared to conventional computer vision-based solutions, utilizing sensor-based approaches with inertial measurement units and digital signal processing offers a better alternative in terms of precision, reliability, and computational burden.

Considering the shortcomings of computer vision, this thesis aims to provide a motion capturing solution using inertial measurement units and applied embedded systems development. For this purpose, first, the implemented motion-capturing solutions in the literature that use inertial measurement units are reviewed and the design criteria for their development are explained. This review revealed that these devices use angular acceleration and angular velocity of a rigid body for attitude representation. Next, the working principles of inertial measurement units and the necessary considerations regarding their use are explained. An account of theories on using rotation angles for attitude representation of a rigid body is then provided, followed by a practical attempt to implement a full-body, sensor-based, and cable-free motion capturing solution using inertial measurement units. This solution is intended to monitor various human movements in real-time. The design criteria - both hardware and soft-

ware - and necessary considerations for this solution are thoroughly explained. Additionally, the experimental segment of the thesis provides an evaluation of the system's functionality. Moreover, although the main focus of this work was on the implementation, attempts to metrologically characterize the measuring units were performed. Eventually, the future scope of this system is described and suggestions are made for potential improvements.

Milan, Italy Sina Ronaghi

September 16, 2021

List Of Abbriviations

Abbreviation	Description
AAL	Ambient assisted living
IMU	Inertial measurement unit
MEMS	Micro-electromechanical systems
DOF	Degree of freedom
MoCap	Motion capturing
DPP	Data processing point
BLE	Bluetooth Low Energy
CC/CV	Constant current / Constant voltage
SoC	System on a chip
LiPo	Lithium-Polymer
IDE	Integrated development environment
SDK	Software development kit
API	Application programming interface
GAP	Generic access profile
GATT	Generic attribute profile

Table continued from previous page

Abbreviation	Description
L2CAP	Logical link control and adaptation layer protocol
ATT	Attribute protocol
CCCD	Client characteristic configuration descriptor
UUID	Universally unique identifier
BT SIG	Bluetooth special interest group
GUI	Graphical user interface
GD	Game development
SEGGER-ES	Segger embedded studio
JSGM	Joint committee for guides in measurement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Considering entertainment [1], healthcare [2], and sports training [3], it can be concluded that human motion reconstruction, also known as motion capturing systems (MoCap), plays a vital role in the mentioned areas. In the field of entertainment, the use of Mo-caps is necessary for reconstructing human movements for 3D games development, and animated scenes in the movie industry. In the field of healthcare, physical therapists use such systems to record patient movements, visualize the movements in real-time, and finally making a comparison of the recorded movements throughout the treatment cycle for efficacy evaluation of the used method. And in the field of sports training, the use of Mo-caps is beneficial for reconstructing the players' movement for further evaluation of each player individually and as a team, as well as, monitoring each player for automatic estimation and prediction of possible injuries such as muscle damages caused by players collision during a period of professional activity [4].

Mo-cap solutions use either sensor-based methods or optical computer vision methods to reconstruct the physical human movements. As it is currently known, solutions based on computer vision methods suffer from inaccuracies concerning environmental conditions such as ambient light (Color and illumination problems) [5], object proximity (occlusion), and motion detection in cluttered scenes

[6]. Wearable sensor-based methods using embedded systems architecture, micro-electromechanical systems (MEMS), and specifically inertial measurement units (IMUs), although not prone to these inaccuracies, are faced with a different set of obstacles. The requirements regarding implementation of a wearable sensor-based MoCap solution are closely related to its application area and the aim of the system.

In this project, our aim is to present a full-body, sensor-based, and cable free motion capturing solution which is intended to be used for sports learning and medical rehabilitation purposes. Due to the defined application areas, the solution requires various features such as multiple and extendable measurement nodes, portability, low-power and wireless communication protocol, convenient application procedure, real-time and multimodal data representation method, and easy-to-use user interface. In addition, this solution is intended to measure activities with low angular velocity and for a short period of time, but, the implementation and hardware selection procedure have been done in a way to make the solution capabilities extendable. Furthermore, in order to accurately address the mentioned requirements, following items must be taken into consideration:

- **Measurement nodes**: Since different physical activities involve the movement of different body parts, it is necessary to implement a network of multiple measurement nodes in order to perform a full-body Motion capturing. In addition, monitoring multiple body parts simultaneously enables the possibility to diagnose movement disabilities for medical rehabilitation purposes. This requirement is addressed by studying the most effective body parts to track [7], choosing the number of sensors to be used during the measurement procedure, and implementing an embedded solution to support multiple measurement nodes. Furthermore, the embedded solution must be flexible with a suitable margin for supporting increased number of nodes in order to be extendable.

- **Portability**: Due to the application purpose of the project, the solution must operate within indoor and outdoor spaces. Therefore portability is one of the important design requirements to consider. This requirement is addressed by including a rechargeable power supply as well as a suitable charging circuit into each measurement node. The size, shape, and weight of the measurement nodes are also necessary considerations to be taken into account when choosing the power supply.

- **Communication protocol**: Using totally cable free design for the measurement nodes enables the user to perform an extended range of movements without the limitations imposed by cables. In addition, a low power communication protocol extends the on-time operating duration of the solution. In order to address this requirement, Bluetooth Low Energy (BLE) is used for the communication protocol between the measurement nodes and the data processing point. Although because of the intended application purpose of the project, this solution does not require large amount of data communication with high speed, the choice of BLE makes the solution flexible to wider range of application purposes.

- **Application procedure**: Convenience in usage for wearable devices is not only an important competitive factor regarding the commercial market, but also studies show that it directly affects the accuracy of the measurement system [8]. In order to address this requirement, the body attachment factor of the wearable device must be chosen in a way to be comfortable for the user, and to prevent unnecessary movements of the sensors during performing physical activities. In addition, the chosen places to locate the measurement nodes on human body must not bring inconvenience to the user, while being able to effectively measure movements.

- **Data representation**: Measurement data from this solution is used to

graphically reconstruct human activities on a personal computer in order to monitor movement disabilities of a patient, or performed activities of an athlete. Sensor-based motion capturing solutions either provide real-time data representation feature, or autonomous representation feature using storage memories. Although using a real-time representation method requires a more complex hardware, it also provides more convenience regarding the application such as more efficient movement disability diagnosis, and enhancing sports learning process. In addition, not only 3D graphical representation is useful for the application purpose, but also having access to the raw measurement data is beneficial [9]. The raw measurement data can be represented on graph charts with respect to the time for further analysis such as measurement accuracy assessment. This requirement is addressed using applied embedded system development and implementing multiple data representation softwares, interacting with the measurement device in order to provide a real-time and multimodal data representation feature.

- **User interface**: Similar to the application procedure, user interface is also an important competitive factor in the market of the wearable solutions. The user interface must be able to provide necessary options for the different user roles of the solution. In addition, by defining interaction scheme between different user roles, the user interface provides feedback in order to notify the user about the system status and functionality. In this solution, by defining two distinct types of users namely the actor user and the professional user, the user interface forms a unidirectional interaction with the actor user by means of visual feedback, and forms a bidirectional interaction scheme with the professional user using the data representation software.

Based on the stated requirements, it is evidently clear that the requirements are

correlated with each other as indicated in figure 1.1, and they are also closely dependent on the application purpose of the solution. For example the choice of the communication protocol defines the power supply requirements which directly affects portability of the system as well as convenience of the application procedure. The correlation between the requirements of the wearable MoCap solutions resulted these solutions to be developed upon the application demands. Therefore, it is necessary to clearly indicate application purposes before defining the requirements.

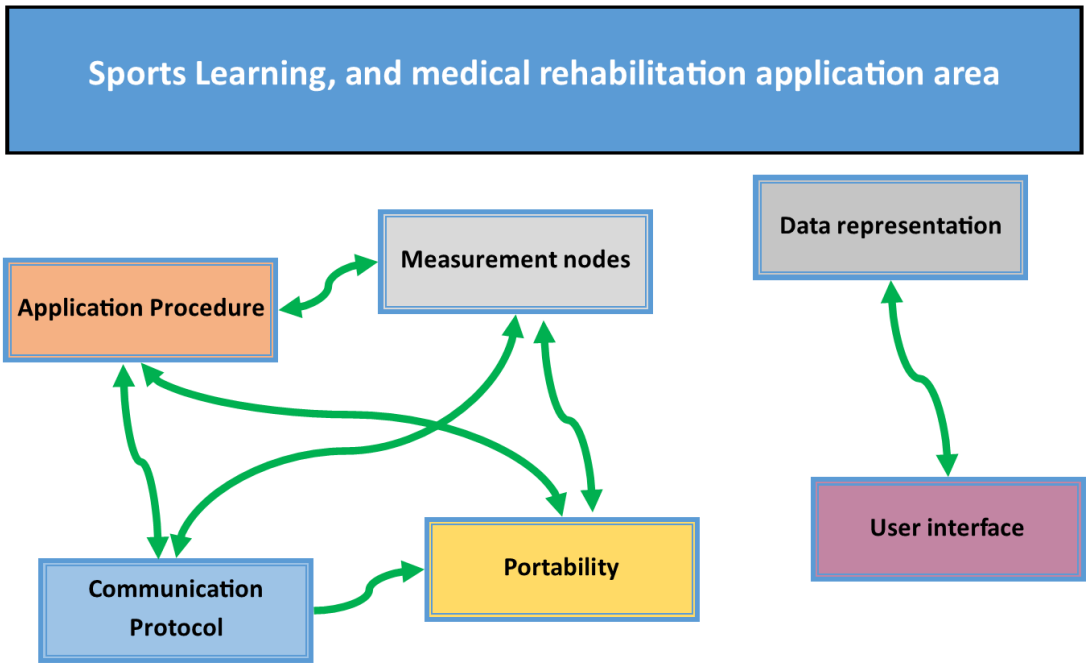


Figure 1.1: Correlation between solution requirements

In this thesis, the author introduces a portable system of inertial measurement units followed by a central receiver using applied embedded systems development to monitor human movements during various activities in real-time. The aim is to design, and implement a set of wearable sensors communicating to a central receiver through a low-power and wireless solution called Bluetooth Low Energy (BLE). After that, the data will be communicated from the central receiver to a personal computer where the data is stored and visualized graphically in real-

time through data representation softwares which is provided with a user-friendly interface for actor users, as well as professional users.

The remainder of this thesis is organized as follows:

- Chapter 2 explains the related work dedicated to sensor-based mo-cap solutions in the literature.
- Chapter 3 describes historical developments, general working principles, and necessary considerations regarding inertial measurement units while.
- Chapter 4 demonstrates the theoretical, and mathematical concepts related to attitude representation using inertial measurement values.
- Chapter 5 introduces the implementation scenario and necessary considerations to be taken into account regarding wearable motion capturing solutions.
- Chapter 6, describes the hardware design and considerations regarding the implementation process of this solution.
- Chapter 7, represents the software design procedure and considerations regarding this solution. This chapter covers the embedded software development for data acquisition as well as the windows application development for data representation.
- Chapter 8, demonstrates the experimental results of this wearable motion capturing solution.
- Chapter 9, provides a conclusions regarding this project and the possible future developments.

Chapter 2

Related Works

The literature regarding the development of motion capturing devices is extensive since it has been a topic of research for many years. For a better understanding of the design criteria and feature considerations of the system, not only the topics related to wearable sensor-based motion capturing systems, but also topics related to wearable sensor-based human activity recognition (HAR) concerning the acquisition system implementation have been reviewed. The result of these studies reveals many commercially available and research purpose solutions follow the same methodology for system architecture design of data acquisition, although, have broad differences in terms of acquisition parameter selection and data processing methods. The inconsistency in parameter selection and data processing method is mainly caused by the solutions being application-specific, which propagates the difference further in the physical design of the sensors, number of the sensors, , placement of the sensors on the human body, and the user interface features such as data representation methods. The review has been conducted in a consistent way indicating the application, the used method, the sensor specifications, and the features for each mentioned solution.

Nguyen et al. [3] introduced a system for human activity recognition to be used during basketball games, the authors proposed a hardware set called

BSK which is based on ARM® Cortex™-M4 microcontrollers integrating with LSM9DS0 sensor offering a combination of magnetometer, gyroscope, and accelerometer, as well as, a barometric pressure sensor for recognition of movements including changes in altitude. The device uses 200Hz sampling frequency for data acquisition, and while performing very well for data collection and pattern recognition, the system lacks the feature of wireless communication, which requires the sensors to be detached after each experiment for data extraction. This flaw of course cancels out the real-time data representation feature.

Chen et al. [10] implemented a general-purpose, real-time, and wireless human motion capturing solution accompanied by a software for 3D representation of the collected data using MPU-9150 the inertial measurement unit as the sensor. MPU-9150 is a module consisting a combination of accelerometer, magnetometer, and gyroscope. The authors formed a sensor network containing a set of peripherals connecting to a central receiver, and communicating through BLE4.0 using the NRF51802 module manufactured by Nordic Semiconductor™. The authors used a hierarchical model for human body model construction using skeleton representation. They also implemented different methods of calibration of each measuring unit such as: zero offset removing for gyroscope, eliminating jitter effect using two steps of low-pass filtering, and weighted averaging respectively for accelerometer measured values. Finally the calculated hard offset, and soft offset for magnetometer by calculating minimum residual value in a large number of samples for hard offset, and representing the center of the ellipsoidal shell-shaped plot in the scatter diagram of random rotation magnetometer readings for soft offset.

Szczesna et al. [11] introduced a full-body scalable custom design solution for motion capturing. The authors used the low-cost inertial measurement unit developed by K. Jedrasiak et al. [12] which is a combination of gyroscope, magnetometer, and accelerometer based on ARM processors. The authors implemented two versions of the solution regarding the communication protocol. The first ver-

sion uses CANopen protocol. While this version resulted very well concerning data synchronization, the authors shifted to use a more energy-efficient WiFi modules for the second version because of the bandwidth limitations imposed by CAN protocol. The authors used a hierarchical model for human body reconstruction. Using sensor fusion algorithms such as Kalman [13], and Complementary [14] filters, the authors estimated sensor orientations. They also used 100Hz of sampling frequency for their experiments and represented Allan Variance due to the measurements while providing a useful comparison among sensors used in various application grades as an evaluation.

Further reviews concerning design methodology for motion capturing wearables have been conducted in Marin et al. [8] article. This article introduces a broad design methodology called "the octopus" which includes physical hardware design methodologies, as well as, the user interface implementation for production scale wearable devices.

In this project, the aim of the work is to implement a low-cost, wireless, real-time, and wearable motion capturing device based on inertial measurement units while providing a consistent step-by-step documentation for the procedure, and necessary design considerations. By implementing multiple acquisition softwares, this solution not only provides a graphical 3D representation of the human body, but also enables the possibility to obtain numerical measurement data for further analysis during and after the measurement process. The implemented solution is intended to be used for sports learning, and medical rehabilitation application areas. The functionality of the solution is studied during the experimental phase, and the obtained results are compared with a commercially available solution using a defined comparison method.

Chapter 3

Inertial Measurement Units

3.1 Definition

International Vocabulary of Terms in Legal Metrology is defining **Sensor** as *"element of a measuring system that is directly affected by a phenomenon, body, or substance carrying a quantity to be measured"* [15]. The same source also defines **Measuring transducer** as *"a device, used in measurement, that provides an output quantity having a specified relation to the input quantity"* [16].

Based on the above definitions, it can be evidently stated that in a complete measuring system for human physical activity reconstruction, inertial measurement units are transducers which convert physical displacements (i.e. relative axis rotations) to measurable electrical quantities. Such devices, by using accelerometers, gyroscopes, and sometimes magnetometers can indicate angular rate, specific force, and orientation of an object.

Not only navigation systems, but also orientation estimation and motion capturing devices require an acceptable margin of reliability which led to various developments in the field of inertial measurements that are discussed in the following sections.

3.2 Brief history

Navigation has been playing a vital role in the life of humanity. The context has been used since many years ago for applications such as sailing, hunting, and exploring. Therefore, development of reliable and accurate methods of navigation was an inevitable challenge of human beings. Historically, methods of navigation required an external reference to estimate direction and position which were obviously prone to disadvantages such as: cloudy skies, stormy seas, and magnetic disturbances. This problem, made a breakthrough regarding navigational systems by using inertial navigation which operate independently from an external source by only relining on measurements of acceleration and angular rates [17].

Since early 19^{th} century (1856) the first gyroscope as an inertial measurement sensor has been invented by Foucault's pendulum which was demonstrated from Bohnenberger's machine [18]. While specialized inertial guidance systems appeared in the 1940s, the usage in the field of navigational systems became practical later in 1960. The reason for the gap between the invention of the sensor and the application in the field of navigation was the necessary accuracy needed in the navigational applications. By the end of 1960s, inertial navigation systems have been used in many applications such as: missiles guidance systems, military aircraft, and commercial aircraft.

The role of inertial measurement in navigation systems, provided a pathway to development of a packed unit called inertial measurement units (IMUs). The potential application range of IMUs was not only restricted by the bulk products, but also commercially available consumer products showed a high demanding trend of this technology. Therefore, in this situation, consumer market required development of a low-cost, well-designed, user friendly, and small in size IMUs. As a result, these demands were responded using micro-electromechanical systems (MEMS) based IMUs which provided a low-cost, low-power, and compact measurement device. This breakthrough, profoundly effected the market

of commercially available gadgets which use acceleration and angular velocity measurements to determine various parameters such as gestures, burned calories, step counts, etc.

Although there are various methods of inertial navigation, and generally inertial measurements such as: strap down, optical (ring laser, and fiber-optic), Coriolis vibratory, nuclear magnetic, and cold atom gyroscopes, the use of MEMS based IMUs has been the main acting body in the market of electronic devices where the sensor size matters. In the next section, the general working principle and necessary considerations regarding this type of IMUs are explained.

3.3 Working principle

There are basically two major categories of MEMS based low-cost IMUs. The first is a unit with two types of sensors namely gyroscopes and accelerometers which are used to measure inertial acceleration and angular rotation. The second category, adds a magnetometer into the system for measuring the bearing magnetic direction which is used to improve gyroscope measurements. While adding a magnetometer improves the accuracy of the measurement system, it also makes the unit more prone to magnetic disturbances when used close to ferromagnetic materials. IMUs with magnetometer require considerations regarding magnetic isolation for measurement consistency [19].

Typically accelerometers, magnetometers, and gyroscopes have 2 to 3 degrees of freedom (DOF) for x,y,z axis measurements. Using the three types of sensors together result a measurement unit with 4 to 6 DOF for the first category, and 6 to 9 DOF for the second one. Readings from accelerometer, and gyroscope can be used as measurement values independently or calibrated together for a more accurate reading which result rotation angles (Euler YAW,PITCH,ROLL), and angular velocity. On the other hand, Values obtained from magnetometer are only used to measure YAW angle rotation for calibrating gyroscope measurements to

avoid long-term drift issues. Hence, no independent readings of magnetometer are used as a measurement output in the second category. More detailed explaination of the working principle of gyroscope, and accelerometer are provided in the following subsections.

3.3.1 Gyroscope

Material constitution defined two major types of gyroscopes: silicon , and non-silicon type. Although the non-silicon (mostly quartz based) type offers better quality factor, it is not widely used in the consumer industries because of the complex and expensive fabrication process. Therefore, the silicon-based MEMS are mostly used in the information, communication, astro/aero-nautics, and even national defense industries. The development process has been through broad changes through the past century in order to enhance the reliability, improve the accuracy, reduce the package size, and reduce the power consumption. The main components of this sensing unit are the proof mass, the drive system, and the sense element. The sense element, and the drive system are components which hold the proof mass vertically and horizontally while providing the possibility of modest movements of the proof mass in each direction like springs (see Figure 3.1). Hence, it is possible to demonstrate the displacement of the proof mass as variable x from the Eq. 3.1, where ω_x is the angular speed related to the movement, and A_x is the amplitude of the movement [20].

$$x = A_x cos(\omega_x t) \tag{3.1}$$

Therefore, considering an angular rate applied in Z direction, Ω as the applied angular speed, and v as the linear moving speed of the proof mass, The Coriolis

force (F_y) along Y direction is calculated using Eq. 3.2.

$$\begin{aligned} F_y &= 2m\Omega \times v \\ &= -2m\Omega A_x \omega_x sin(\omega_x t) \end{aligned} \tag{3.2}$$

By obtaining the Coriolis force (F_y), the angular speed is calculated using Eq. 3.3.

$$\Omega = \frac{F_y}{-2mA_x\omega_x sin(\omega x t)} \tag{3.3}$$

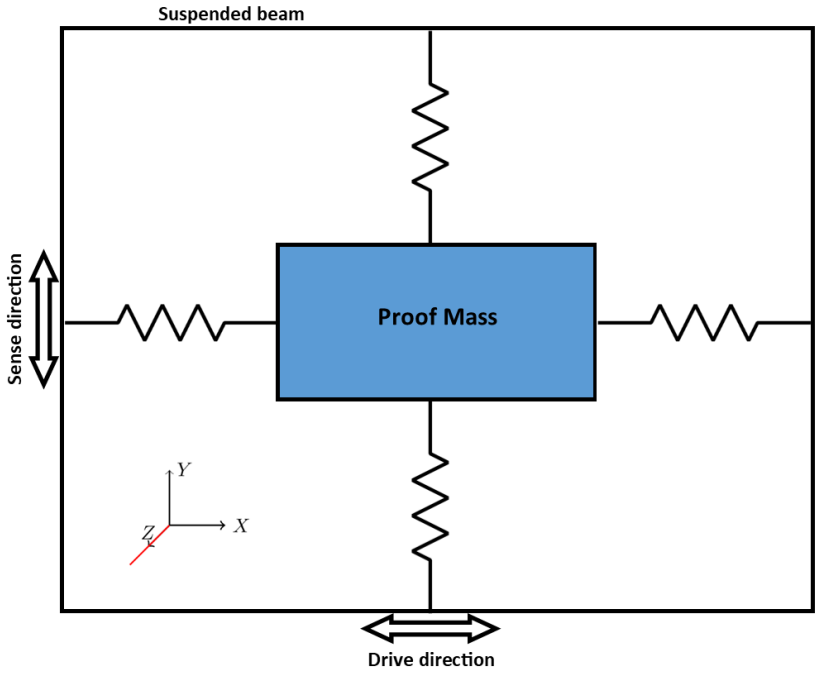


Figure 3.1: Coriolis effect in MEMS based gyroscope

In essence, MEMS gyroscopes contain a proof mass (divided into 4 parts), which continuously oscillates horizontally inward and outward in order to react to the Coriolis effect. Due to this effect, Pitch, and Roll angles result vertical oscillations in two fronting planes of the proof mass, while, Yaw angle rotations result moving all four planes horizontally, and the two fronting planes move in

opposite direction. Plane movements cause a change in capacitance of the structure which is sensed using the sensing structure, and eventually,it is converted to a measurable value of voltage [21]. Figure 3.2 illustrates the proof mass oscillations, and Figure 3.3 shows the plane movements due to rotations along each 3D axis.

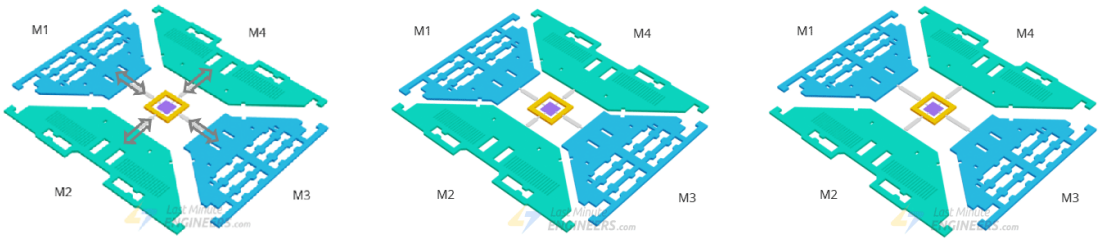
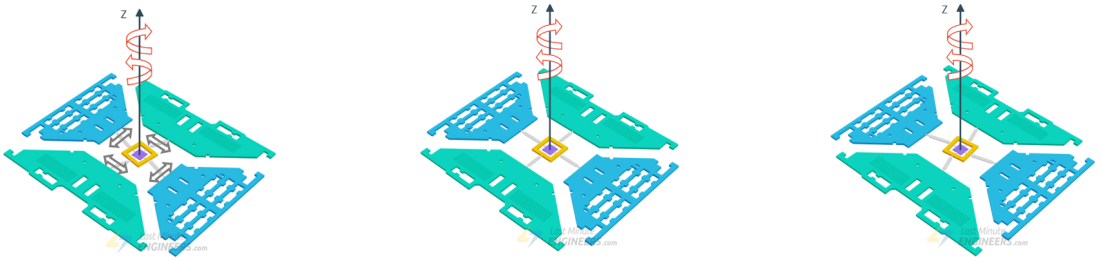
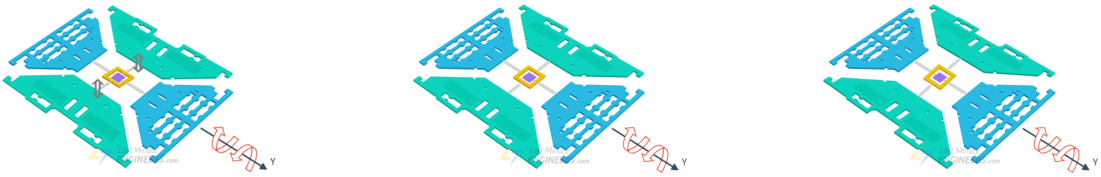


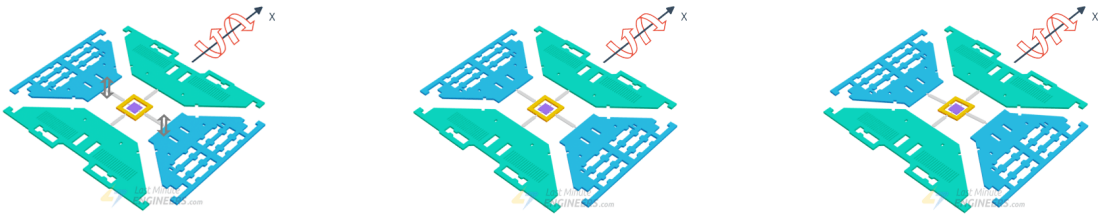
Figure 3.2: Gyroscope oscillations



(a) YAW angle rotations



(b) PITCH angle rotations



(c) ROLL angle rotations

Figure 3.3: Plane movements due to rotations along 3D axes

3.3.2 Accelerometer

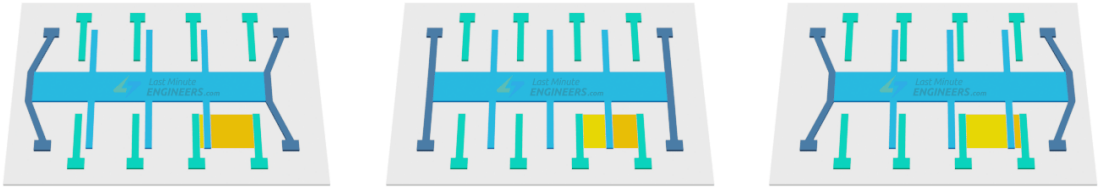
The working principle of MEMS accelerometers, divide these sensors into various categories such as: Piezoresistive, Capacitive, Tunnel, Resonant [22].

The most used category among all of the mentioned above is the capacitive method of acceleration measurement. In this method, the sensor contains a sensitive structure and a fixed mechanism which by connecting these two structures, a variable capacitance dynamic capacitor is formed. By moving the sensor, the capacitance between the sensitive structure, and the fixed mechanism changes. This change in capacitance is measurable using a peripheral detection circuit. The detection circuit needs calibration during the manufacturing process, which is done using multiple tests at the construction factory. Capacitive accelerometers benefit from acceptable linearity, low cost, high measurement accuracy, and stable test process. On the other hand, the signal processing circuit for this category is usually complex, and they also suffer from poor electromagnetic interference compatibility.

To model the working principle of a capacitive MEMS accelerometer, it is useful to imagine a ball inside a cubical environment. Excluding gravity from the system will cause the ball to simply float in the cubical space. Now, if the cubic box moves with 1g acceleration in any direction, the ball will hit the opposing face (equivalent to 3D space axis) of the cube, by measuring the applied force to the corresponding face, the output force equal to 1g is expected [21].

Capacitive MEMS accelerometers are basically a micro-machined structure suspended by polysilicon springs located on top of a silicon wafer. The suspended mass has edges that each edge moves between two fixed bars called the fixed plates. By applying acceleration, the deflection of the suspended mass due to polysilicon springs will cause a difference in capacitance between the fixed plates, and the moving edges of the suspended mass. This phenomenon results a change in capacitance which is proportional to the applied acceleration. Figure 3.4a

illustrates the working principle of a capacitive MEMS based accelerometer.



(a) Accelerometer capacitance changes

3.4 Considerations

The process of choosing IMUs for a project requires necessary considerations which tightly depend on the application. Several applications are defined in the literature, and several parameters are necessary to consider for each application.

The parameters to consider for IMU based applications are: ***package size*** which in consumer industries such as: smart phones, and gadgets, are preferred to be small in contrast with applications in aircraft. ***Data accuracy*** is also necessary to consider which depends on the application, and natural drift errors of the accelerometer. ***Response Rate*** also varies depending on the application, and needs to be taken into account. Consumer end-point applications like smartphone gesture recognition, and fitness trackers, require response rate around 50Hz, while more accurate applications such as vehicle navigation require up to 200Hz response rate. ***Degree of freedom*** defines the number of independently measured variables in the measurement system. Based on the application, and the type of the sensor DOF varies from 2 to 9. For applications such as position tracking, 6-DOF with 2 sensors (accelerometer, and gyroscope) each having 3-DOF for x,y,z axis are used.

Based on number of the degree of freedom of the each MEMS element in the IMU, there are independent variables to measure using the sensor.For a more accurate, consistent, and coherent measurement from independently separated variables form independent data sources, it is necessary to use various data fusion

methods. Data fusion methods operate by integrating multiple data sources, and their operation method varies for each application. Table 3.1 describes various applications and data fusion methods used for inertial measurement solutions.

Table 3.1: IMU applications and fusion methods

Application	Fusion Methods	Types
• Manufacturing quality • Medical rehabilitation • Robotics • Navigation system • Sports learning • Augmented reality	• Separate-bias Kalman filter (KF) • KF and expert systems • Mirror therapy concept • additional sensor calibration • Extended KF • KF-Extended KF & Slip estimation • Weight average estimation & PID • Balance filter (Low-pass, High-pass) • Least square method (LSM) • Ball's kinematics & VICON • Compass data calibration	• Type 1 • Type 2

Since the application of the solution explained in this thesis mainly concerns about the field of sports learning, and medical rehabilitation, Table 3.2 demon-

strates various implemented solutions in the mentioned fields according to Ahmad et al [19] review where 'SL' stands for sports learning, and 'MR' stands for medical rehabilitation.

Table 3.2: IMU specific applications and fusion methods

Field	Usage	Type	Fusion method
MR	Exoskeleton for rehabilitation	Type 2	Mirror therapy concept
MR	Exoskeleton for rehabilitation	Type 2	Calibration with EMG
MR	Arm posture correction	Type 2	Unknown
SL	Measuring sports equipment trajectory	Type 1	None
SL	Measuring golf swing trajectory	Type 1	None
SL	Measuring golf swing trajectory	Type 1	LSM-Based calibration
SL	Measuring bowling spin dynamics	Type 1	None
SL	Measuring bowler's hand dynamics	Type 1	None
SL	Measuring kinematics of baseball/softball	Type 1	Ball's kinematics, VICON

Chapter 4

Attitude Representation

According to the basic scheme of a fully functional measurement system, data representation is one of the vital parts of every measurement device. Generally, based on the nature of the acquired signals, there are various ways for data representation which are highly co-related to the application of the system. Since, our aim in this thesis is to graphically reconstruct human physical activities using acquired inertial measurements, understanding the concepts related to the 3D attitude representation of a rigid body is necessary for graphical representation. This chapter mainly focuses on the theory of attitude representation by angular rotation, velocity, and acceleration of a rigid body in 3D space based on measurement readings of inertial measurement units.

The aim of the chapter is to dive into the most used mathematical methods of 3D attitude representation when dealing with rotation values in order to explain the working principle and mention the main advantages, and disadvantages of each method. It is also worth mentioning that the literature regarding this subject is extensive, therefore, inconsistent in defining the reference frames and parametrization of the mathematical expressions. Hence, in order to avoid inconsistency, before going deep into the concept, the first section 4.1 is dedicated to definition of the parametrization and the reference frames followed by section

4.2 explaining the most used methods of attitude representation of a rigid body in 3D space.

4.1 Definitions and parametrizations

The process of attitude representation has gone through various parametrizations due to practical applications. These methods are preferred over each other based on demands of each application such as: ease of implementation, formulaic simplicity, and physical as well as mathematical complications involved in each [23]. Due to the extensive availability of online literature regarding mathematical approaches regarding this concept, the work of Diebel et al. [24] which is a consistent, sufficient, and well defined content has been chosen as the main reference material of this chapter.

4.1.1 Coordinate systems

Movement is a noun describing an act of move which denotes as relative displacement of an object with respect to a real, or hypothetical fixed reference. Therefore, when dealing with any sort of attribute which describes movement (rotation angles in this particular case), defining a reference frame for describing the action is evidently necessary.

The literature for defining coordinate systems in attitude representation, defines ***world coordinate system***, and ***body-fixed coordinate system*** as reference frames. The former is fixed in inertial space (denoted as X_w) while the latter is rigidly attached to the object whose attitude is described (denoted as X_b). Accordingly in order to distinguish points that are defined in world, and body-fixed coordinate systems, the parameters are denoted as X and X' respectively. For example, the value of X_w which is a point both located and described in the world coordinate system is considered zero (Similarly $X'_b = 0$).

4.1.2 Rotation and transformation matrix

In mathematical expressions, it is very common to represent rotation using a rotation matrix being applied to a vector which results rotating the vector direction while maintaining the length of the vector constant. The special orthogonal group of all rotation matrices is denoted by $SO(3)$ $(3 \times 3 dimension)$, therefore if $(R \in SO(3))$, then:

$$det\ R = \pm 1 \quad and \quad R^{-1} = R^T \tag{4.1}$$

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.2}$$

wehere $detR = 1$ is denoted as a Proper rotation, and $detR = -1$ is called an Improper rotation.

Since two coordinate systems has been introduced formerly, it is convenient to define a form of transformation operation in order to encode the attitude represented in world coordinate to fixed-body coordinate, and vice versa. Considering Z and Z' (both $\in R^3$) as world coordinates, and fixed-body coordinates respectively, based on Eq. 4.1 it is possible to write:

$$z' = Rz \tag{4.3}$$

$$z = R^T z' \tag{4.4}$$

Now, if the coordinate transformation is required to a point not related to the origin of the target coordinate system, it is possible to perform the transformation

subtracting the offset of the target origin before starting the operation:

$$x' = R(x - x_b) = Rx + x'_w = -Rx_b \tag{4.5}$$

$$x = R^T(x' - x'_w) = R^T x' + x_b = -R^T x'_w \tag{4.6}$$

It is also possible to write (4.6), and (4.5) in form of matrix equations in which it is denoted as the transformation matrix:

$$\begin{bmatrix} x' \\ 1 \end{bmatrix} = \begin{bmatrix} R & -Rx_b \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} R & x'_w \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{4.7}$$

$$\begin{bmatrix} x \\ 1 \end{bmatrix} = \begin{bmatrix} R^T & x_b \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x' \\ 1 \end{bmatrix} = \begin{bmatrix} R & -R^T x'_w \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x' \\ 1 \end{bmatrix} \tag{4.8}$$

4.1.3 Coordinate rotation

For the final part of the definiton and parametrization, the matrix representation format of the coordinate rotations is presented. This form of rotation is a single rotation operation around a single coordinate axis which is either x, or y, or z (enumerated as 1, 2, and 3 respectively). The rotation matrix with parametric angle α along each axis is presented as:

$$R_1(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{4.9}$$

$$R_2(\alpha) = \begin{bmatrix} cos(\alpha) & 0 & -sin(\alpha) \\ 0 & 1 & 0 \\ sin(\alpha) & 0 & cos(\alpha) \end{bmatrix} \tag{4.10}$$

$$R_3(\alpha) = \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

4.2 Methods of representation

The most common way of representing attitude of a rigid body is using triple Euler angles (Formerly mentioned as Yaw/Pitch/Roll angles in navigation systems). Euler angles which were intruded by Leonhard Euler in 18^{th} century, are an easy way to demonstrate the orientation of a rigid body in 3-dimensional space. Although Euler angles are widely used because of their easy-to-use, and easy-to-understand mathematical expression, using them introduces a major drawback referred to as singularities in certain important rotation functions, also known as, ***Gimbal Lock***. The mentioned drawback along with the need of a more accurate method when applications needed integration of incremental attitude changes over time, led researchers to parametrize attitude representation using Unit Quaternions. Quaternions, which were introduced by William Rowan Hamilton in 19^{th} century, are a form of a four-component complex numerical system which are mostly used in pure mathematics, but indeed, have practical uses in applied science such as quantum mechanics, and navigation systems, as well as attitude representation in general.

It is also worth mentioning, other methods of attitude representation such as *Rotation vectors* are available, even though they are out of the scope of this project. Therefore a brief description of Euler angles, Quaternions, and important transformation functions for each method are provided in this section.

4.2.1 Euler angles

As previously mentioned, Euler angles were initially introduced by Leonhard Euler. Euler angles represent rotation by performing sequential rotation operation with respect to a particular sequence and the rotation angle value. By representing the 3D space with three perpendicular axes denoted as i, j, k, it is possible to rotate any rigid body object by angles ϕ, θ, ψ respective to the denoted 3D

axes. Therefore, it is possible to define Euler angles in form of a rotation vector:

$$u := \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \tag{4.12}$$

In which R_{ijk} is the rotation function which applies each angle value to the corresponding axis:

$$R_{ijk} := R_i(\phi)R_j(\theta)R_k(\psi) \tag{4.13}$$

The coordinate transformation functions previously mentioned in Eq. 4.3 are valid as well:

$$z' = R_{ijk}(u)z \tag{4.14}$$

$$z = R_{ijk}(u)^T z' \tag{4.15}$$

Euler angle rates matrix

Applying a derivation operation with respect to time on the Euler angle vector provides the *Euler angle rates* ($\dot{u}$) which forms the relationship between angular velocity by *Euler angle rates matrix* ($E_{ijk}(u)$). In essence, multiplying Euler angle rates matrix to the vector of Euler angles rates results the angular velocity of an object in the world coordinates. Needless to mention the angular acceleration in this case is obtained by taking the time derivative of the angular velocity. The operation is indicated as follows:

$$E_{ijk}(\phi, \theta, \psi) := [R_k(\psi)^T R_j(\theta)^T \hat{e}_i, R_k(\psi)^T \hat{e}_j, \hat{e}_k] \tag{4.16}$$

$$\omega = E_{ijk}(u)\dot{u} \tag{4.17}$$

Considering the conjugate of Euler angle rates matrix, it is also possible to obtain angular velocity in the body-fixed coordinate system as follows:

$$E'_{ijk}(\phi,\theta,\psi) := [\hat{e}_i, R_i(\phi)\hat{e}_j, R_i(\phi)R_j(\theta)\hat{e}_k] \tag{4.18}$$

$$\omega' = E'_{ijk}(u)\dot{u} \tag{4.19}$$

Alternatively, by obtaining angular velocity in one coordinate system, it is possible to convert the value to the other coordinate system by applying coordinate transformation (see Eq. 4.5, and 4.6) as follows:

$$\omega' = R_{ijk}(u)\omega \tag{4.20}$$

$$\omega = R_{ijk}(u)^T\omega' \tag{4.21}$$

Finally, using Euler angle rates matrix, and its conjugate, it is possible to obtain the Euler angles vector, and its transpose by applying below operation:

$$R_{ijk}(u) = E'_{ijk}(u)[E_{ijk}(u)]^{-1} \tag{4.22}$$

$$R_{ijk}(u)^T = E_{ijk}(u)[E'_{ijk}(u)]^{-1} \tag{4.23}$$

Valid sequences

As already mentioned, performing rotations based on Euler angles is a sequential process which means the rotations are applied to one axis at a time. For this reason, it is necessary to indicate the rotation priority of the process as a property of the operation. If we consider a enumerating coordinate axes as $x = 1, y = 2, z = 3$, among all $3 \times 3 \times 3 = 27$ sequences, only 12 of them satisfy the constraint of not having two consecutively equal numbers (2 rotations in a row for 1 axis). Among all the 12 valid sequences, the (1,2,3), and (3,1,3) sequences are mostly used choices in various applications. Table 4.1 describes the sequences along with their applications and their singularities.

Table 4.1: Rotation Sequences of Euler angles

Sequence	Transition	Usage	Singularities
$(3,1,3)$	$\psi : (x \to x')$ $\theta : (y \to y')$ $\phi : (z \to z')$	Aerospace engineering and computer graphics	$\theta = \frac{\pi}{2} + n\pi$
$(1,2,3)$	$-\psi : (-y \to z')$ $\frac{\pi}{2} - \theta : (-x \to -z)$ $\phi : (-z \to -y')$	Gyroscopic spinning motion of a rigid body	$\theta = n\pi$

It is also worth mentioning, useful transfer functions such as $R(u) \to E(u)$, and $R(u) \to Q$ are slightly different for each sequence. Therefore, in the following sections only the transfer function of sequence (1,2,3) is provided since it is the desired sequence of this project.

Transfer function: $R(u) \to q_{w,x,y,z}$ **[Seq(1,2,3)]**

The following transfer function maps Euler angles vector to the corresponding unit Quaternion, where $Cos(x)$ is denoted as C_x, and $Sin(x)$ is denoted as S_x:

$$Q(\phi,\theta,\psi) = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} := \begin{bmatrix} C_{\phi/2}C_{\theta/2}C_{\psi/2} + S_{\phi/2}S_{\theta/2}S_{\psi/2} \\ -C_{\phi/2}C_{\theta/2}C_{\psi/2} + S_{\theta/2}S_{\psi/2}S_{\phi/2} \\ C_{\phi/2}C_{\psi/2}C_{\theta/2} + S_{\phi/2}S_{\theta/2}S_{\psi/2} \\ C_{\phi/2}C_{\theta/2}C_{\psi/2} - S_{\phi/2}S_{\psi/2}S_{\theta/2} \end{bmatrix} \tag{4.24}$$

Transfer function: $R(u) \to E(u)$ **[Seq(1,2,3)]**

The following transfer function can compute Euler angle rates matrix, its inverse, and its conjugate which was mentioned in Eq. 4.17 to 4.23 based on Euler angles, where $Cos(x)$ is denoted as C_x, and $Sin(x)$ is denoted as S_x:

$$E_{123}(\phi,\theta,\psi) = \begin{bmatrix} C_\theta C_\psi & -S_\psi & 0 \\ C_\theta S_\psi & C_\psi & 0 \\ -S_\theta & 0 & 1 \end{bmatrix} \tag{4.25}$$

$$E'_{123}(\phi,\theta,\psi) = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \tag{4.26}$$

$$[E_{123}(\phi,\theta,\psi)]^T = \frac{1}{C_\theta} \begin{bmatrix} C_\psi & S_\psi & 0 \\ -C_\theta S_\psi & C_\theta C_\psi & 0 \\ C_\psi S_\theta & S_\psi S_\theta & C_\theta \end{bmatrix} \tag{4.27}$$

$$[E'_{123}(\phi,\theta,\psi)]^T = \frac{1}{C_\theta} \begin{bmatrix} C_\theta & S_\phi S_\theta & C_\phi S_\theta \\ 0 & C_\phi C_\theta & -S_\phi C_\theta \\ 0 & S_\phi & C_\phi \end{bmatrix} \tag{4.28}$$

4.2.2 Quaternions

Quaternions is the name of a four component extended complex numbering system which is divided into real and imaginary part, and it is used as a method of attitude representation. Concept of Quaternion which was firstly introduced by Olinde Rodriguez (1840), then later independently discovered and further explained by William Rowan Hamilton (1843), is a proper method to overcome the problems arising from the singularities of Euler angles which would eventually lead the system into the state of Gimbal lock and the loss of at least one degree of freedom. Quaternions are more efficient to implement on embedded system platforms since triangular functions require more processing steps than complex numerical multiplications. Nevertheless, from the practical point of view, the main disadvantages of Quaternions are: the lack of intuitive physical meaning of a Quaternion vector, and the unity norm constraint, which requires a vector

of Quaternion to have a unity norm in order to be considered as valid a rotation vector. Unity norm constraint is problematic specially when optimization algorithms are applied to the raw measurement values.

Quaternions are usually represented as vectors in which the first element ($q_0 := w$) is the imaginary part, while the following three parameters are the real values ($q_{1:3} := (x, y, z)$) as indicated below:

$$q_{w,x,y,z} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T = \begin{bmatrix} q_0 \\ q_{1:3} \end{bmatrix} \tag{4.29}$$

In some mathematical operations, the inverse of Quaternion matrix (q^{-1}) is needed which can be calculated using Quaternion adjoint ($\overline{q}$), and norm ($\|q\|$) as indicated in the following equation:

$$\overline{q} = \begin{bmatrix} q_0 \\ -q_{1:3} \end{bmatrix} \tag{4.30}$$

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{4.31}$$

$$q^{-1} = \frac{\overline{q}}{\|q\|} \tag{4.32}$$

Quaternion rates matrix

The relationship between unit Quaternions, angular velocity, and angular acceleration is connected through Quaternion rates matrix similar to the Euler angles. In case of Quaternions, the angular velocity in the world and body-fixed coordinate system (ω_q, and ω'_q respectively) is obtained by multiplying time derivative of the unit Quaternion ($\dot{q}$) to the Quaternion rates matrix ($W(q)$, and $W'(q)$

respectively) as indicated below:

$$W(q) := \begin{bmatrix} -q_1 & q_0 & -q_3 & q_2 \\ -q_2 & q_3 & q_0 & -q_1 \\ -q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \tag{4.33}$$

$$W'(q) := \begin{bmatrix} -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \tag{4.34}$$

$$\omega_q := 2W(q)\dot{q} \tag{4.35}$$

$$\omega'_q := 2W'(q)\dot{q} \tag{4.36}$$

It is also possible to do the reverse operation of above, and obtain first time derivative of the unit Quaterinon by taking into account Quaternion rates matrix, and angular velocity (both world, and body-fixed coordinates are applicable) simply by performing the following expression.

$$\dot{q} = \frac{1}{2}W(q)^T\omega = \frac{1}{2}W'(q)^T\omega' \tag{4.37}$$

Following the above expressions, the angular acceleration in the world and body-fixed coordinate system ($Dot\omega_q$, and $Dot\omega'_q$ respectively) can be obtained by taking into account multiplication of the Quaternion rates matrix, and the second time derivative of the unit Quaternions ($\ddot{q}$) as indicated below:

$$\dot{\omega}_q := 2W(q)\ddot{q} \tag{4.38}$$

$$\dot{\omega}'_q := 2W'(q)\ddot{q} \tag{4.39}$$

Similar to the case of angular velocity, it is possible to do the inverse operation and obtain the second time derivative of the unit Quaternion by taking into account the Quaternion rates matrix and the angular acceleration (both body-fixed and

world coordinates are applicable) as indicated below:

$$\ddot{q} = \frac{1}{2}W(q)^T\dot{\omega} = \frac{1}{2}W'(q)^T\dot{\omega}' \tag{4.40}$$

Transfer function: $q_{w,x,y,z} \rightarrow R(u)$ **[Seq(1,2,3)]**

The transfer function that maps Quaternions to their corresponding Euler angles vector in sequence (1,2,3) is indicated below where $atan2(x)$ stands for 2-parametric inverse tangent function of variable x, and $asin(x)$ is the inverse sine function of variable x.

$$R(u)_{1,2,3} = \begin{bmatrix} atan2(2q_2q_3 + 2q_0q_1, q_3^2 - q_2^2 - q_1^2 + q_0^2) \\ -asin(2q_1q_3 - 2q_0q_2) \\ atan2(2q_1q_2 + 2q_0q_3, q_1^2 + q_0^2 - q_3^2 - q_2^2) \end{bmatrix} \tag{4.41}$$

4.3 A careful observation

A careful observation reveals that the case of the rotational movements is similar to the case of the linear movements, in which, the velocity and the acceleration values are obtained, respectively, from the first and the second time derivative of the linear displacement values. The only difference is that the term *linear* is replaced by the term *angular* for the case of rotational movements.

Until now, the expressions for obtaining angular velocity, and angular acceleration from rotation values have been described, while on the other hand, the actual goal is to obtain attitude of a rigid-body object based on its angular velocity, and angular acceleration. Therefore, it is not possible to directly measure the rotation values for **attitude representation**. Hence, it is necessary to obtain an estimate of the attitude (**attitude estimation**) by perform integration process on angular velocity, and angular acceleration values with respect to time.

The mentioned process effectively will give rise to a common issue addressed

as **accumulated error**, which is the accumulation of the errors of the angular velocity, and angular acceleration measurements throughout the integration process. To overcome the mentioned issue, it is necessary to use methods such as *Kalman*, and *Complementary* filtering. Most of the commercially available inertial measurement units perform these methods on the raw measurement data in order to reduce the accumulated errors

Chapter 5

Implementation Scenario

Up to now, the general working principle of the inertial measurement units, and the mathematical expressions regarding attitude representation have been explained. The main focus of this thesis is the implementation and the development of a sensor-based motion capturing device. Therefore, this section is dedicated to the step-by step explanation of a consistent pathway of implementing a MoCap solution by taking into account necessary considerations.

Implementing wearable devices, in general, requires various design considerations to be taken into account. The design criterion varies due to the application, Therefore, in a more specific context,this chapter covers the implementation of a full-body wearable cable-free motion capturing device. It is evidently clear that commercially available wearable products require more effort in terms of product design. While, on the other hand, devices with the purpose of being used during lab researches require more technical development. In this sense, this thesis mainly covers the technical development issues rather than the product design obstacles.

In section 5.1 the used terminology through the rest of this section is explained, and in section 5.2 the important design factors for a consistent implementation of a wearable MoCap solution are defined.

5.1 Terminology

Considering a fully functional wearable motion capturing product, the factors to be considered are accurately indicated by Marin et al. [8] in which by analysing commercially available products, as well as, research purpose products, they described a broad technical, and product design terminology to be used during the development phase of wearable motion capturing devices.

The terminology describes *device* as motion capturing elements to be placed on human body, and *data processing points (DPP)* as the stationary device which receives the data, evaluates them, does the necessary processing steps on them, and represents them in an appropriate way. Since the Bluetooth Low Energy has been used as the communication protocol of this project, we may address *device*, and *DPP* as *Peripheral* and *Central* respectively, which are the technical terms used in the specifications of the communication protocol.

Furthermore, the literature regarding methods of motion capturing is divided into optical methods and sensor-based, in which the sensor-based methods are in forms of wearable devices. Therefore, it is necessary to indicate that not all the MoCap devices are in forms of wearables, and also not all the wearable devices containing inertial measurement units are for the application of motion capturing since some devices use IMUs for gesture, or activity recognition (e.g fitness trackers). In this sense, to be specific, this project introduces a *full-body motion capturing wearable device.*

5.2 Design factors

5.2.1 Contextual factors

Contextual factors describe the characteristics of the solution which eventually will be introduced as design requirements. These factors include the study of the user, as well as, the study of the environment where the solution is being applied.

Based on the terminology, this project, which contains multiple devices and a DPP, is intended to be used by two distinct types of users. The distinction between the users defines user roles using the contextual factors as follows: *the actor user*, similar to a test subject, who by wearing the device, performs the desired activities to be measured, while on the other hand, *the professional user* who is interested in studying, and analyzing the performed activities by obtaining the measurement data. It is necessary to mention that the distinction between user roles does not necessarily mean the requirement of two individuals to operate simultaneously with the system. Hence, one individual can perform two user roles as well.

Furthermore, *environment* of the solution describes the criteria to be taken into consideration regarding the design. As previously mentioned, a solution which is intended to be used as a consumer end-point product requires more effort regarding product design comparing to a solution which is intended to be used during lab researches. It is also worth mentioning that throughout the pathway from a concept to production, the required economical resources to address design obstacles increase, while the the amount of available tools decrease drastically as pathway goes on. Therefore, the considered environment at the current stage of this project is to be used during lab researches for study purposes where most of the resources have been considered for technical development rather than product design. Table 5.1 summarizes the contextual factors of this solution.

Table 5.1: Contextual factors of the solution

Environment	Laboratory
Actor user	Test subject
Professional user	Researcher
Device	Sensor tags (Peripherals)
DPP	Central receiver + PC

5.2.2 Interaction factors and scheme

There are generally two schools of thought for describing interaction factors. The first is the interaction between the system, and the user which is called user interaction factor by means of user interface. The second is the interaction between the different parts of the system within the application of the solution. The former divides the user role schematics into unidirectional scheme, and bidirectional scheme.

User interaction scheme

Based on the formerly mentioned user type definitions, the scenario for the user interaction scheme of an actor user is unidirectional from the device to the actor user by means of feedback. An actor user can be informed about various parameters for example with visual feedback using light emitting diodes (LEDs) the actor user can understand the device is on, and calibrated for application. On the other hand, for the case of a professional user, the interaction scheme is bidirectional between the user and the system. The bidirectional scheme provides the possibility of changing measurement parameters, or working modes of the system by interacting with the DPP. It is also evident that the actor user, and the professional user also have a bidirectional interaction between themselves which the experiment procedure. In this project, the actor user is notified by the blinking green LED on the sensor tags when the device is ready to be used, and the professional user can monitor and modify representation parameters by interacting with the installed acquisition software on a personal computer. Figure 5.1 indicates the schematic of the interaction between the user and the system.

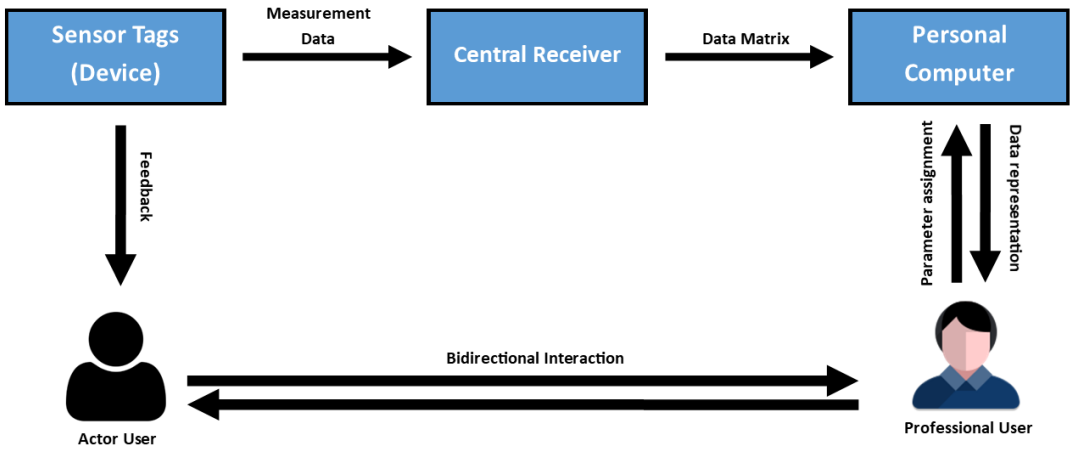


Figure 5.1: User interaction schematic

System interaction scheme

As previously described, this system consist of multiple devices and a DPP. The devices are equivalent to wearable sensors while the DPP is equivalent to combination of a central receiver, and a personal computer. In a more broad context, the combination of the wearable sensors, and the central receiver device could be called the *Acquisition group*, and the personal computer could be called the *Representation group*.

Sensor tags (devices) consist of a 3-axis accelerometer, a 3-axis gyroscope, and a communication module which enables them to establish a unidirectional connection to the central receiver using BLE. The role of the central receiver in this scenario is forming a data matrix after obtaining the data from all the sensor tags. Finally, the data matrix is communicated from the central receiver to a personal computer using a unidirectional serial communication. Figure 5.2 illustrates the architecture of the sensor tags, and Figure 5.3 illustrates the basic system interaction schematic of the solution.

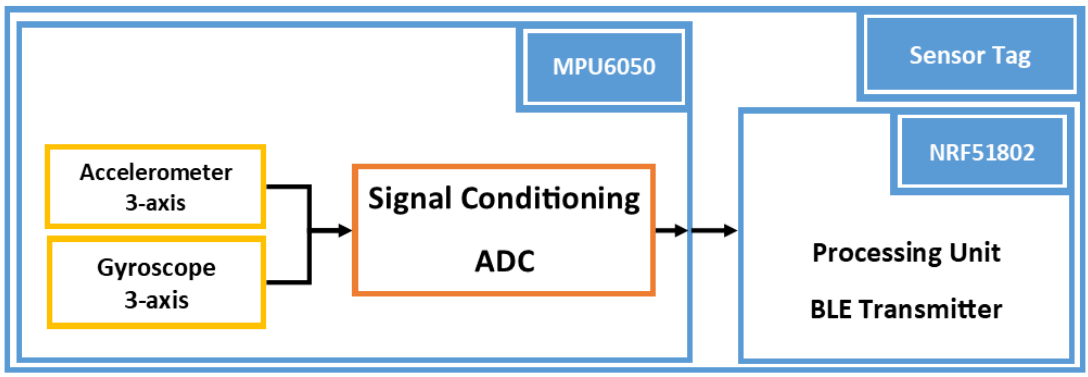


Figure 5.2: Sensor tag architecture

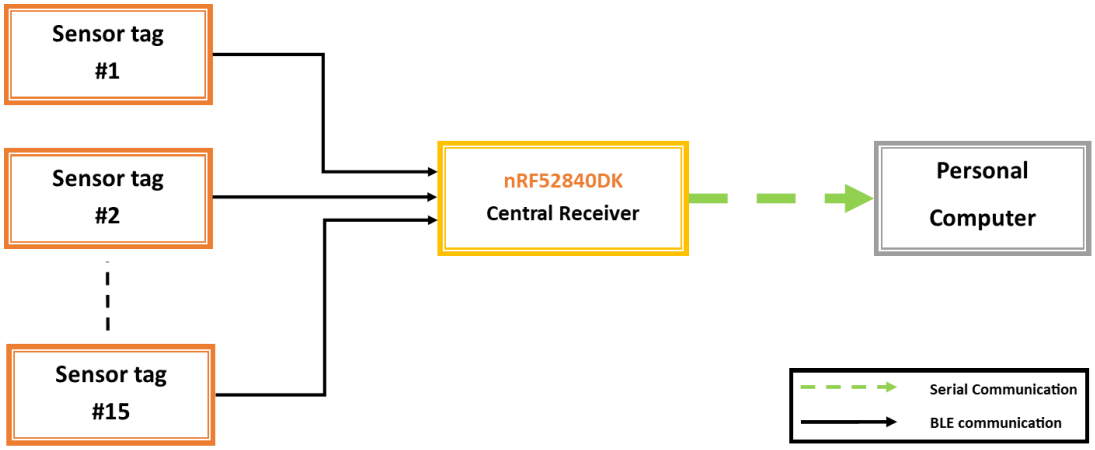


Figure 5.3: System interaction schematic

Modified scheme of the sensor tags

Regarding the technological factors of a consistent design criteria of wearable MoCaps, it is required to equip the sensing tags (devices) with rechargeable batteries. Since the commercial sensor tags which were chosen for this project lacked the possibility to operate with rechargeable batteries, a new modified scheme of the for the sensor tags has been designed. The scheme also considered a stand alone charging circuit for every sensor tag which provides the possibility to charge the sensor tags without the need of removing the batteries. The modified scheme of the sensor tag is illustrated in Figure 5.4. Detailed information regarding the

hardware design and specifications are provided in chapter 6.

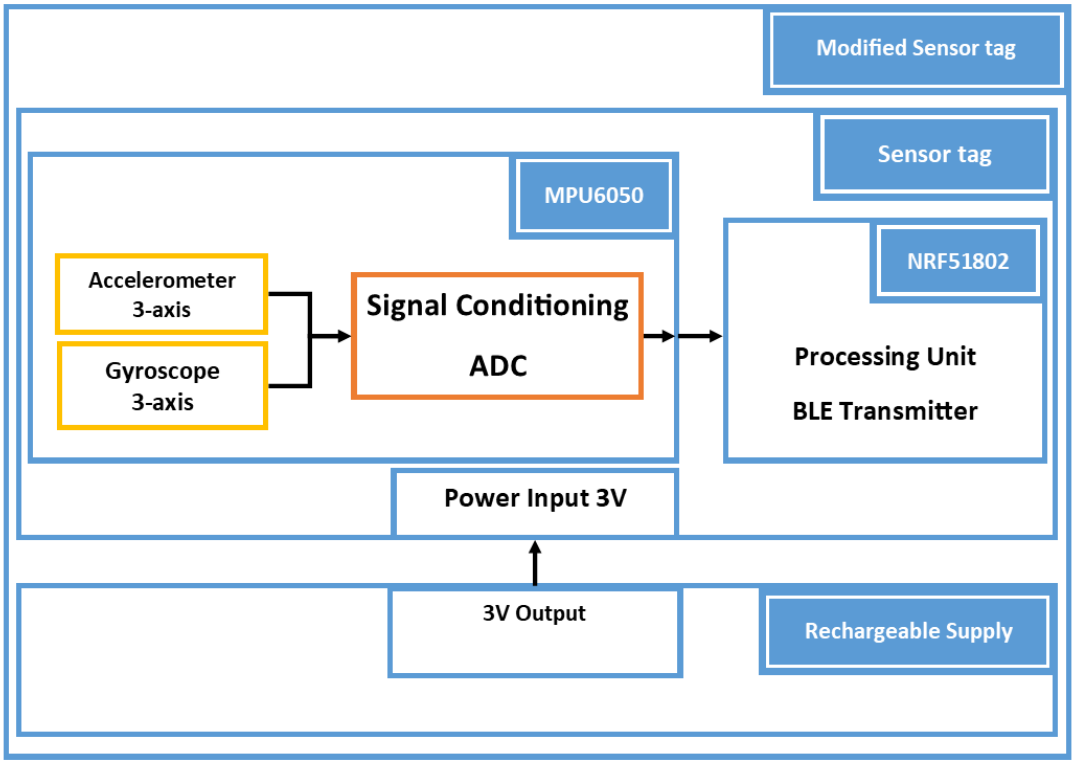


Figure 5.4: Modified sensor tag architecture

5.2.3 Sensor placement

Due to the working principle of IMUs, it is evidently clear that the acceleration and angular velocity values of different body parts during an activity are different. Therefore, in order to reconstruct human activities using inertial measurements, it is necessary to consider the sensor placement on the human body in a way to avoid any friction, and displacement of the sensors while performing various activities as well as choosing areas with the least movement in order to minimize measurement errors. Considering the mentioned reasons, the places right above the joints, and flat areas are preferably used since they do not experience extreme deviation during movements, and the sensor displacement on these areas are minimum.

Also, regarding the design criteria of wearable MoCap solutions, sensor location on the human body should be chosen in a way to avoid user movements while performing desired activities. According to Marin et al. [8], it is suggested to use outermost parts of the body in order to avoid interference during the movements.

Finally, although higher number of sensors may result the reconstruction process to be more accurate, it may also increase the cost of the system as well as limiting the actor user movements.

Considering all of the above, this solution uses 15 wearable sensors for full-body motion capturing which are attached to the human body as illustrated in Figure 5.5, where the black dots refer to sensors which are attached in the front side of the body, while the orange dots refer to sensors which are attached in the back side. The measured parameters by each sensor are indicated in Table 5.2.

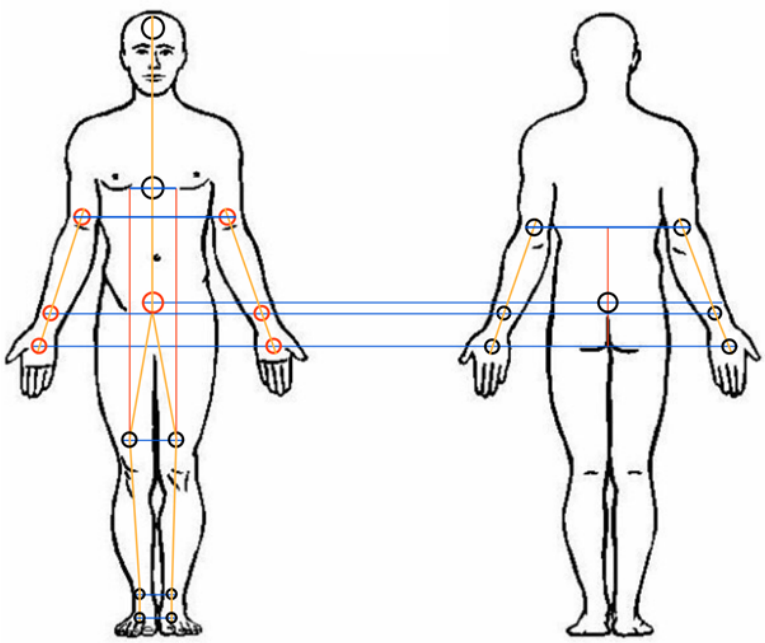


Figure 5.5: Sensor tags attachment to the human body

Table 5.2: Sensor names and description

#	Sensor	Description
1	Forehead	Measures the rotation and side-bend of forehead with respect to origin
2	Chest	Measures anterior/posterior tilt and Literal tilt to the left/ right.
3	Left Arm	Measures Abduction/Adduction, flexion/extension of left arm.
4	Right Arm	Measures Abduction/Adduction, flexion/extension of left arm.
5	Left wrist	Measures Abduction/Adduction, flexion/extension of the left wrist
6	Right Wrist	Measures Abduction/Adduction, flexion/extension of the right wrist
7	Left Hand	Measures Abduction/Adduction, flexion/extension of the left hand
8	Right Hand	Measures Abduction/Adduction, flexion/extension of the right hand
9	Sacral (Back)	Measures Rotation and side-bend of back with respect to origin.
10	Left Knee	Measures Flexion/Extension
11	Right Knee	Measures Flexion/Extension
12	Left Ankle	Measures Plantar-Flextion/Dorsi-Flexion and Supination/Pronation
13	Right Ankle	Measures Plantar-Flextion/Dorsi-Flexion and Supination/Pronation
14	Left foot	Measures Rotation and side-bend
15	Right foot	Measures Rotation and side-bend

5.2.4 Body attachment factor

One of the vital parts of the design process when dealing with wearable devices in general is defining a parameter which is referred to as the *body attachment factor* or the *body attachment method* in the literature [8]. The body attachment factor describes the method of attaching the wearable device to the human body, and it is strictly related to the application of the solution. To be more specific, regarding implementation of a wearable device which is intended to be used for measurement applications, the body attachment factor directly affects the accuracy, and reproductibility of the solution, as well as the convenience of usage for the actor user.

Although there are various methods to address the body attachment factor in different applications, wearable MoCap solutions typically use methods such as: fixed fabric support, fixed adhesive support, and semi-rigid fixed support. Among the mentioned methods, the fixed fabric method has more flexibility for different body shapes as well as more economical feasibility regarding implementation. In this method the sensor is placed inside a rigid plastic station, or a placement pocket which is attached to a fabric elastic band and the band is used to fix the sensor position on the human body. Due to elasticity of the fabric band and by using buckle (or similar) structure, it is possible to adjust the band circumference for different body shapes.

In this project, the fixed fabric method is used by providing fabric elastic bands with different circumferences according to the body parts which the sensor tags are intended to be fixed. The circumference of each elastic band is adjustable by using clips, and a pocket with the capacity of containing a rigid object with dimension $3cm \times 3cm \times 1cm$ has been provided to contain the sensor tags (the pocket size is $5cm \times 5cm$).

it is also worth mentioning that the body attachment method is tightly related to the application of the wearable device, as well as the target subjects. Therefore,

the implemented body attachment method of this wearable solution is specifically designed for the test subject of this project. Table 5.3 represents the size of each body part of the subject corresponding to the sensor location, and Figure 5.6 illustrates the implemented body attachment method for sensor number 5 (Left wrist).

Table 5.3: Sizes of the test subject's body parts

Body part	**# Sensor**	**Circumference (CM)**	**Number needed**
Wrist	5 and 6	16	2
Arm	3 and 4	25	2
Hand	7 and 8	18	2
Chest	2	85	1
Forehead	1	55	1
Sacral (Back)	9	82	1
Knee	10 and 11	36	2
Foot	14 and 15	20	2
Ankle	12 and 13	23	2

(a) solution with a non-rechargeable sensor tag inside

(b) solution with a rechargeable sensor tag inside (see section 6.1.2)

Figure 5.6: Left wrist implemented body attachment solution

Chapter 6

Hardware Design

After explaining the working principle of IMUs, the necessary theoretical concepts for attitude representation, and the implementation scenario, it is possible to provide a detailed explanation of the hardware implementation procedure for our wearable motion capturing solution.

In this chapter, we will introduce the hardware components which are used to implement our solution in order to clarify the main specifications and features of each component. The embedded architecture of the sensor tags, as well as the central receiver, are explained, and eventually, the modified scheme of the sensor tags in order to be compatible with rechargeable batteries is introduced.

6.1 Hardware specifications and features

As previously indicated, the architecture of this solution requires three main components: the sensor tags (Device), the central receiver (DPP), and the personal computer (DPP). By using all of these components, it is possible to acquire angular velocity and angular acceleration of each sensor tag in order to perform the attitude representation procedure using the acquired data. In this section, the features and specifications of the sensor tags, and the central receiver are

provided respectively.

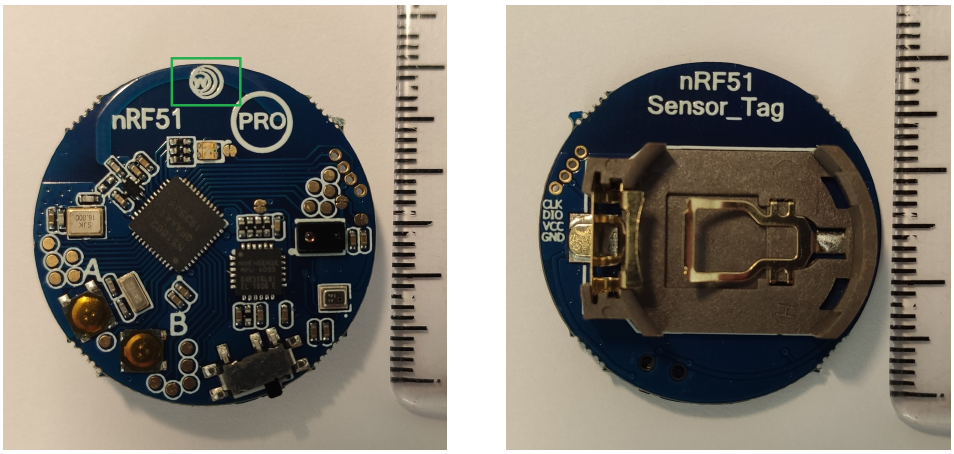
6.1.1 Sensor tags

Data acquisition in this solution is done using a network of 15 sensor tags. The tags used in this solution are commercially available modules, also known as, nRFtag (or Sensor_tag). Each tag consists of MPU6050 inertial measurement unit, BMP180 pressure sensor, AP3216 ambient light sensor, power button, battery holder, and nRF51802 programmable processing unit.

From each sensor tag, the inertial measurement unit (MPU6050) is used as the sensor, and the nRF51802 system on a chip (SoC) is used as the processing unit and the BLE communication module. The sensor tags general features and specifications are described in Table 6.1.

Table 6.1: Sensor tags features and specifications

Product Name	**nRFtag (Sensor_tag)**
Application	Wearable devices
Supply voltage	3V Coin cell 2032 package
Embedded modules	nRF51802 MPU6050 BMP280 AP3216
Operating temperature	$-25 \sim +75$ ºC (Recommended 25 ºC)
Size	Circular $d = 30mm$)
Additional features	
nRF51802 on this module is programmable using a connection from 4 vias CLK, DIO, VCC, GND to nRF52840DK (the central receiver) pins SWDCLK, SWDIO, VTG, GND DETECT respectively.	



(a) Top view (b) Bottom view

Figure 6.1: Sensor tag module

MPU6050 which is used as the inertial measurement unit of this solution is a combination of a 3-axis accelerometer and a 3-axis gyroscope manufactured by TDK InvenSense. Table 6.2 highlights the main features and specifications of this IMU, extracted from the manufacturer datasheet [25].

Table 6.2: MPU6050 features and specifications

Product Name	**MPU6050**
Application	Inertial Measurement Unit (Sensor)
MEMS	3-axis accelerometer 3-axis gyroscope
Supply voltage	$2.375 \sim 3.46\ V$
Operating temperature	$-40 \sim +85$ ^{o}C (Recommended 35 ^{o}C)
Gyroscope specifications	
Gyroscope full-scale range	± 250, ± 500, ± 1000, ± 2000 ^{o}/sec
ADC	Integrated 16-bit ADC

Table continued from previous page

Operating current	$3.6mA$
Standby current	$5\mu A$
Accelerometer specifications	
Accelerometer full-scale range	$\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$
ADC	Integrated 16-bit ADC
Normal operating current	$500\mu A$
Additional features	

- Digital motion processing (DMP)
- Overall operating current using DMP: $3.9mA$

nRF51802 is the programmable processing unit of the sensor tags. This chipset is manufactured by Nordic Semiconductor™. By using this chipset, it is possible to organize the available peripherals of the sensor tags, acquire measurement data from MPU6050 through I2C protocol, perform measurement data calibration, and communicate measurement data to the central receiver using BLE. The main features and specifications of this processing unit are provided in Table 6.3 which are obtained from the manufacturer datasheet [26].

Table 6.3: nRF51802 features and specifications

Product Name	**nRF51802**
Application	Processing unit, Wireless communication
Supply voltage	01.75 $\sim$ 3.6 V depending on the working mode
Operating temperature	$-25 \sim 75$ ^{o}C Recommended (25^{o}C)
Processing unit electrical specifications	
Processor	ARM® Cortex^{TM}-M0 32bit
LF/HF CLK	16 $\sim$ 32.768MHz
Current consumption	Run from flash: 4.4mA Run from RAM: 2.4mA Startup: 400μA
Radio transceiver specifications	
Frequency	BLE 4.2: 2.4 $\sim$ 2.483GHz with 1MHz channel spacing
Current consumption	TX startup+run @ P_{out}=4dBm: 23mA RX startup+run @ 250kbps: 12.6mA
I2C Peripheral specifications	
SCL Clock frequency	Standard mode: 100kHz Fast mode: 400kHz
Current consumption	@100kbps: 380μA @400kbps: 400μA

6.1.2 Rechargeable wearable sensors

Considering the design criteria previously described for wearable MoCap solutions, having wearable devices is more convenient from the perspective of the actor user. For this matter, a modified design model for wearable devices is introduced. This modification is done to be able to replace the manufacturer-recommended non-rechargeable coin-cell power supply (CR2032 200mAh) with a rechargeable and compatible power supply. In order to proceed with this modi-

fication, there are two possible methods. Table 6.4 summarizes the features and specifications of each method for a more clear comparison.

The first method is to use Li-Al or Li-MnO_2 rechargeable coin-cell batteries with a package size of 2032[1] (such as ML2032). The advantage of this method is that this type of battery fits into the factory-provided battery holder of the sensor tags, and it provides a relatively constant 3V nominal output voltage. Therefore, no further modification is required by using this method. On the other hand, the disadvantages are that the nominal capacity of this type of battery is 65mAh which is considerably lower than the manufacturer recommended power supply, and throughout the charging period, the charging current must be limited to 2mA which results in a long charging period.

The second method is to use Lithium Polymer (Li-Po) rechargeable batteries. The advantage of this method is that this type of battery is commercially available in a variety of shapes and nominal capacities, and the battery can be fully charged safely using the constant current/constant voltage (CC/CV) method within one hour. On the other hand, the disadvantages of this method are that, the output voltage of this type of battery is 3.7V, and that output voltage also decreases throughout the discharging period. To overcome the former disadvantage, it is required to use an additional voltage regulator component at battery's output. On the other hand, the latter disadvantage requires additional considerations such that to avoid unnecessary circuit interruptions.

Considering the application purpose of this solution, the required up-time of each wearable device, and the general portability design criteria regarding wearable solutions, the second method was chosen for the implementation.

[1]Package 2032 represents a cylindrical object with 20 mm cross-sectional diameter, and 3.2 mm height

Table 6.4: Rechargeable methods comparison

Method	Factory Recommended	Method 1	Method 2
Name	CR2032	ML2032	Li-Po
Output Voltage (V)	3	3	3.7
Nominal Capacity (mAh)	200	65	150
Rechargeable	No	Yes	Yes
Weight (g)	2.8	3	4.5
Dimension (mm)	Coin cell $d = 20$	Coin cell $d = 20$	$25 \times 20 \times 2$
Additional Components Needed	No	No	Regulator Capacitors

To proceed with the second method (Using Li-Po batteries), it is necessary to take into account a few considerations:

- The output voltage of the Li-Po battery is 3.7V, while the input voltage of the wearable sensor tag is 3V. It is possible to solve this issue using a DC/DC 2.5 $\sim$ 6 V to 3V buck converter. The converter used in this project is based on the CE6208 CMOS regulator. It is important to note the dropout voltage since the output voltage of the Li-Po batteries tend to decrease during discharge period. The necessary specifications are highlighted in Table 6.5 from the manufacturer datasheet [27].

- Li-Po batteries require a charging circuit compatible with the CC/CV charging method. This issue is addressed using a TP4056 Li-Po battery charger module. This module keeps the charging current until the battery reaches the peak voltage of 4.2V, then, it will maintain the voltage while reducing the current. The charging current can be calculated using the parameter C (Charging rate indicated by the manufacturer) from the below

expression.

$$I_{Charging} = C \times Nominal\ Capacity \tag{6.1}$$

The considered charging current for this solution is 60mAh (C=0.4). Therefore, to maintain this charging current, it is necessary to modify the R_{prog} resistor on the TP4056 module circuit. According to the datasheet of the module, the required resistor can be calculated using Eq 6.2.

$$R_{prog}(\Omega) = \frac{1}{I_{bat}(A)} \times 1200 \tag{6.2}$$

Where I_{bat} defines the charging current of the battery. Therefore, resistor R_{prog} on the TP4056 charging module has been replaced by a 20 $k\Omega$ SMD resistor. Table 6.6 highlights the main features and specifications of the TP4056 module from the manufacturer datasheet[28].

- To reduce noise interference, and to avoid unnecessary circuit interruptions of the charging circuit caused by the inrush startup current consumption of the sensor tags, four capacitors ($100nF \sim 100\mu F$) were added to the final PCB. These capacitors are placed at the input and output of the DC/DC buck converter ($10\mu F$ and $100\mu F$), as well as the input of the battery charger ($10\mu F \mid\mid 100nF$). All the capacitors considered for this solution are provided with a 1206 SMD package.

Figure 5.4 illustrates the final scheme of the rechargeable wearable devices, Figure 6.3 shows the circuit schematic of the modified wearable devices, Figure 6.4 illustrates the designed PCB of the rechargeable sensor tags using Altium Designer software, Figure 6.5 shows the final implemented PCB of the wearable device, Figure 6.6 represents the soldered PCB of a wearable device using SMD components, and Figure 6.7 demonstrates the additional components added to the circuit.

Table 6.5: DC/DC buck converter features and specifications

Product Name	**DC/DC Buck Converter module**
Components	CE6208 CMOS regulator 2 x 100nF capacitors
Input Voltage (V)	$2.5 \sim 6$
Output Voltage (V)	3
Output Current (A)	1.0 (Guaranteed)
Dropout Voltage (V)	0.5 at 1.0A
Accuracy	$\pm 2\%$
Operating Temperature ºC	$-40 \sim +125$
Soldering Temperature ºC (10 Sec)	260
Dimension (mm)	11.7×7.7

Table 6.6: TP4056 features and specifications

Product Name	**TP4056 Charger Li-Po Battery Charger**
Input Voltage (V)	$4 \sim 8$V (Recommended 5V)
Charging voltage (V)	$4.2V \pm 1.5\%$
Charge termination threshold	$C/10$
Trickle charge threshold (V)	2.9V
Operating Temperature ºC	-40 $\sim$+85
Soldering Temperature ºC (10 Sec)	260
Current Input/Output **(@Ta=25 º,** $V_{cc} = 5V$**)**	$I_{cc} = 150$ $I_{BAT} = 500 \sim 1000mA$
Module Dimension (mm)	28×17

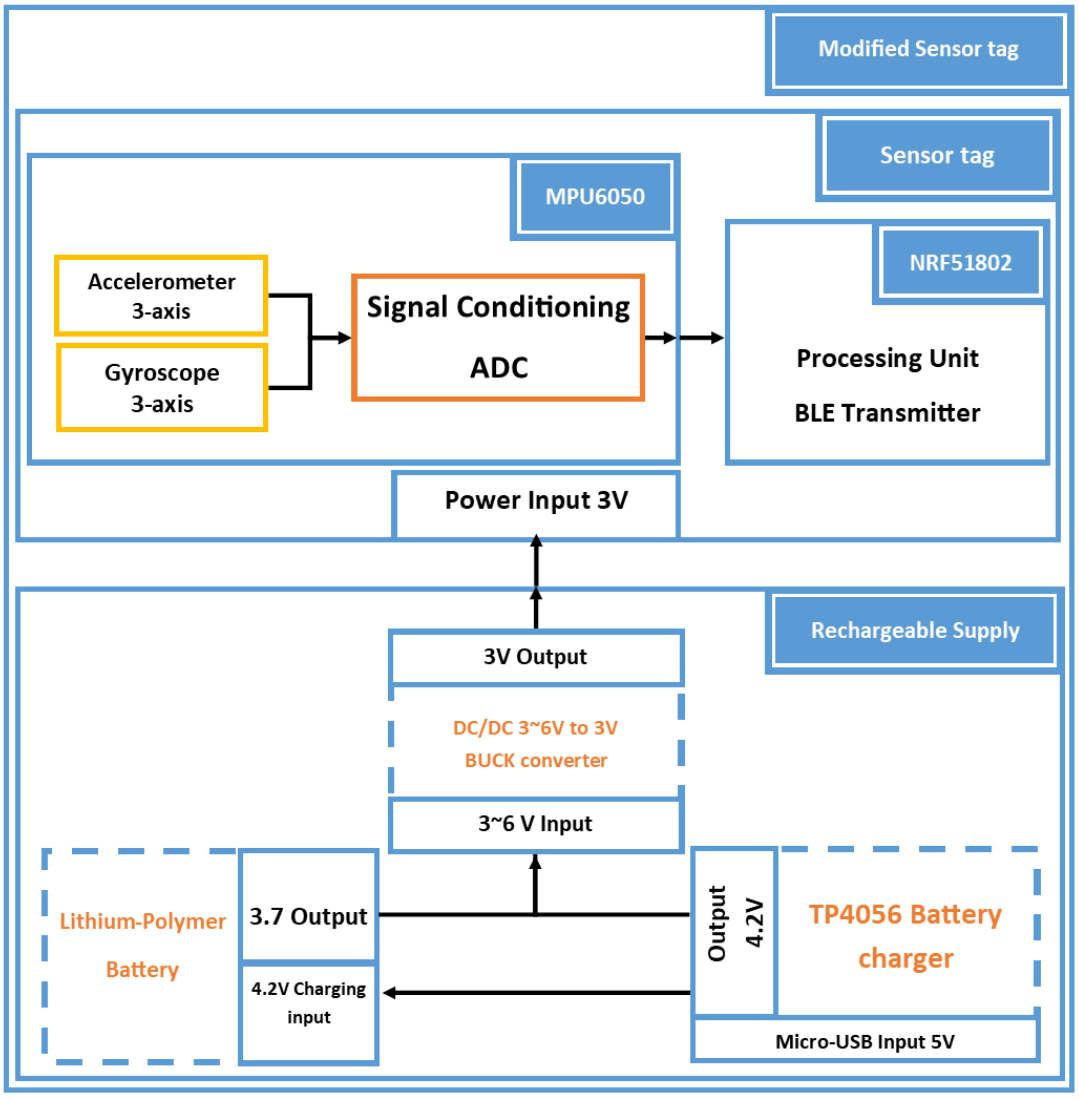


Figure 6.2: Final scheme of the sensor tags

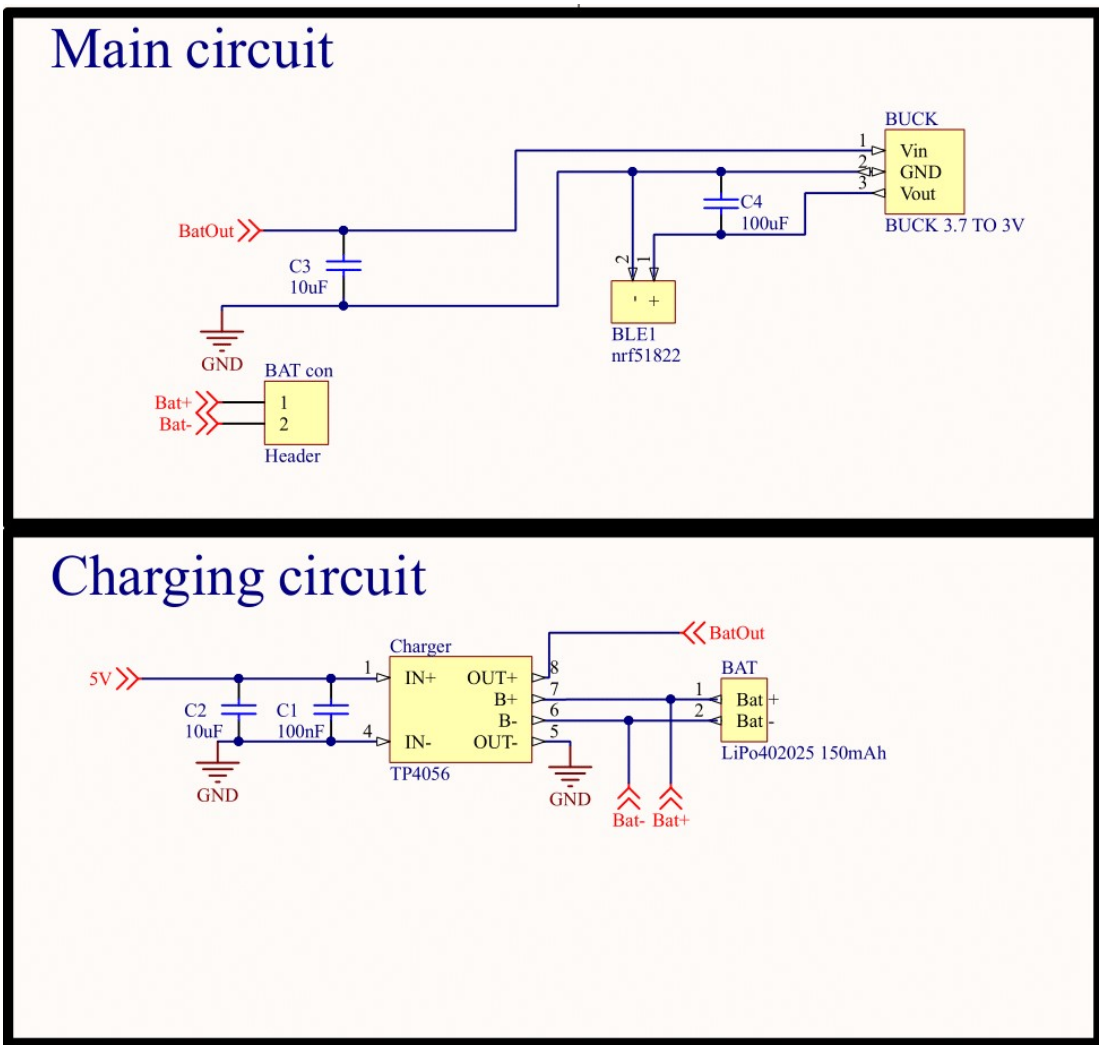
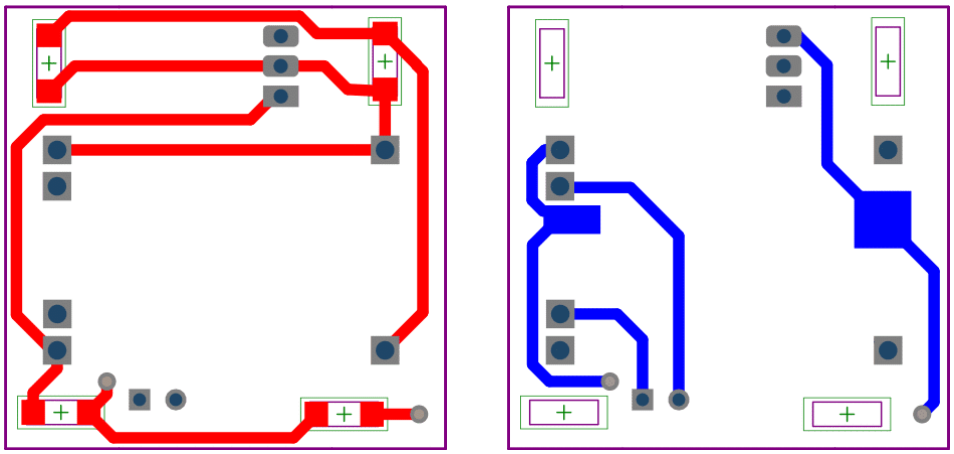
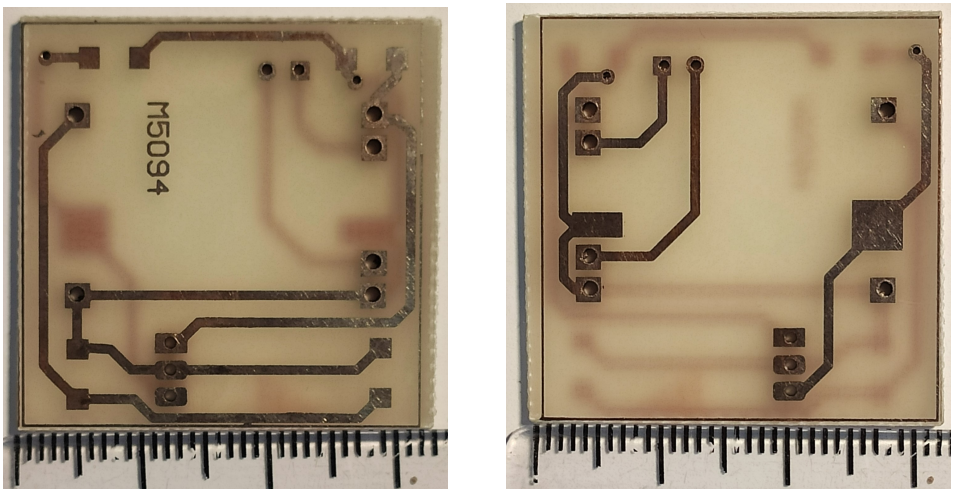


Figure 6.3: Circuit schematic of the modified wearable devices



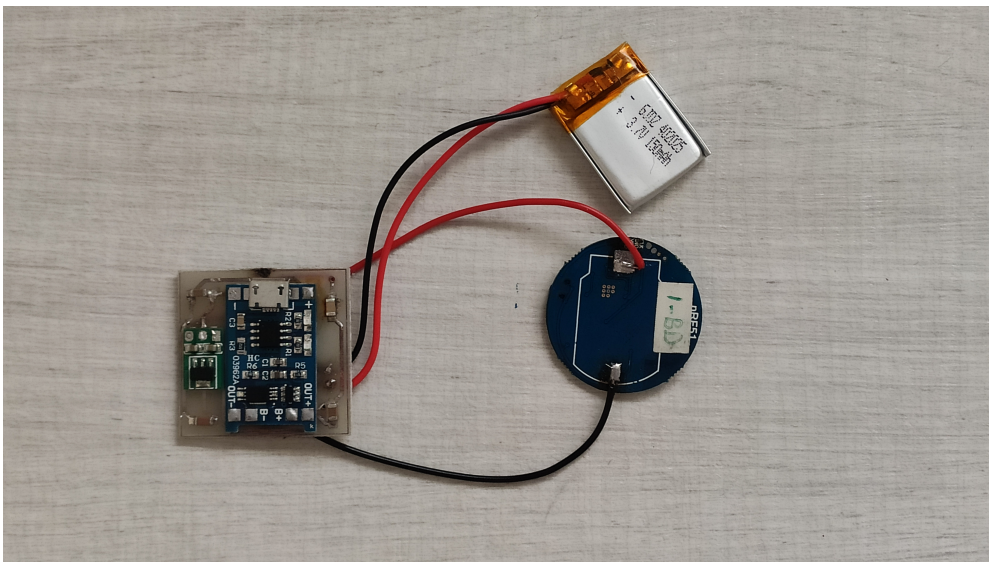
(a) Top layer (b) Bottom layer

Figure 6.4: Designed PCB for the rechargeable wearable devices

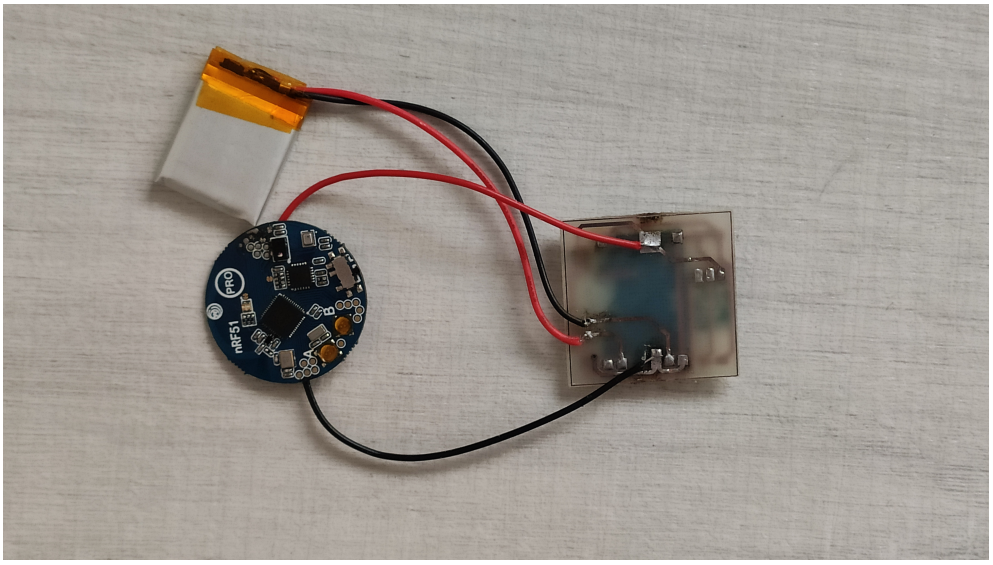


(a) PCB top view (b) PCB bottom view

Figure 6.5: Implemented PCB for the rechargeable sensor tags

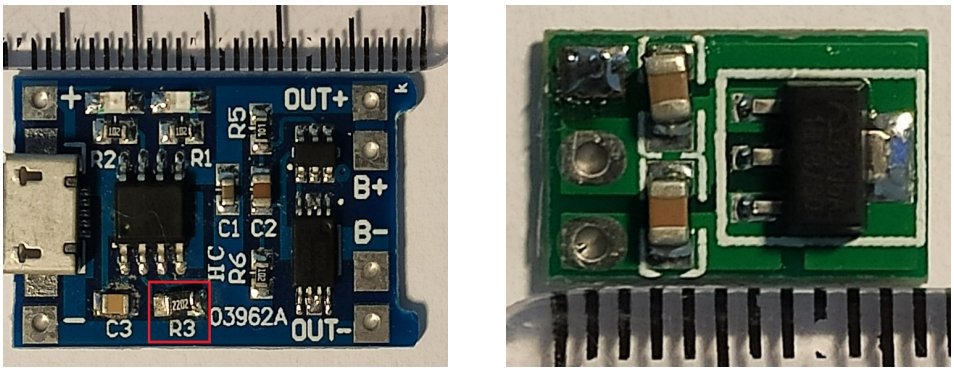


(a) Top view



(b) Bottom view

Figure 6.6: Soldered PCB of a wearable device using SMD components



(a) TP4056 module, the red square highlights the R_{prog} resistor.

(b) DC/DC buck converter

Figure 6.7: Additional required components for the implementation

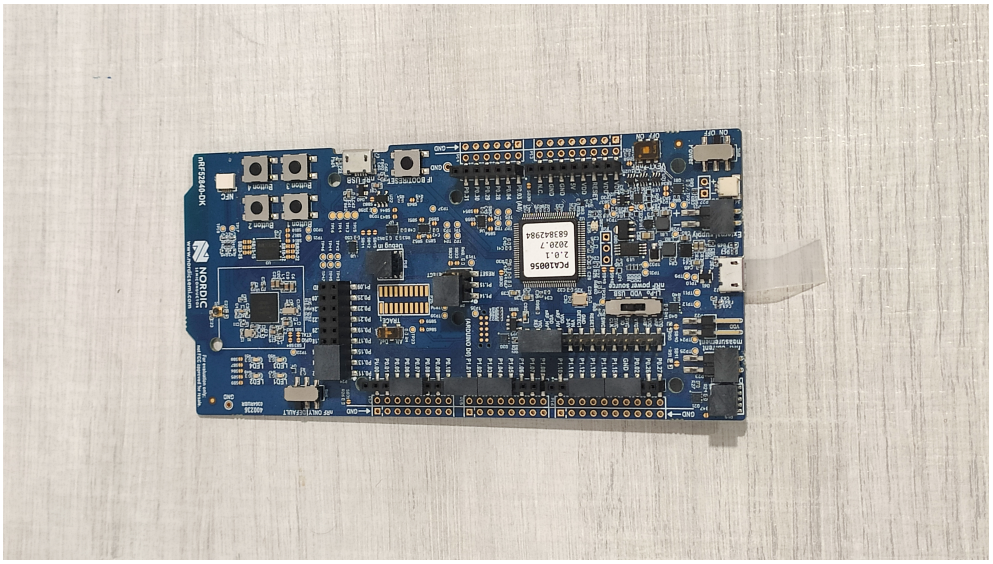
6.1.3 Central receiver

The central receiver in this project works as a connection bridge between the sensor tags and the personal computer. This component performs its purpose by establishing multiple and concurrent secure connections to the sensor tags, receiving the measurement data, constructing the data matrix, and transferring the data matrix to the personal computer.

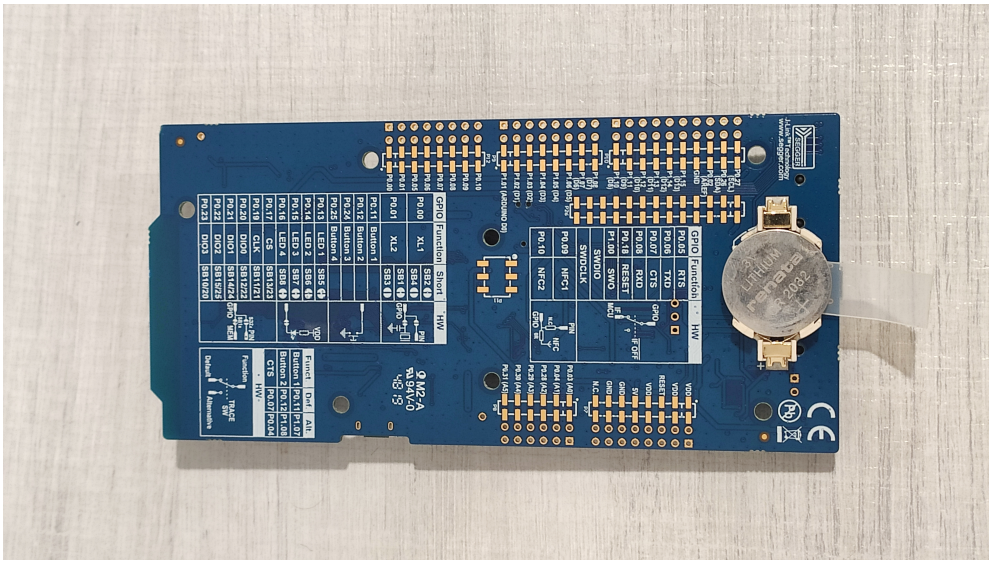
To perform the mentioned tasks, a nRF52840 development kit (Figure 6.8) manufactured by Nordic Semiconductor™ has been considered. Software and hardware specifications of this development kit easily enable the user to form 15 concurrent connections to the sensor tags for real-time measurement. Table 6.7 highlights the important features and product specifications of the development kit from the manufacturer datasheet[29]. It is also worth mentioning that this development kit provides a micro-USB output plug to perform a wired connection to a personal computer. This output port is used also for supplying the development kit, therefore power consumption of the central receiver is not a concern of this solution.

Table 6.7: nRF52840DK features and specifications

Product Name	**nRF52840DK**
Application	Processing unit Wireless communication Interfacing personal computer
Supply Voltage (V) (independent of DCDC converter)	1.7∼5.5V
Operating temperature ^{o}C	$-40 \sim 85$ (Recommended 25)
Processing Unit electrical specifications	
Processor	ARM® Cortex™-M4 32bit
HFCLK	64MHz
LFCLK	32.768kHz
Memory	
Code Size	256kB
Page Size	4096Bytes
Number of pages	256 (4kB each)
RAM Size	256kB
Flash Block Size	1MB
Radio Transceiver	
Operating Frequency	BLE 5 2360∼2500MHz (1MHz channel spacing) 40Channels



(a) Top view



(b) Bottom view

Figure 6.8: nRF52840DK by Nordic Semiconductor™

Chapter 7

Software Design

After setting up the hardware of the solution, it is possible to move on with the final chapter regarding the implementation of a motion-capturing solution. In this chapter, the necessary considerations regarding the software design of this project are discussed. To provide a consistent method for material representation, this chapter is divided into two main sections namely the embedded software design (data acquisition) and the personal computer software design (data representation). The former will explain the logic, scenario, definitions, and the implementation procedure of the embedded software for the central receiver and the wearable devices. The latter will discuss the provided tools regarding data representation on a personal computer.

7.1 Embedded software design

In this section, a detailed explanation regarding the embedded software design of the solution is provided. The embedded software, by interacting with the wearable devices and the central receiver, mainly concerns the data acquisition part of the system. Therefore, in this section, we will address the combination of the central receiver and all the wearable devices as the **acquisition group**.

Due to the software design considerations, it is necessary to indicate the desired features and specifications of the solution. The necessary features regarding the acquisition group are indicated below:

- 15 concurrent, high-speed, and wireless connection must be established between the central receiver and the beacons.
- The connection must be stable and reliable throughout the whole experiment by supporting a reasonable range of distance.
- The central receiver must be able to distinguish the wearable devices from each other to organize the received data to the corresponding body part.
- Central receiver must be able to identify the index of the measurement data for synchronization.
- A lightweight format for data string communication between the central receiver and the wearable devices should be constructed. This data format, which is also referred to as data matrix, should be able to contain the measurement raw data as well as some attributes from the wearable devices.

To implement the mentioned features, it is required to build an embedded firmware for the wearable devices as well as the central receiver. Building an application-specific BLE solution requires introducing the hardware-specific toolchain and the basic BLE concepts which are introduced as follows.

7.2 Hardware specific toolchain

The Nordic Semiconductor embedded hardware, as the processing unit of the wearable devices and the central receiver, enable the ability to implement various embedded applications as well as establishing a low-power, high-speed and wireless communication link. In general, to build firmware on any type of embedded hardware, it is necessary to set up the software development toolchain.

The toolchain provides the required tools and the environment to develop, compile, link, build, and debug the embedded solution. To set up the toolchain, the following items are required:

- **IDE**: An integrated development environment (IDE) is needed to provide the programming environment for the development process. In this solution, Keil μVision was used.

- **SDK**: A software development kit (SDK) is required to provide the necessary libraries, user guides, examples, application programming interface (API) references, and data structures for interacting with the hardware. The SDK v10, which is provided by the hardware manufacturer, was used in this project.

- **SoftDevice**: Since this solution uses BLE communication protocol, using a SoftDevice is necessary. According to the manufacturer documentation, *SoftDevice is a wireless protocol stack that complements an nRF5 series SoC. While it is possible to build applications without using a SoftDevice, all nRF5 SDK example applications that use Bluetooth® Low Energy or ANT™ require a SoftDevice* [30]. This protocol stack contains the Physical and the controller layer of the BLE protocol and allows the BLE applications to interact with the host using GAP, GATT, and L2CAP (see Sec. 7.3). Figure 7.1 illustrates the SoftDevice stack protocol architecture from the developer's documentation.

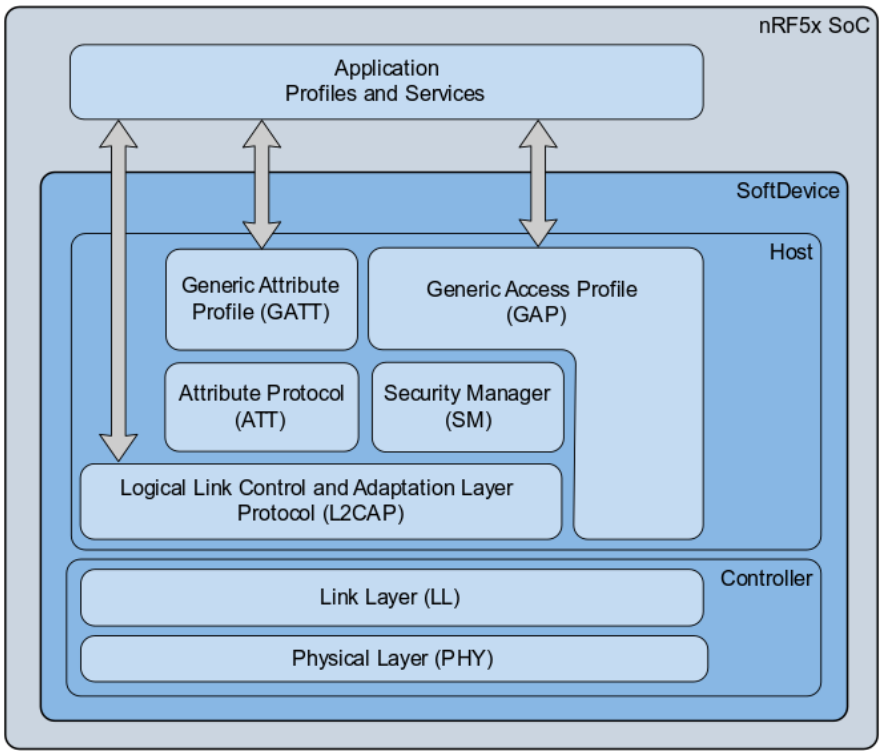


Figure 7.1: SoftDevice stack protocol architecture

7.3 Bluetooth® Low Energy basic concepts

7.3.1 BLE Generic access profile (GAP)

Before defining the BLE generic access profile (GAP), it is necessary to mention the BLE application working modes. There are basically two working modes namely the **advertising mode** and the **connected mode**. The former establishes a unidirectional form of communication for broadcaster-oriented application types, and the latter provides a bidirectional communication tunnel for connection-oriented application types.

For setting a BLE application into the connected mode, it is required to define one device as the broadcaster. The broadcaster always advertises first, therefore, to proceed to the connected mode, it is necessary to pass through the advertising mode first. The frequency bandwidth of the BLE protocol is from 2402MHz

to 2480 MHz with 2MHz of channel spacing which results in 40 channels. The advertising procedure is done using channels 37, 38, and 39 (primary advertising channels).

The GAP defines two main aspects namely the device discovery type and the device connection type. The former describes how devices discover each other using scanning and advertising; the latter describes how the devices will connect [31]. To enlarge these aspects, the GAP parameters are defined below:

- **Roles**: Describe whether the role of application in the advertising mode (broadcaster/observer) or the connected mode (central/peripheral). The former does not require establishing a connection while the latter requires a connection establishment. It is also important to note that a single device can play different roles simultaneously.

- **Modes**: Describe the current state of the BLE device. These states are temporary and they might vary during the working procedure of the application. Some examples of these states are Broadcast, discoverability, connectivity, bonding, etc.

- **Advertisement parameters**: Describe the parameters and settings of the advertiser to send out advertising packages from the primary channels (see Sec. 7.3.2).

- **Connection parameters**: Describe the parameters for establishing a persistent, synchronized, and bidirectional connection (see Sec. 7.3.3).

- **Security**: Describe the required security modes and the security levels of a BLE device for allowing a connection to be established (see Sec. 7.3.4).

7.3.2 GAP Advertisement parameters

Advertisements are the packets that are sent out by the advertiser from the primary channels repeatedly with a fixed time interval. There are different types

of advertisements in a BLE application namely connectable/non-connectable, scannable/non-scannable, and directed/undirected advertisements.

The advertisement interval describes how often a device sends out advertisement packages. This parameter can vary from 20ms to 10.24s with an increment value of 0.625ms. The value of the advertisement interval directly affects the battery life, therefore, this parameter should be chosen carefully for the peripheral devices.

The advertisement data contains information such as device name, transmit power, service UUID, etc. Bluetooth 4.1 supports advertisement packets up to 31Byte length, while Bluetooth 5 supports up to 8 times more than this value. When the application requires a bigger advertisement data capacity, it is possible to use the scan request/scan response method, in which the advertisement packets will be divided into multiple packages on demand of the receiver.

7.3.3 GAP Connection parameters

The connections in a BLE solution are persistent and synchronized links that allow data exchange bidirectionally. For a connection to establish, two main components are required namely the central and the peripheral. In BLE applications, the peripheral acts as a slave and a server simultaneously. The term 'slave' means that the peripheral accepts one and only one central receiver to establish a connection, and the term 'server' means the peripheral updates the central with advertisement packets. Therefore, due to this terminology, the central becomes the master and the client.

The peripherals always start the connection establishment procedure by initially sending advertisement packets from its primary advertisement channels. The central receiver, on the other end, is listening to the advertisement packets. When the central receiver receives the desired advertisement package, it will respond using a connection request on the same channel frequency in which the

package was received. The connection is considered *created* when the peripheral receivers the connection request. After an interval, which is also known as the connection interval, a data packet will be sent from the central to the peripheral and the peripheral is should respond to this packet as well. After the response from the peripheral, the connection is successfully *established*. Figure 7.2 illustrates the connection establishment procedure.

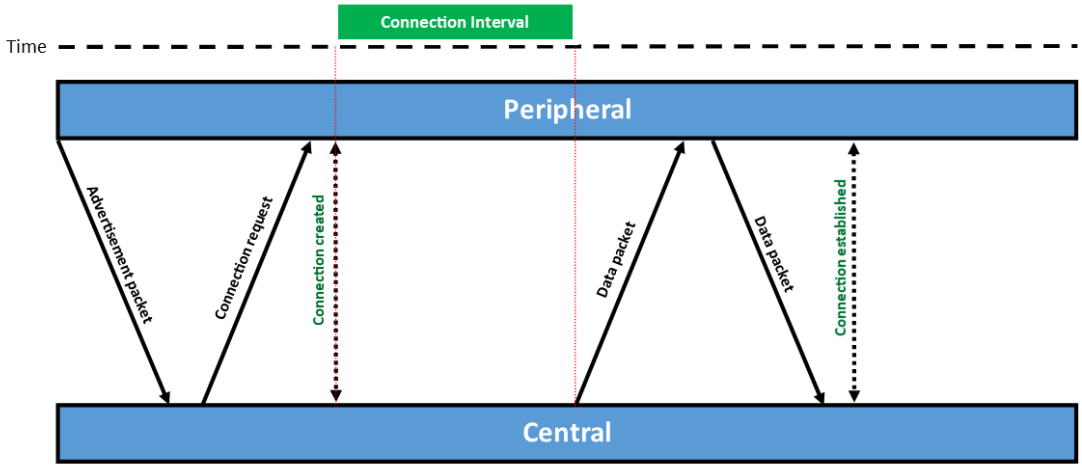


Figure 7.2: Connection establishment procedure

Each connection request contains several parameters, in which some of which are crucial to the performance and functionality of the solution. These parameters are mentioned below:

- **Connection interval**: Determines when a connection event occurs. This value varies from 7.5ms to 4s with an increment value of 1.25ms. It is important to note that the slave must always respond to the data packets that come from the master. In the event of data unavailability, the slave must respond with an empty packet, otherwise, the connection event will be closed.

- **Slave latency**: Defines the number of data packets that the slave can leave without a response, while the connection remains open. This value

indicates the number of incoming packets from the master which will be ignored by the slave.

- **Supervision timeout**: Determines the amount of time since the last data exchange from two connected devices, in which the connection will remain open. This parameter varies from 100ms to 32s with a step value of 10ms. The supervision timeout also has the following relationship with the value of slave latency.

$$Supervision > (1 + Slave\_latency) \times Connection\_interval \times 2 \qquad (7.1)$$

The Eq. 7.1 indicates that the maximum value of the slave latency is limited to 500.

- **Channel map**: Each connection event is done on a different RF channel. The channel map provides the list of the best communication channels among the 37 data transfer channels. The criteria for choosing the best channel is to select the channels that have minimum interference with the other broadcasting devices in the area.

- **Hop sequence**: In the process of selecting the best channel from the channel map, it is possible to define the Hop parameter to indicate the number of channels to skip during the selection procedure. For example, if the starting data transfer channel is ch1 and the Hop parameter is 4, the next channel to analyze is ch5. If the number of channels exceeds the maximum amount of channels (40), the process will start over from channel 1.

7.3.4 GAP Security parameters

Security modes and levels are among the other aspects that GAP introduces. These modes are implemented using the security manager (SM) layer [32]. Table 7.1 defines the security modes and levels in a BLE application obtained from the Bluetooth® core specifications [33].

Table 7.1: BLE security modes and levels

Mode	Level	Description
Mode 1	Level 1	No security
	Level 2	Unauthenticated with encryption
	Level 3	Authenticated with encryption
	Level 4	LE secured authentication with encription
Mode 2	Level 1	Unauthenticated with data signing
	Level 2	Authenticated with data signing

7.3.5 BLE Generic attribute profile (GATT)

The generic attribute profile relates to the concept of data transfer during the event of connection. The term 'ATT' comes from the attribute protocol, which defines how the server exposes its data to the client and how the data is structured. The attribute protocol defines two roles for the communication procedure, namely the server and the client.

The server role is dedicated to the device that exposes its data to inform or control. This type of role may accept commands from peer devices and it is capable of reacting using responses and notifications. On the other hand, the client role is related to the device that interacts with the server to receive the exposed data or to control the server behavior. This type of role is capable of sending

commands, sending requests, and also listening to the incoming notifications from the server.

The attribute protocol (ATT) structure contains several parameters, which are provided below [34]:

- **Handle**: 16-bit unique identifier for each attribute. This property is used to address the attributes and it remains constant throughout the whole operation.

- **Type**: Defines the type of the available data using a universally unique identifier (UUID). The UUID size varies from 16-bit to 128-bit, in which the 16-bit and 32-bit UUIDs are dedicated to the types that are identified by Bluetooth® Special interest group (BT SIG) and 128-bit UUIDs are for the custom made types.

- **Value**: This parameter is the data that the server intends to expose. The length of this parameter is variable and it depends on the attribute type.

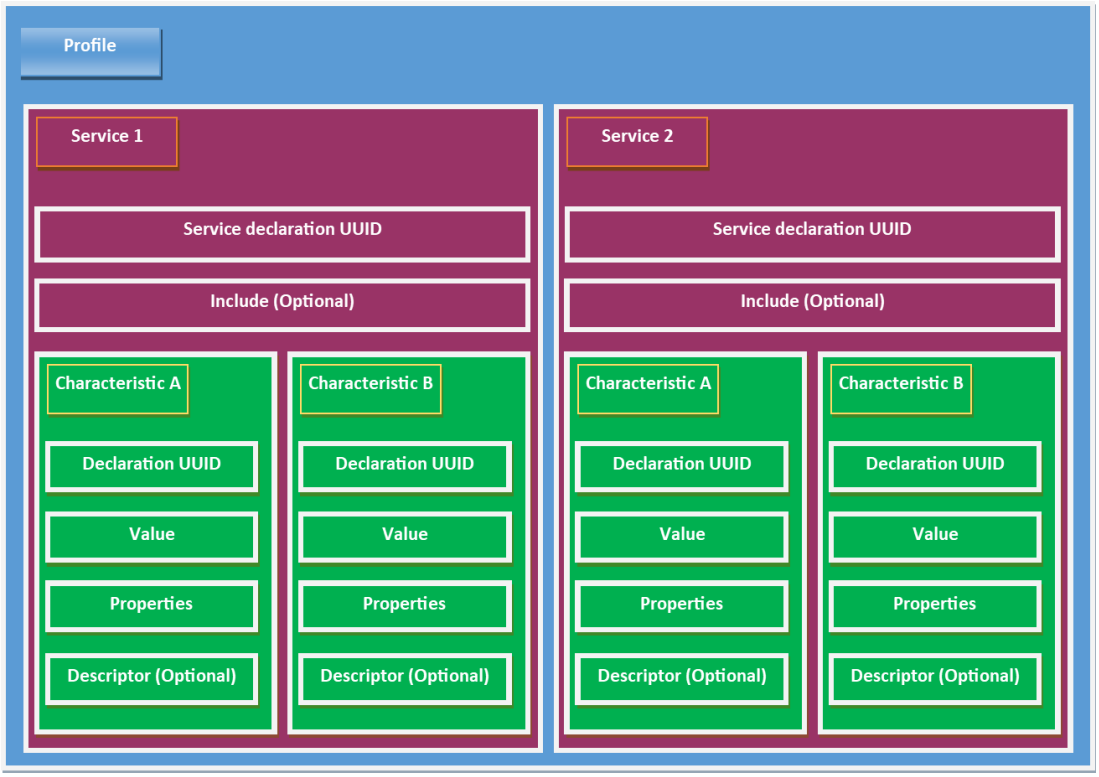
- **Permission**: Provides information about the executable ATT operations and their security requirements.

The GATT defines features and service characteristics, as well as the procedure to interfere with the attributes. The services are distinguished by a declaration UUID and they are a combination of one or more attributes to perform a functionality. These attributes can be characteristics, which are pieces of information that the server wants to expose, or non-characteristics.

The characteristics, also similar to the services, are distinguished by a declaration UUID. Furthermore, the characteristics have two additional parameters namely the properties, and the descriptor. The former describes the operation that can be performed using this characteristic such as read, write, write without response, and notify; and the latter provides additional information about the

characteristic. Among all the types of descriptors, the client characteristic configuration descriptor (CCCD) is the most important descriptor for a BLE profile. The CCCD is a switch that can enable or disable server updates.

By using one or more services together, it is possible to form a BLE application profile. In essence, an application profile describes the overall functionality of the solution, while the services define the sub-functionalities and the characteristics that perform those functionalities. Figure 7.3 illustrates the general hierarchical model of the BLE application profile.



The h

Figure 7.3: Hierarchical model of the BLE application profile

7.4 Peripherals application

As previously indicated, the processing component of the wearable devices is the nRF51802 SoC by Nordic Semiconductor™. This SoC is in charge of service

initialization, measurement readings, calibration, and BLE communication.

Like any other embedded solution, the main functionalities of wearable devices are built by a set of instructions. These instructions are written using the Embedded C programming language, and they tend to initialize the embedded device. After the initialization process, the program repeatedly executes the core functionalities of the application.

Since the embedded application of the wearable devices was done by our colleague researchers at Politecnico di Torino, we are not allowed to expose the source codes of the application, hence a general description of the core functionalities of the application is provided accordingly.

7.4.1 Toolchain

Due to the previously described concept of the hardware-specific toolchain, Table 7.2 describes the development toolchain specifications of the wearable devices. It is important to note that the toolchain is exclusively defined by the hardware and the application, therefore, it might not be compatible with another kind of hardware or other types of applications.

Table 7.2: Toolchain features and specifications

Tool	Version	Producer	Purpose	Features
Keil μVision	5	ARM ltd. ARM germany GmbH	Integrated development environment	Project management Run-time environment Compile and Build Debug
SDK	10	Nordic Semiconductor	Software development kit	Hardware drivers Libraries Examples User Guides API References DS documentations
SoftDevice	S110	Nordic Semiconductor	BLE communication	Bluetooth 4.1 stack Asynchronous behavior No RTOS dependency Multiprotocol operation support

7.4.2 Initialization process

The initialization process requires activating the necessary services and peripherals such as pins, the timer interrupts, radio transceiver, initial calibration service, etc. Table 7.3 provides brief documentation about the necessary functions during the initialization process in the order of execution.

Table 7.3: nRF51802 SoC function description for initialization process

Function	Type	Arguments	Description
pin_init()	Void	NULL	Initialize the GPIOs of the SoC
APP_TIMER_INIT()	Macro	PRESCALER OP_QUEUE_SIZE scheduler_function	Allocates memory for internal queues and performs the library initialization.
ble_stack_init()	Void	NULL	Initialize the SoftDevice Handler and the BLE stack registers Mesh handler for SoC events.
Buffers_init()	Void	NULL	Clear All Buffer slots, resetting their buffer pointer
uSerialInterface_Init()	Void	task_id	Initialize serial interface of the SoC
gap_params_init()	Static void	NULL	Initialize generic access profile initialization
services_init()	Static void	NULL	Initialize application services
advertising_init()	Static void	NULL	Initialize advertising functionality
conn_params_init()	Static void	NULL	Initialize the Connection Parameters module
miuProgram_Setup()	Void	NULL	Initialize core application parameters
MPU6050_dmpLoadCalibration	Int	NULL	Initiate calibration using digital motion processing (DMP) unit. When ready, !=Null
Start_IMU_Measurement()	Void	NULL	Reset and enable FIFO, start the measurement

7.4.3 Core functionalities

The core functionalities of the wearable devices are acquiring the measurement data from the inertial sensors, constructing a data string structure based on the measurement data, initiating a connection to the central device, and communicating the data string using the advertisement packets. Therefore, the core functionalities are divided into three main sections, namely the measurement part, the advertisement packet construction, and the BLE communication part,

which will be described separately.

The measurement part

As previously mentioned, the attitude estimation process based on the raw measurement data of the inertial measurement units is prone to accumulated errors. Therefore, a calibration procedure for the gyroscope values is needed. The measurement procedure of the wearable devices is done using an asynchronous scheme, in which the calibration process is done in parallel with the measurement process. Figure 7.4 demonstrates the parallel command-based interruption algorithm of the measurement program, Figure 7.5 illustrates the flowchart of the measurement procedure, and Table 7.4 describes the copyrights of the mentioned programs.

Table 7.4: Copyrights of the resources

Program	**Copyright**	**Availability**
MPU6050.c	Jeff Rowberg (I2C library collection)	open source
dmp.c	InvenSense ltd.	open source
Calibrate.c	Simone Corbellini (Polito)	All rights reserved
miuProgram.c	Simone Corbellini (Polito)	All rights reserverd

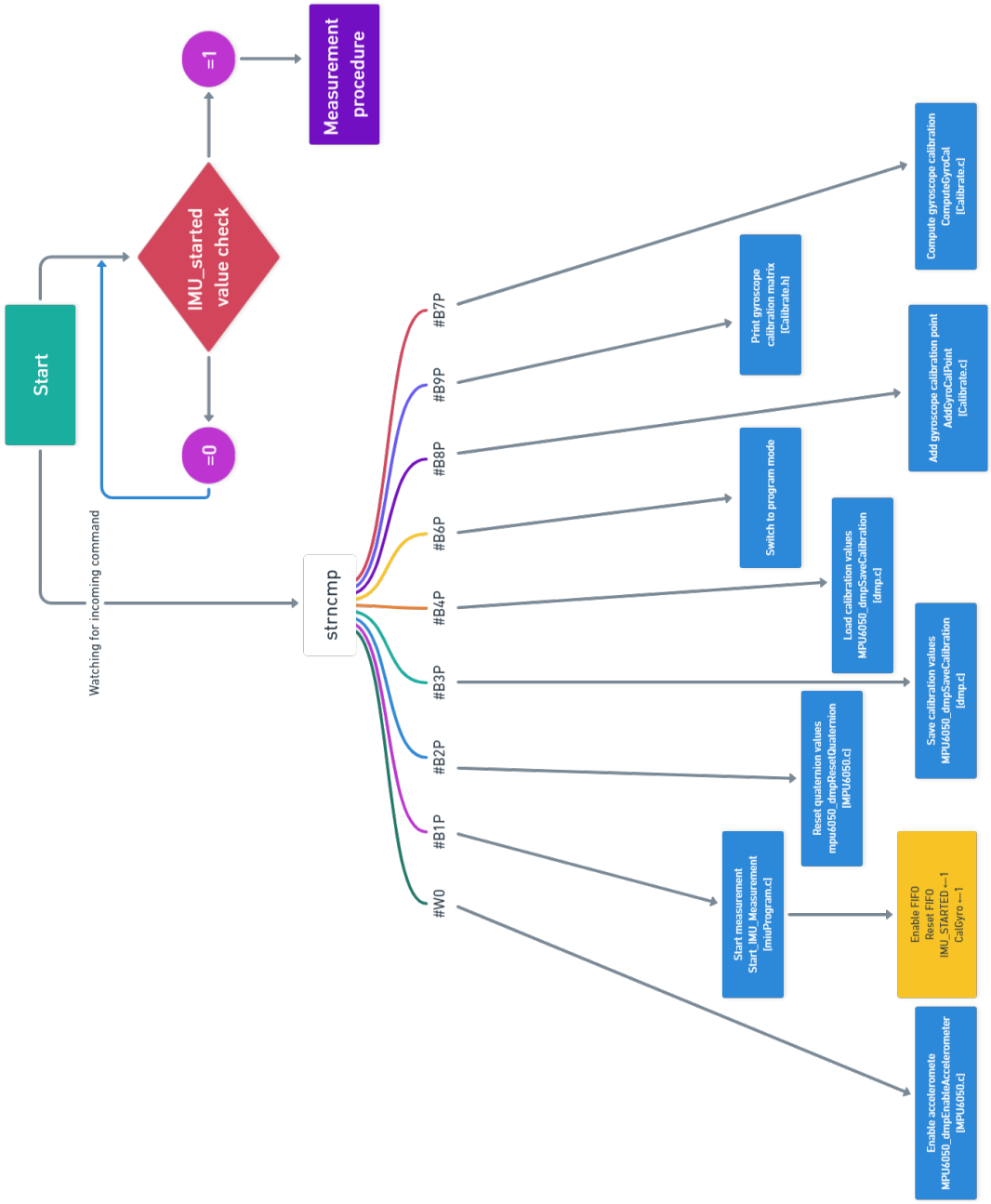


Figure 7.4: Command-based interruption algorithm

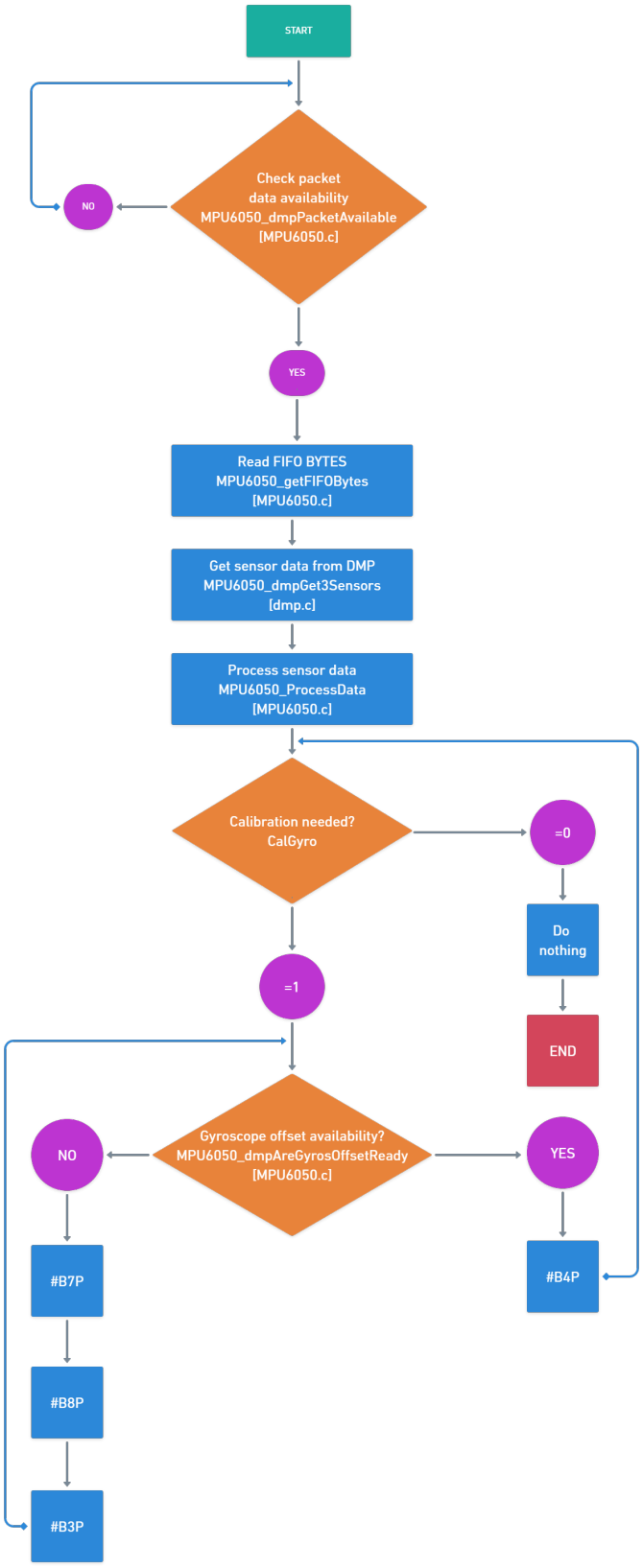


Figure 7.5: Measurement procedure flowchart

Advertisement packet construction

The peripheral communicates data through the advertisement packets to the central. Since the attributes and the advertisement packets have large data sizes, it is necessary to divide the advertisement data into smaller data packets. Therefore, three types of data packets regarding the payload have been introduced for specific purposes, which are explained in Table 7.5.

Table 7.5: Description of the advertisement packets

Packet type	Data type	Description
Type #0	Short int	Yaw-Pitch-Roll angles with 0.01^{o} resolution
Type #1	Short	Accelerometer X,Y,Z, and magnitude[1]
Type #2	Short	Integral of gyroscope signals (X,Y,Z) with 0.01

Advertisement packets are 31 Byte long data strings that include information such as GAP parameters, the peripheral address, the manufacturer-specific data, and the advertisement value. Figure 7.6 illustrates the advertisement data package structure.

7.4.4 BLE communication

Due to the previously described concepts about BLE applications, this solution requires a peripheral/central type of communication. the wearable devices in this solution are the peripherals that connect to a single central receiver. The peripherals expose the measurement data to the central receiver, therefore, the central receiver is the client and the peripherals are the servers. On the other hand, since multiple peripherals connect to one central, the central receiver is the master and the peripherals are the slaves.

To use the aforementioned service, it is required to set up a BLE link by

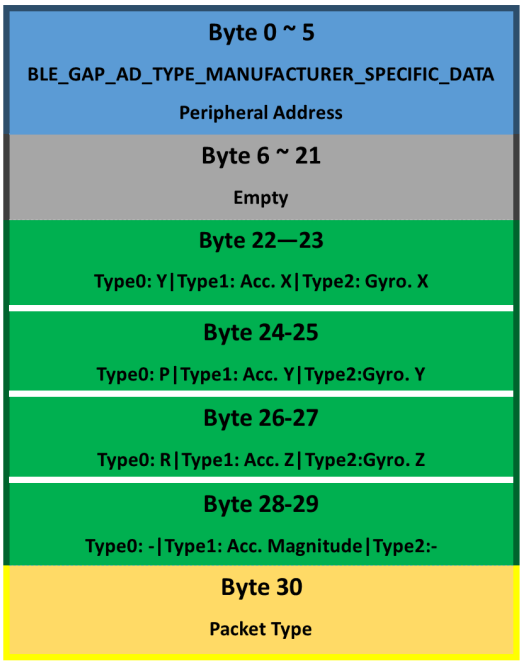


Figure 7.6: Structure of the advertisement packets

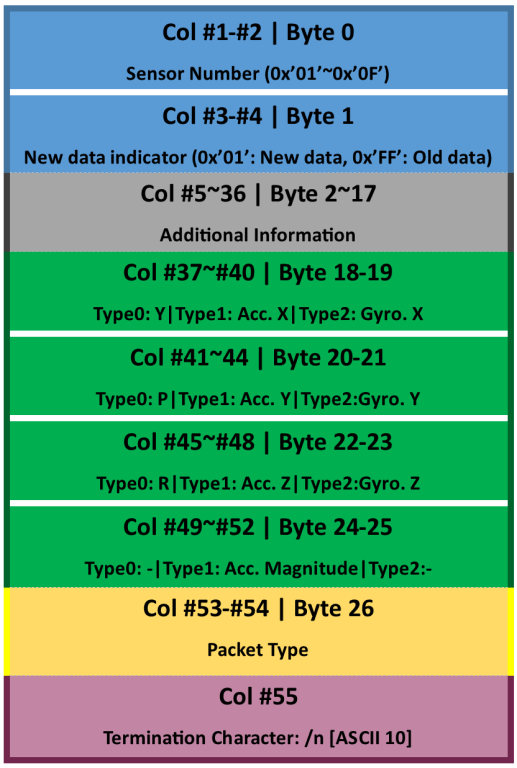


Figure 7.7: Data matrix structure

defining the GAP parameters namely the role, advertising parameters, connection parameters, and the security level. Table 7.6 defines the mentioned parameters.

Table 7.6: GAP parameters of the peripherals

Parameter	Value
GAP general parameters	
Role	Peripheral(Slave/Server)
Device name	homeTag
GAP Security parameters	
Security mode	1
Security level	1

GAP Connection parameters	
Maximum connection interval (ms)	30 (1.25ms step)
Minimum connection interval (ms)	10 (1.25ms step)
Supervision timeout (ms)	5000 (10ms step)
Slave latency	0
Hop sequence	0
GAP Advertisement parameters	
Advertise device name	true (Full name)
Advertising interval (ms)	64 (0.625ms step)
Advertising timeout (s)	180
Advertise manufacturer data	False
Include device address	true
Include appearance	False
Active advertisement channels	37,38,39

After defining the GAP parameters and establishing the BLE link, it is possible to form an application profile to define the data structure, data exposure, functionality, and behavior of the BLE application. To implement a BLE application, capable of transferring data bidirectionally, it is possible to use the NUS service, which is exclusively introduced by Nordic Semiconductor with a 128-bit UUID. The NUS service stands for Nordic UART Service. This service provides an asynchronous and bidirectional data transfer link between the peripherals and the central. The NUS service contains two characteristics which are named af-

ter the wired UART communication, namely BLE_NUS_TX and BLE_NUS_RX. Figure 7.8 illustrates the BLE application profile of the wearable devices.

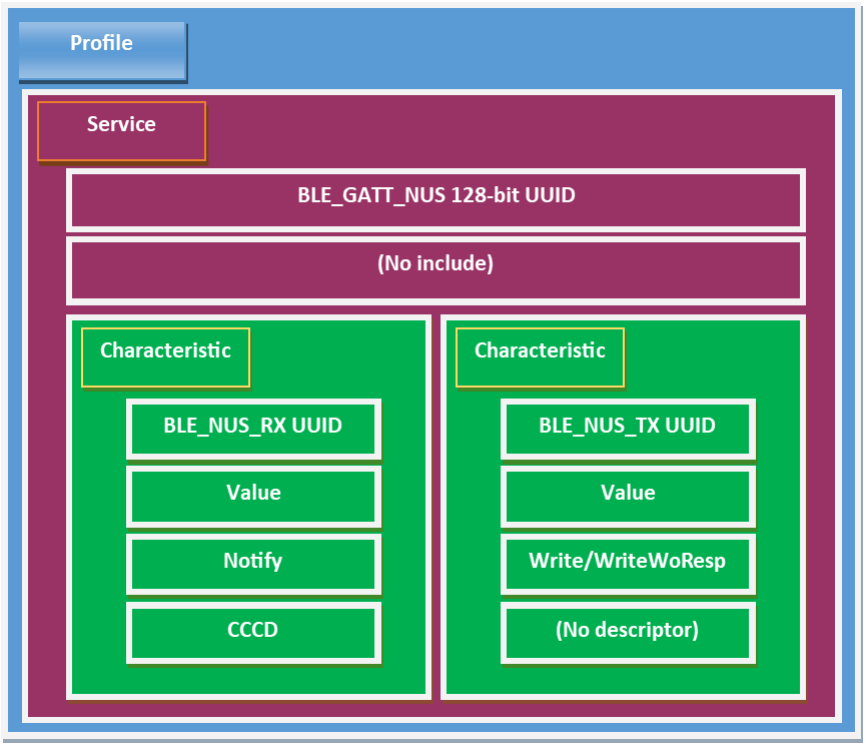


Figure 7.8: BLE Application profile of the wearable devices

7.5 Central application

The processing component of the central receiver is the nRF52840 SoC by Nordic Semiconductor^{TM}. The purpose of this component is to receiver the advertisement packages from the wearable devices by establishing 15 concurrent BLE connections, distinguish the measurement data index, indicate the incoming data to the corresponding wearable device, form the data matrix based on the data index and the corresponding sensors, and transfer the data matrix to a personal computer by establishing a serial communication.

Similar to the case of wearable devices, the main functionalities of the central receiver are built by a set of instructions, which are written in the Embedded

C programming language. These instructions include the initialization of the required SoC services, hardware peripherals, and the core application loop that repeatedly performs the core functionalities.

The current application of the central receiver is implemented on top of the *Nordic UART service over BLE for the central* example. This example provides a bidirectional connection link between the central receiver and one peripheral. The example has been modified to support 15 concurrent connections.

7.5.1 Toolchain

It is important to note that the toolchain which was previously described for the wearable devices is not used for the central receiver. This difference is because the SDK and SoftDevice are hardware-specific resources that may differ based on the SoC type. In addition, the SoftDevice version is also optimized for the role that the device is intended to work on. The full SoftDevice compatibility list is available by the manufacturer's manual [35].

Table 7.7 describes the toolchain for the development of the central receiver embedded application.

Table 7.7: Features and specifications of the central embedded application

Tool	Version	Producer	Purpose	Features
SEGGER Embedded studio	5.60	SEGGER Microcontroller GmbH Rowley Associates ltd.	Integrated development environment	Project management Run-time environment Compile and Build Debug
SDK	15.2	Nordic Semiconductor	Software development kit	Hardware drivers Libraries Examples User Guides API References DS documentations
SoftDevice	S140	Nordic Semiconductor	BLE communication	Bluetooth 5.1 stack High-throughput 2 Mbps Concurrent multiprotocol support

7.5.2 Initialization process

The initialization process of the central application consists of two steps. The first is the service definition, and the second is the function initialization. In essence, the service definition is done by defining the required services and hardware components that are required during the application. For this purpose, the SDK configuration header file needs to be modified. This file describes the static configurations of an application that is built on top of an SDK. These configurations include the service definitions, general configurations, library dependencies, and required embedded peripherals. The modification can be done manually by a simple text editor or graphically using the CMSIS Configuration wizard graphical user interface (GUI). Table 7.8 describes the list of the services that are defined for the central application.

Table 7.8: SDK Configuration header file

Service	Description
nRF_BLE	
BLE_DB_DISCOVERY	UUID database discovery module
NRF_BLE_SCAN_ENABLED	Scanning module
nRF_BLE_Services	
BLE_NUS_C_ENABLED	Nordic UART service for central
nRF_Drivers	
NRFX_PRS_ENABLED	Peripheral resource sharing module
APP_UART_FIFO_INIT	Application UART FIFO functionality
nRF_Libraries	
APP_TIMER_ENABLED	Application timer functionality

Table 7.8 continued from previous page

Service	Description
NRF_BALLOC_ENABLED	Block allocator module
NRF_FPRINTF_ENABLED	fprintf function enabled
NRF_MEMOBJ_ENABLED	Linked memory allocator module
NRF_PWR_MGMT_ENABLED	Power management module
NRF_STRERROR_ENABLED	Convert error codes to strings
nRF_Log	
NRF_LOG_BACKEND_RTT_ENABLED	Backend log real-time transfer
NRF_LOG_ENABLED	Logger module
nRF_SoftDevice	
NRF_SDH_BLE	SoftDevice handler
NRF_SDH_BLE_ENABLED	SoftDevice BLE event handler
NRF_SDH_SOC_ENABLED	SoftDevice SoC event handler

After defining the service definitions, it is necessary to initialize some of the service functionalities. It is important to note that not all of the services require initialization. Hence, only those services that require configuration parameters for operation require initialization. In addition, not all of the initialization statements are corresponding to the services, ultimately, they are corresponding to the core functionality of the application. Figure 7.9 illustrates the initialization statements as well as their corresponding configuration program file.

Figure 7.9: Initialization statements for the central receiver application

7.5.3 Core functionalities

The core functionality of the central receiver application includes scanning for a BLE device with an activated NUS service, receiving the advertisement package from the corresponding device, store the measurement data as well as the unique peer address of the sender into the data matrix, and transfer the data matrix to the personal computer for further processing. Since these functionalities need to be executed continuously, they are implemented using the main loop of the embedded program.

When a connection link is established with a peripheral, initially, the BLE event handler function checks the total number of active links. If this number is less than the maximum number of links (15), the central will continue scanning for new devices, otherwise, the scanning process will be terminated. In parallel, the BLE event handler listens to the incoming advertisement packets from the active links. By comparing the peer address of the incoming packet with the pre-defined list of peer addresses corresponding to the sensors, the central will manage to form the data matrix.

The data matrix is a $n \times 55$ 2-dimensional matrix that the parameter $'n'$ corresponds to the number of sensors (see Table 5.2). Therefore, each row is dedicated to one sensor which is distinguished by its peer address. On the other hand, the columns are measurement data that are received from wearable devices. It is important to mention that each row of the data matrix contains 27 Bytes of data which are divided into 2 columns and the 55^{th} column is the termination character (ASCII 10). Flowchart of the core functionality of the central application is provided in Figure 7.10, and the data matrix structure is illustrated in Figure 7.7 (The figure is intentionally placed next to Figure 7.6 for a better comparison). In addition, The source codes of the application are provided in the appendix.

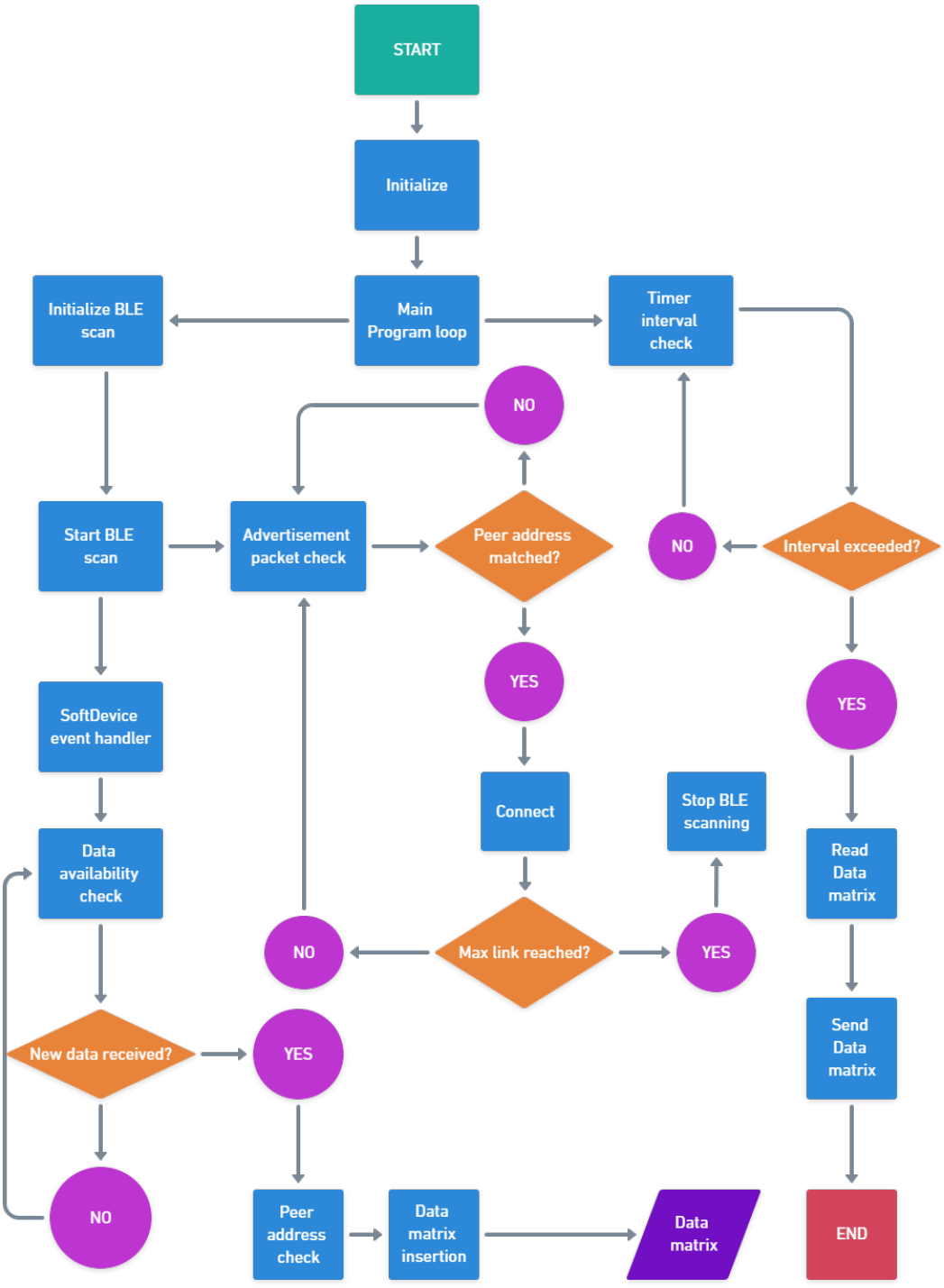


Figure 7.10: Flowchart of the core functionalities of the central applications

7.5.4 Communication Parameters

The central receiver establishes two forms of communication tunnels which a BLE connection links with peripherals and a serial communication a personal computer. These connections are specified in Table 7.9.

Table 7.9: Communication parameters of the central receiver

BLE link	
Parameter	**Value**
Scan interval	160 (Unit 0.625ms)
Scan session duration	0
Scan window	80 (Unit 0.625ms)
Minimum scan connection interval	7.5ms
Maximum scan connection interval	30ms
Scan buffer	31 Bytes
Total link count	20 (Available by SoftDevice)
Central link count	15 (Peripherals)
Serial communication	
Parameter	**Value**
UART TX Buffer size	8192
UART RX Buffer size	256 (default)
IRQ priority	Lowest
Baud rate	230400
Hardware flow control	Disabled
Parity	Disabled

7.6 Data representation

As formerly mentioned, in order to represent the measurement data (rotation angles), data representation softwares were created by means of windows applications. These applications use various techniques to interpret the incoming data from the central receiver. Due to the data interpretation process, the incoming data packets are decoded and the measurement values are extracted for attitude representation. Each windows application is provided with a GUI for the user and they are created using three different software environments namely MATLAB, NI LabView, and Unity game engine.

The general principle for the data interpretation process in all of the three software environments is to decode the incoming hexadecimal data matrix from the serial interface. The decoded data will be converted into Yaw-Pitch-Role and Quaternion angles. After that, the converted angles will be represented numerically and graphically using the implemented applications. The principle of data representation followed by a more detailed explanation for each software environment are provided in this section

7.6.1 Data interpretation process

Using the data matrix architecture explained in the previous section, it is possible to extract information such as sensor number, new data availability, data packet type, and measurement values. It is important to note that the raw data matrix contains hexadecimal values and the measurement angles are floating-point numbers, therefore, type conversion is needed to obtain the measurement values. Type conversion is done using the IEEE 754-2019 Standard [36] for single-precision floating-point arithmetic which converts hexadecimal strings to 32-bit single-precision floating-point numbers.

Due to the previously mentioned Euler angle singularities (see Sec 4.2), the section corresponding to Quaternion data from the data matrix is considered for

the attitude representation process (see Figure 7.7). Since the Euler angles are easier to understand compared to the Quaternions, the Quaternions are eventually converted to the Euler angles using Eq. 4.41. In this way the represented numerical data are more understandable and the singularities due to the Euler angles are avoided.

7.6.2 Data representation using MATLAB

Description

The purpose of this application is a simple graphical representation of the sensor data on a disk-shaped object. After defining the serial communication parameters, the program decodes rows and columns of the incoming data matrix. The sensor number is extracted using the first pair of matrix columns. Then, the next pair columns indicate the sensor's status (ON/OFF) and the data packet availability.

Furthermore, the Quaternion values, which are placed from the 5^{th} to 36^{th} columns of the data matrix by the central receiver, are extracted. Moreover, these values are converted to single-precision floating-point numbers using the *hexsingle2num* function [22]. Later, the numerical Quaternion values are converted to Euler angles using the *quat2angle* built-in function of the MATLAB program.

Eventually, the Quaternion values and the calculated Euler angles are represented in the debug terminal of the MATLAB program. Finally, these values are also used to rotate a circular surface (with the shape of the peripherals) in the 3D space based on the incoming measurement data.

Figure 7.11 indicates the flowchart of the MATLAB application, Listing 1 shows the terminal output during a measurement process, and Figure 7.12 illustrates the 3D data representation. In addition, the codes of this application are available in the Appendix as well as the Github repository of this project [37].

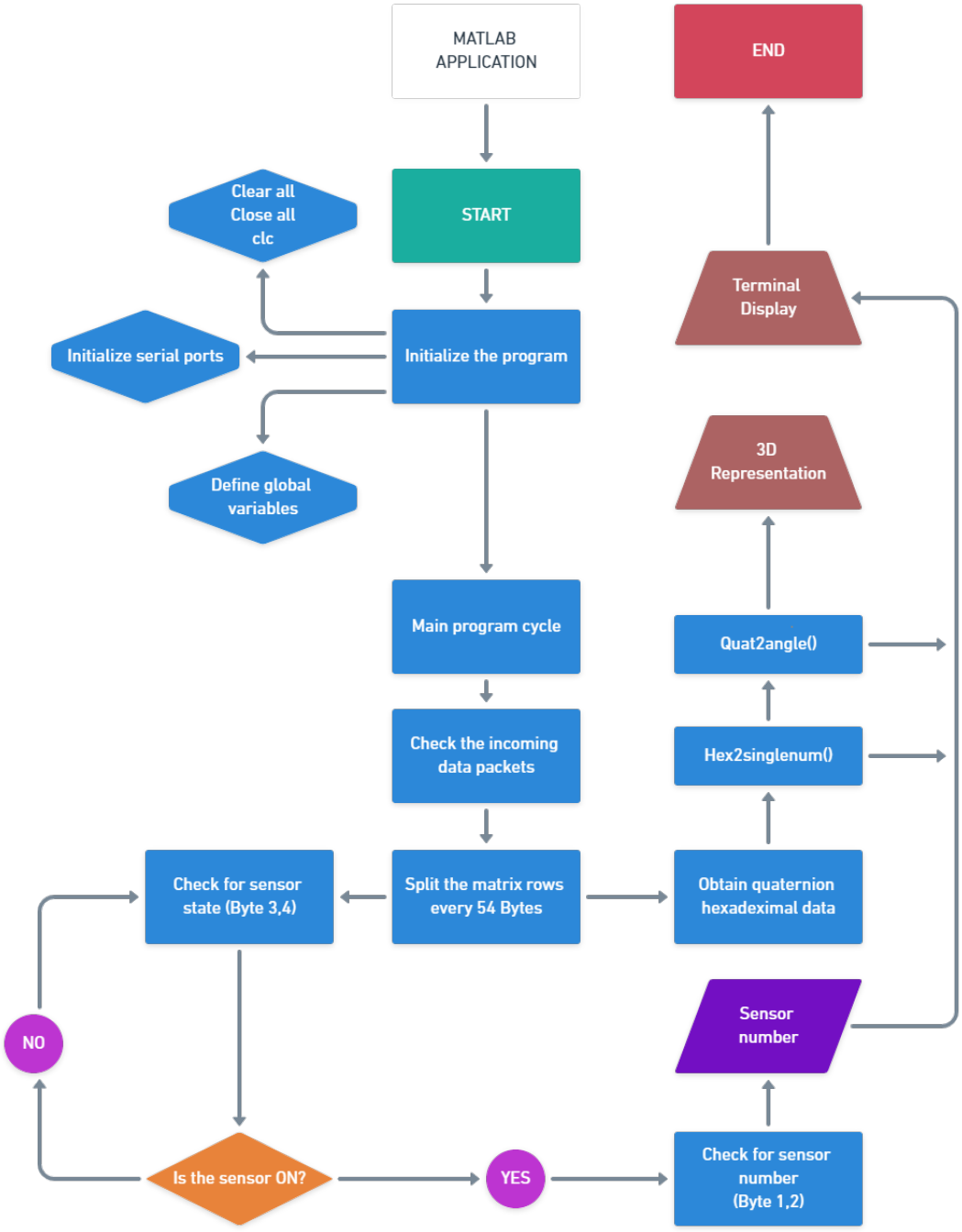


Figure 7.11: Flowchart of the MATLAB data representation application

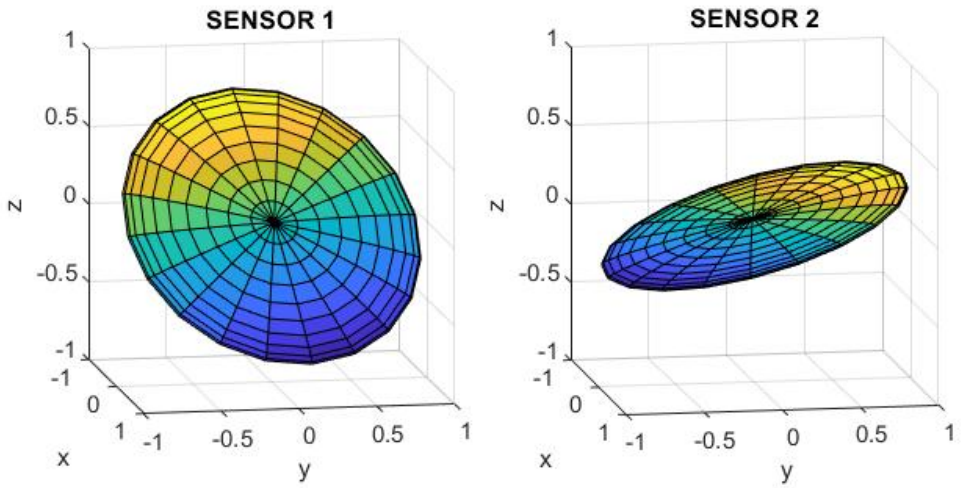


Figure 7.12: 3D data representation of 2 sensors using the MATLAB application

```
quatW_s1 : 0.996276 quatW_s2 : 0.967565
quatX_s1 : 0.078654 quatX_s2 : 0.015171
quatY_s1 : 0.007692 quatY_s2 : -0.011132
quatZ_s1 : 0.034479 quatZ_s2 : 0.251919

Yaw_s1   : 4.009102 Yaw_s2   : 29.167585
Pitch_s1 : 0.567438 Pitch_s2: -1.672434
Roll_s1  : 9.047937 Roll_s2: 1.361458
```

Listing 1: Output terminal of the MATLAB application

7.6.3 Data representation using LabView

This software aims to provide a GUI that represents the serial communication parameters, the incoming data, and the data representation with respect to the

elapsed time on a waveform chart for each sensor. The application also provides the ability to store measurement data in a text-plain format for later analysis. In addition, The serial communication parameters can be modified by the user during various applications.

This application GUI consists of two main pallets:

- **The serial communication pallet**, which uses VISA functions and VIs to perform a continuous serial read operation with the defined serial parameters.

- **The representation pallet**, which is used to represent processed data matrix values numerically on text boxes and graphically on waveform charts. The user, by changing the state of the $'YPR/Quat'$ switch in Figure 7.14a, can choose to view either Quaternions or Euler angles from the waveform chart. The former consist four separate lines for each of the Quaternion components and the latter consist three separate lines for rotations around the Yaw, Pitch, and Roll axes. In addition, there is also a file name input that assigns a .txt file name to store the measurement data during the experiments separately for each sensor.

Figure 7.13 indicates the flowchart of the LabView windows application, Figure 7.14 illustrates the GUI of the application, where In Figure 7.14a, white line: W, red line: X, green line: Y, blue line: Z. This application is also available on the Github repository of the project [37].

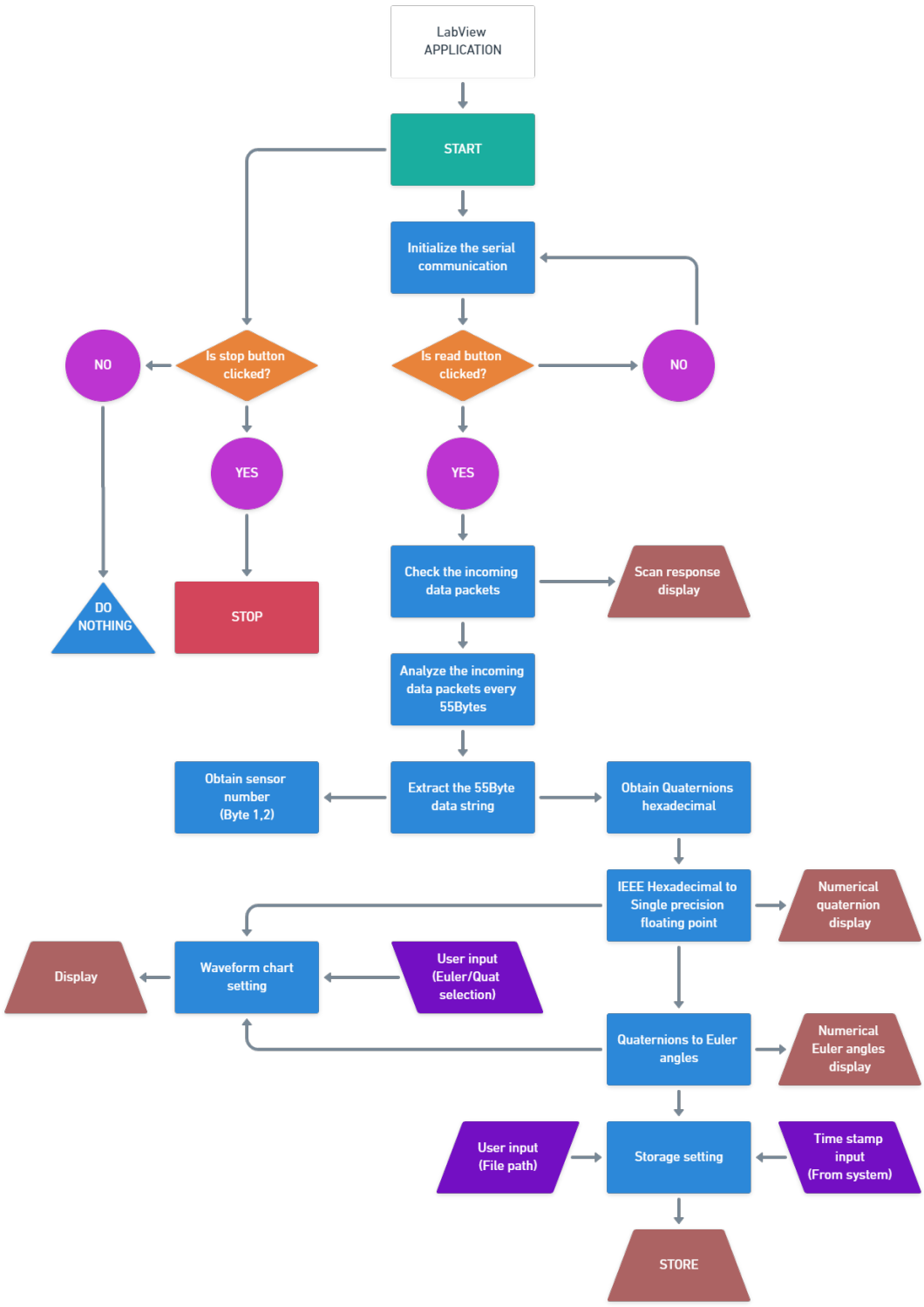
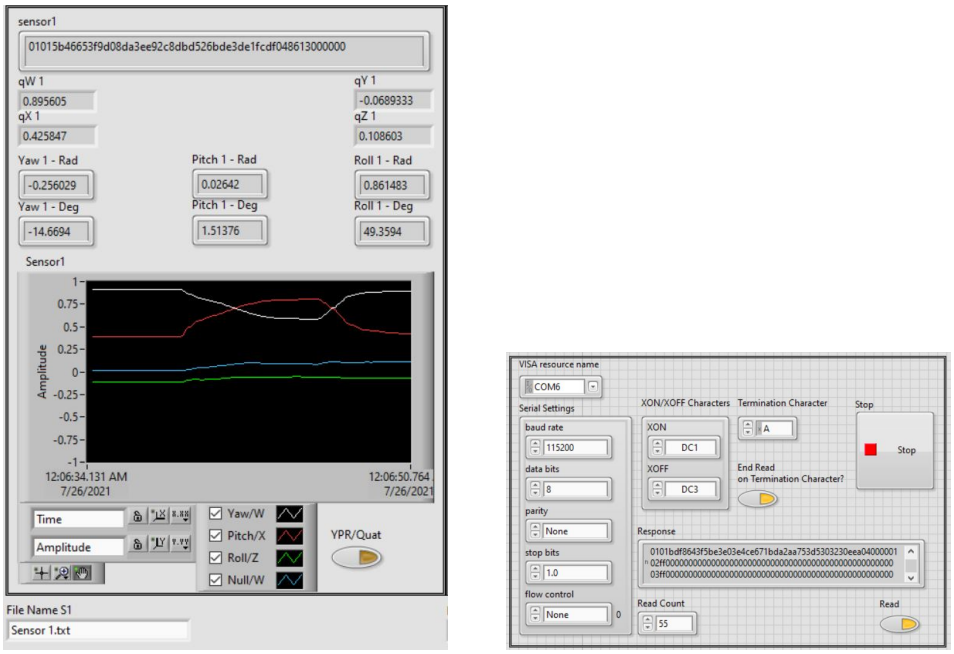


Figure 7.13: Flowchart of the LabView data representation application



(a) Representation pallet of the application - White line: W, Red line: X, Green line: Y, Blue line: Z

(b) Communication pallet of the application

Figure 7.14: GUI of the LabView data representation application

7.6.4 Data representation using Unity

The goal of this application is to do a 3D representation of the measurement values using a 3D humanoid model. To implement this application on the Windows operating system, it is necessary to use a graphical engine such as Unity or other similar applications. Unity is a cross-platform engine that is used for video game development and allows the developer to interact with graphical objects, also known as, game objects. Using the C# or Visual Basic programming languages, it is possible to interact with these objects, to make a rigid body attitude representation.

To accomplish this task, it is necessary to build a 3D human model including the body parts which are attached by joints and bones. In the game development community, *Rigging* is the process of assigning connection points to the 3D model

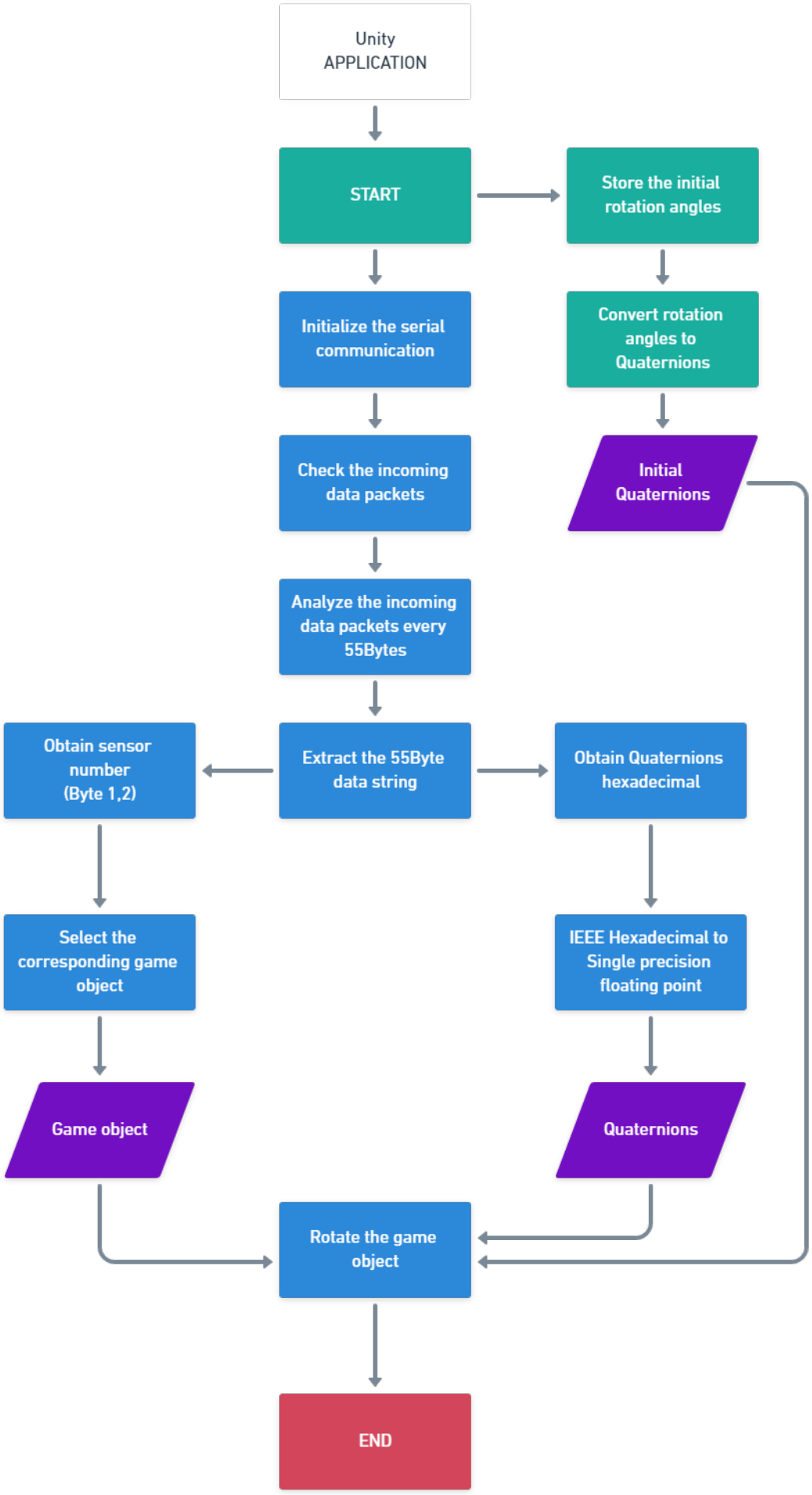
of a rigid body. A rigged 3D model is a hierarchical scheme of interconnected nodes (joints) using connectors (Bones) that can be rotated and transformed in the 3D space relatively. The rigging process is done using various 3rd party applications, and alternatively, it is possible to use pre-maid rigged humanoid characters from various GD communities such as Mixamo [38]. Mixamo is a part of the Adobe family, and it is a freely accessible 3D character and 3D animation repository.

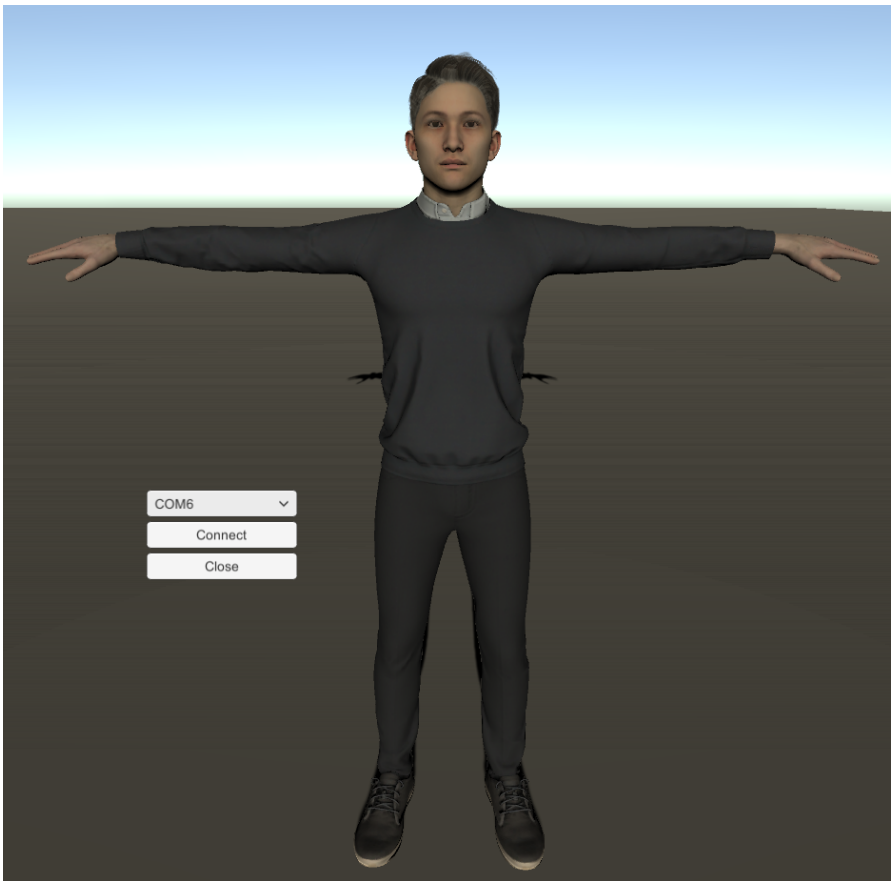
The logical procedure of application regarding the serial communication, and the data matrix interpretation is similar to the previous cases, in which the data matrix is obtained using the serial interface. By splitting the incoming data matrix, it is possible to extract the information related to sensor number, sensor data availability, attributes, and measurement data.

Furthermore, the rigged joints need to be referred to in the application as game objects with the order previously mentioned in Table 5.2. Then, the initial rotation angles values of the character need to be stored. The initial position is set by the 3D model and in this case, it is called *T-position*. The initial rotation angle of the body parts will be used later for relative rotation with respect to the current rotation angle value.

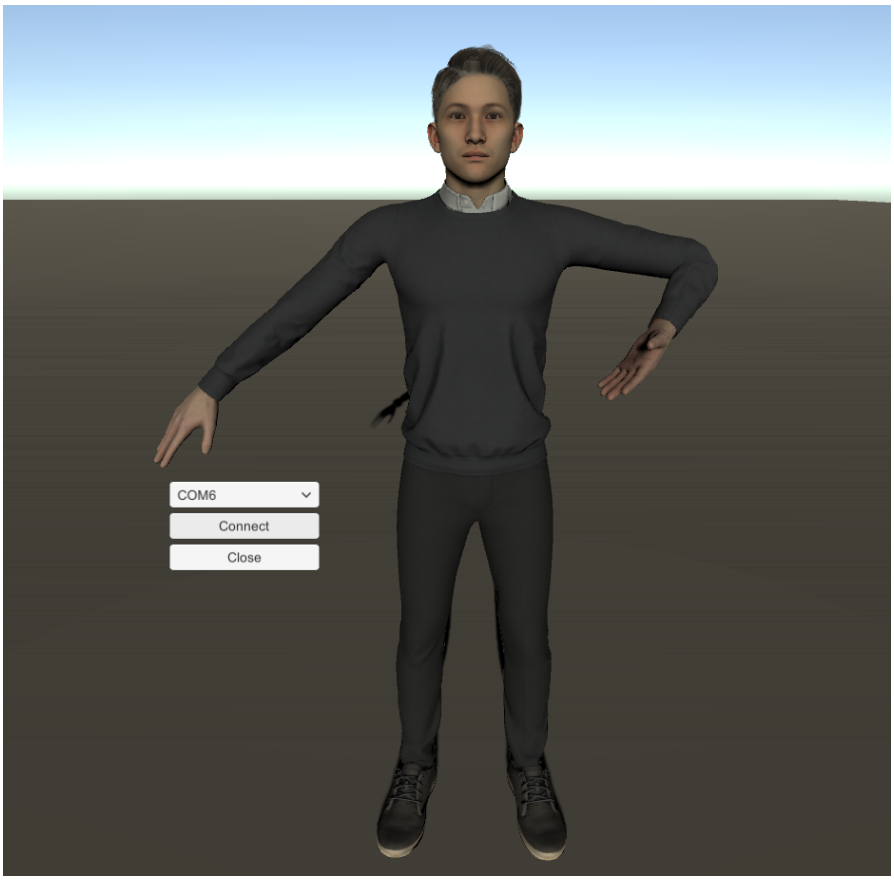
To use this application, it is required to place the wearable devices on the subject's body. The sensor number should correspond to Table 5.2. The subject should remain in T-position for 15 seconds so the initial gyroscope calibration process finishes. Once the calibration process is done, the LED2 one of the wearable devices will blink, which indicates the device is ready for the measurement process.

Figure 7.15 illustrates the logical flowchart of the Unity application, and Figure 7.16 shows the GUI of the Unity data representation application. In addition, the source codes of this windows application are available in the Github repository of the project [37].


Unity APPLICATION
START
Store the initial rotation angles
Initialize the serial communication
Convert rotation angles to Quaternions
Check the incoming data packets
Initial Quaternions
Analyze the incoming data packets every 55Bytes
Obtain sensor number (Byte 1,2)
Extract the 55Byte data string
Obtain Quaternions hexadecimal
Select the corresponding game object
IEEE Hexadecimal to Single precision floating point
Game object
Quaternions
Rotate the game object
END



(a) T-position figure



(b) Measurements using sensor #3, #4, and #5

Figure 7.16: GUI of the Unity data representation application

Chapter 8

Experiments

In this chapter, the necessary guides and considerations regarding the test procedure of the implemented solution are provided. The implemented solution is subjected to various experiments in order to verify the functionality of the system. These experiments have been conducted separately for each of the representation applications and the results are presented accordingly.

8.1 Programming the wearable devices

After setting up the firmware of the wearable devices, it is required to program the executable on the SoC. The sensor tags provide dedicated I/O pins exclusively for this purpose. In section, the step-by-step process and necessary requirements for programming the wearable devices are explained.

Requirements and setup

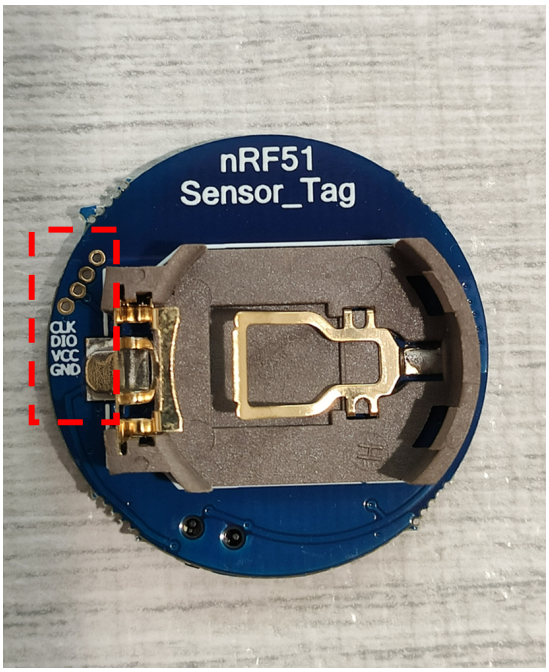
To program the executable file on the nRF51802 SoC, the first step is installing nRF Connect on a personal computer. This cross-platform application provides the necessary tools for developing, programming, and testing Nordic Semiconductor's Bluetooth and cellular IoT products. The application is freely available

by the manufacturer.

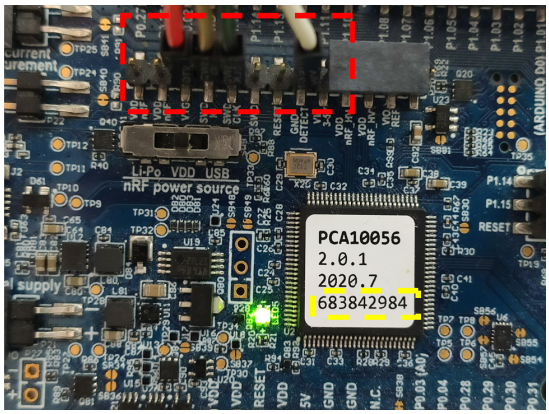
In addition, a Nordic semiconductor development kit such as nRF52840 is also required. This development kit is connected between the SoC and the personal computer and it performs the programming process through its I/O pins. In essence, the nRF51802 SoC needs to be connected to the development kit (using the I/O pins on both ends), and the development kit needs to be connected to the personal computer (using a male mini-USB/male USB cable). Table 8.1 indicates the connection between the development kit and the sensor tag, and Figure 8.1 illustrates the programming pins on each hardware component.

Table 8.1: Connection between the wearable device and the development kit

nRF51802 SoC (Sensor_tag)	**nRF52840DK (Development kit)**
CLK	SWD CLK
DIO	SWD I/O
VCC	VTG
GND	GND DETECT



(a) Dashed red square highlights the programming pins of the sensor tag



(b) Dashed red square shows the programming pins of the development kit

Figure 8.1: Programming pins of the hardware components

Since both of the hardware components must be turned on during the programming process, the final concern is providing the power source. The development kit is supplied using the personal computer during the process, therefore, no further considerations are required for this component. On the other hand, the sensor tag requires a dedicated power supply. This requirement is due to the fact that both hardware components require dedicated ground references. It is

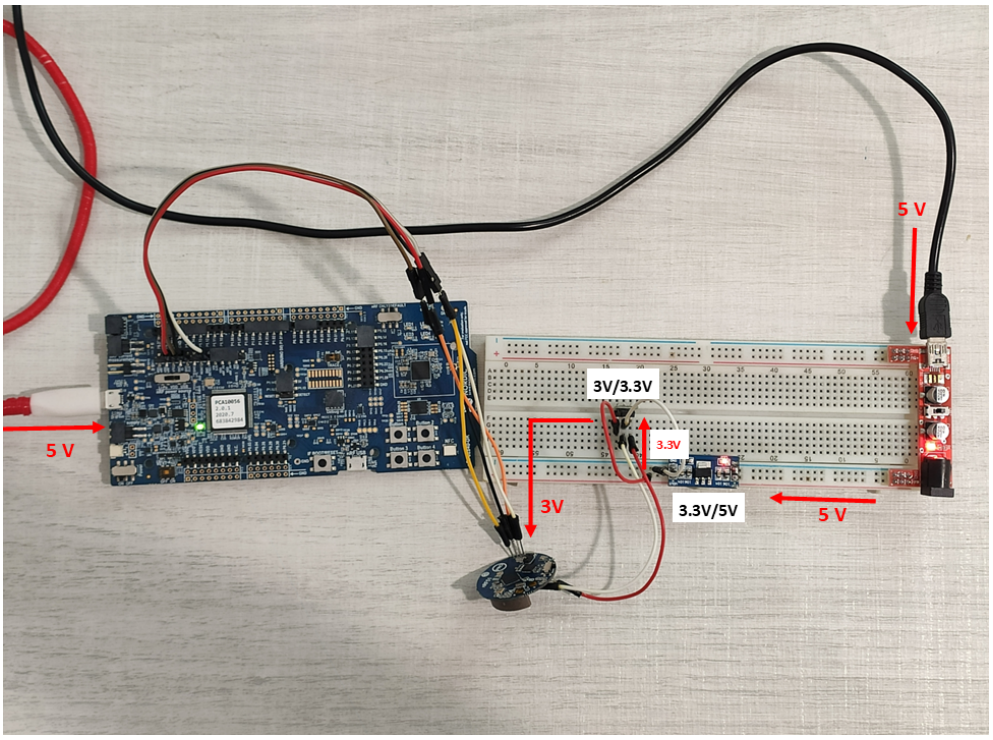


Figure 8.2: Connection between the hardware components

possible to power up the sensor tag using a 3V coin cell battery or other types of dedicated power supply with 3V output. Figure 8.2 illustrates the supplied devices and the connection between them.

Uploading the executable files

After setting up the hardware components, it is possible to upload the executable (.hex) file onto the sensor tag. For this purpose, the nRF Connect application is required. By navigating to the *Programmer* section of nRF Connect application, it is possible to choose the connected J-Link[1] devices to the COM ports of the personal computer. The name of the device corresponds to the on-board J-Link chip-set of the nRF52840DK (dashed yellow square in Figure).

After selecting the corresponding development kit from list of the devices, the programmer application should connect to the SoC of the wearable device. If the

[1]SEGGER J-Link is a USB powered JTAG debug probe.

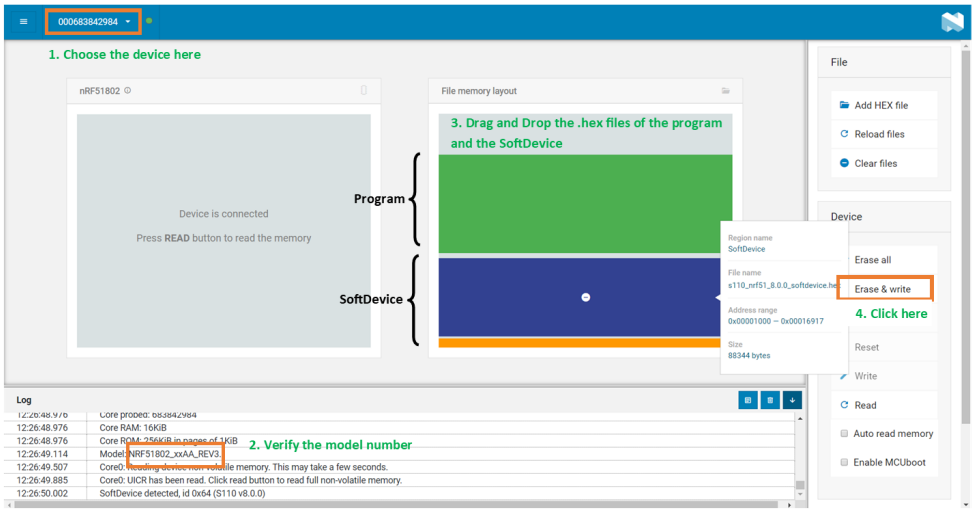


Figure 8.3: Environment and the programming steps using the programmer application

hardware components are properly connected, the debug terminal should indicate the SoC model as NRF51802_xxAA_REV3. If the model number is verified, it is possible to upload the executable files by addressing them into the *file memory layout* section and clicking on the *Erase & write* button. It is important to note that the executable files consist the .hex file which was resulted from compiling the peripheral program and the .hex file of the SoftDevice S110. Figure 8.3 illustrates environment of the programmer application.

8.2 Programming the development kit

The programming process of the development kit is a simple and robust task. There are generally two possible solutions to program the development kit. The first solution is by using the SEGGER Embedded studio (SEGGER-ES) IDE, and the second solution is to Drag-and-Drop the programming files into the flash space simply using the plug and play driver of the J-Link chip-set. The first method offers better features such as real-time debugging using the SEGGER-ES IDE, while the second method is easier to perform. These two methods are

explained in this section.

8.2.1 The first method: SEGGER-ES IDE

After compiling the code instructions of the solution, it is possible to simply upload the executable application using the SEGGER-ES IDE. This action can be performed done by navigating to the *Build* menu and selecting the *Build and Run* option (or press Ctrl+F5, Ctrl+T subsequently). Since this project requires uploading the SoftDevice S140 as well, solely uploading the executable file of the code instructions is not sufficient. Therefore, it is required to indicate the SoftDevice settings. The step-by-step guide for indicating the SoftDevice S140 settings are provided in the Nordic Semiconductor's info center [39].

8.2.2 The second method: Drag-and-Drop

By connecting the development kit to a personal computer using a Male micro-USB/Male USB cable, it is possible to drag-and-drop the executable file of the application as well as the executable file of the SoftDevice S140 into the available removable drive of the flash space. It is important to note that by using this method, the real-time debugging feature is not available. Therefore, although this method is sufficiently robust during the production phase, it is not suitable during the development process.

8.3 Experimental results

In this section, several experimental modules are performed in order to demonstrate the functionality and compatibility of this solution using the implemented windows applications. For MATLAB and Unity applications, three modules of experiment are performed and the graphical results are presented. The numerical

results from the LabView software have been used for metrological characterization of the system.

The aim of this solution is 3D reconstruction of human's physical activities, therefore, the results are visual and a better representation can be done in form of a video. Hence, a publicly available YouTube playlist is created for this solution, where the experiments are represented in form of videos [40]. Before proceeding to tests, the experimental modules followed by a brief description are provided below (For the list of sensors and their placement see Table 5.2).

- **Upper body experiment**: These experiments are aimed to capture the upper body activities by tracking sensor numbers #1 to #8.
- **Lower body experiment**: These experiments are aimed to capture lower body activities by tracking sensor numbers #9 to #15.
- **Full body experiment**: These experiments perform the full-body motion capturing process by employing all the 15 sensors.

8.3.1 MATLAB data representation application

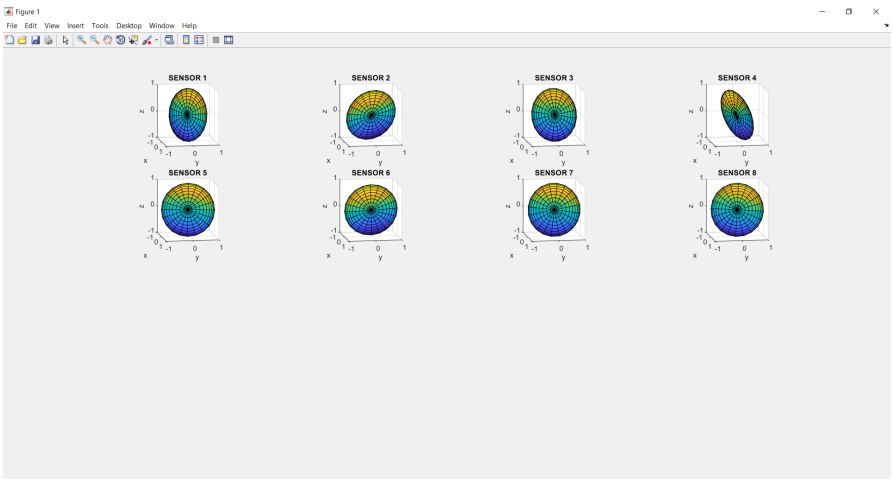
Considerations

By connecting the central receiver to a personal computer, it is possible to represent the incoming data from the wearable devices in MATLAB application using dedicated disk shaped objects. Each disk represents the orientation of one sensor during the measurement process in real-time. In order to correctly represent the orientation of the body, it is necessary to take into account the calibration phase of the sensors. The sensors should be calibrated before being used by placing them on horizontally on a flat surface. The sensor is calibrated and ready to use when the LED of the wearable device blinks with the color green. In addition, the direction of the Y-axis (green square in Figure 6.1) should point towards the personal computer's monitor.

Representation

Figure 8.4 to 8.9 illustrates the test results from the MATLAB application in 3 experimental modules, and each module contains 2 postures.

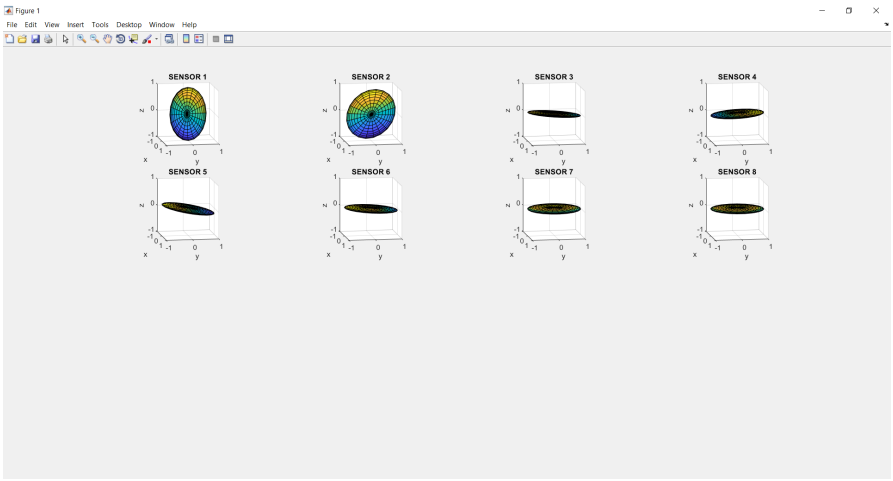
(a) test subject's posture (stand still)



(b) Representation using MATLAB (stand still)

Figure 8.4: Upper body experiments - MATLAB - Stand still

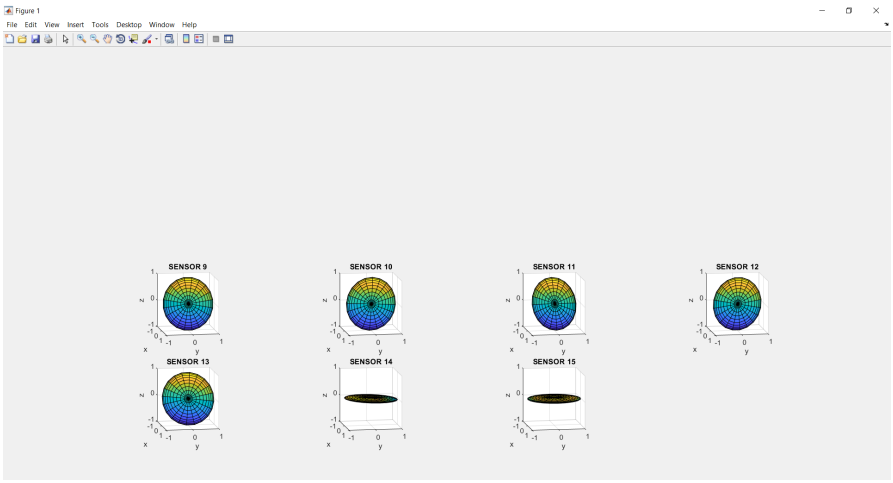
(a) test subject's posture (T-pose)



(b) Representation using MATLAB (T-pose)

Figure 8.5: Upper body experiments - MATLAB - T-pose

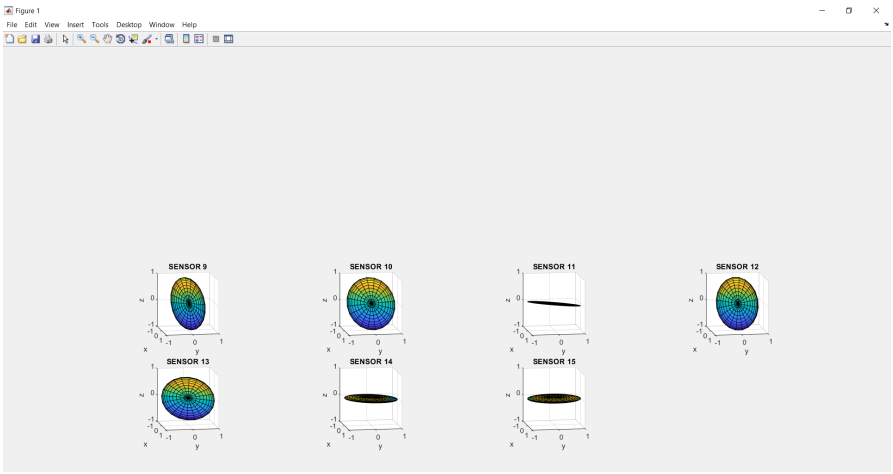
(a) test subject's posture (stand still)



(b) Representation using MATLAB (stand still)

Figure 8.6: Lower body experiments - MATLAB - Stand still

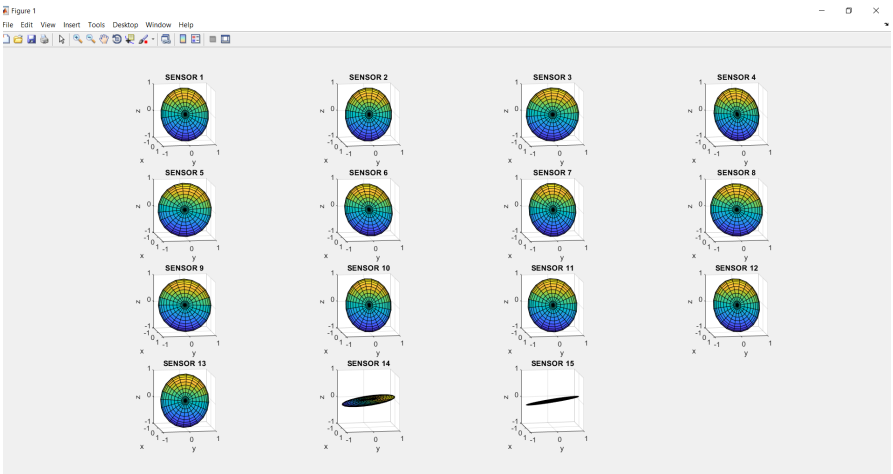
(a) test subject's posture (arbitrary posture)



(b) Representation using MATLAB (arbitrary posture)

Figure 8.7: Lower body experiments - MATLAB - Arbitrary posture

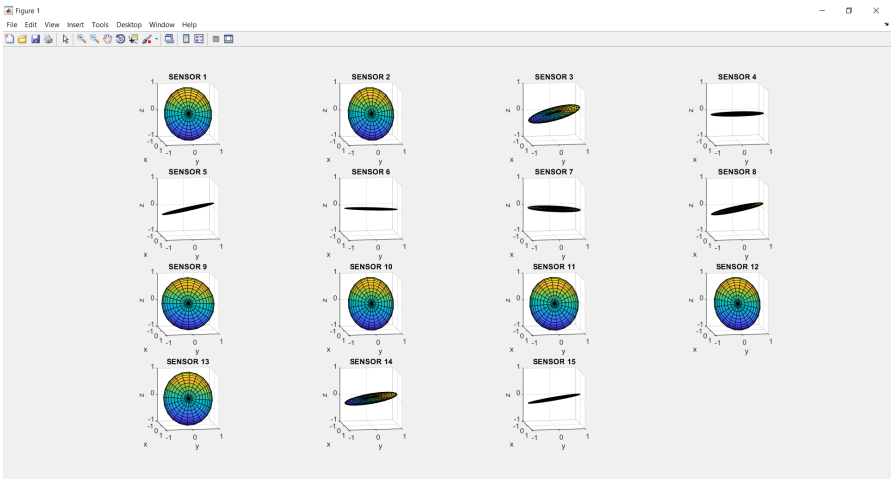
(a) test subject's posture (stand still)



(b) Representation using MATLAB (stand still)

Figure 8.8: Full body experiments - MATLAB - Stand still

(a) test subject's posture (T-pose)



(b) Representation using MATLAB (T-pose)

Figure 8.9: Full body experiments - MATLAB - T-pose

8.3.2 Unity data representation application

The Unity application is used to graphically represent the incoming data from the sensors on a humanoid 3D model. Figure 8.10 to 8.12 represent the upper experiments using Unity application.

Considerations

In order to proceed with the experiments using Unity software, it is necessary to take into account few points. The sensor placement on the body must be done in a certain way, otherwise, the 3D reconstruction of the physical activity might not corresponds to the actual physical activity.

On the sensor tags, a visual sign indicates the direction of the Y-axis(green square in Figure 6.1), this sign has been considered as an indicator to place the sensors. Table 8.2 summarises the correct direction of each sensor with respect to the Y-axis indicator.

Table 8.2: Correct direction of the Y-axis indicator

Sensor number	**Direction of the indicator**
1 - Forehead	Facing up
2 - Chest	Facing up, pointing to sensor 1
3 - Left arm	Facing towards the left hand's fingers
4 - Right arm	Facing towards the right hand's fingers
5- Left wrist	Facing towards the left hand's fingers
6 - Right wrist	Facing towards the right hand's fingers
7 - Left hand	Facing towards the left hand's fingers
8 - Right hand	Facing towards the right hand's fingers
9 - Sacral	Facing up, pointing to sensor 2
10 - Left knee	Facing towards the left foot's fingers
11 - Right knee	Facing towards the right foot's fingers
12 - Left ankle	Facing towards the left foot's fingers

Table continued from previous page

Sensor number	Direction of the indicator
13 - Right ankle	Facing towards the right foot's fingers
14 - Left foot	Facing towards the left foot's fingers
15 - Right foot	Facing towards the right foot's fingers

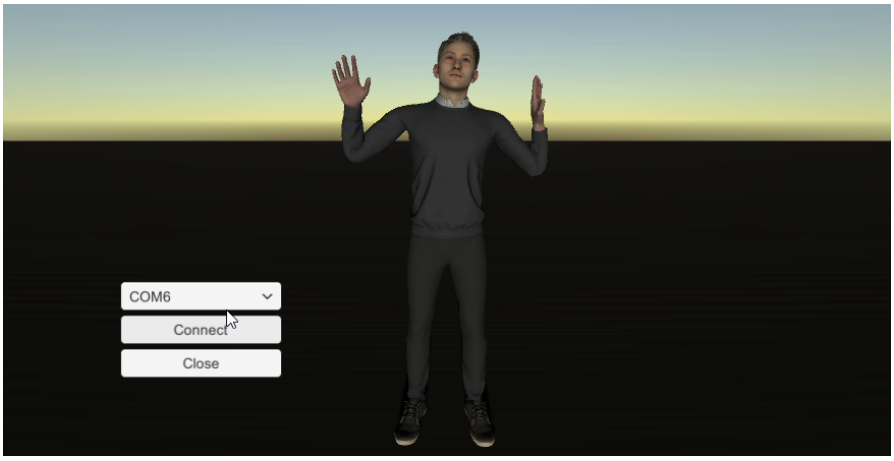
In addition, it is important to remain in T-Pose posture (Straight standing legs and T-shaped arms) for the whole calibration process, otherwise, visual drift errors due to incomplete calibration of the inertial sensors might be witnessed.

Representation

Figure 8.10 to 8.12 demonstrate the experimental results using Unity software in 3 experimental modules.

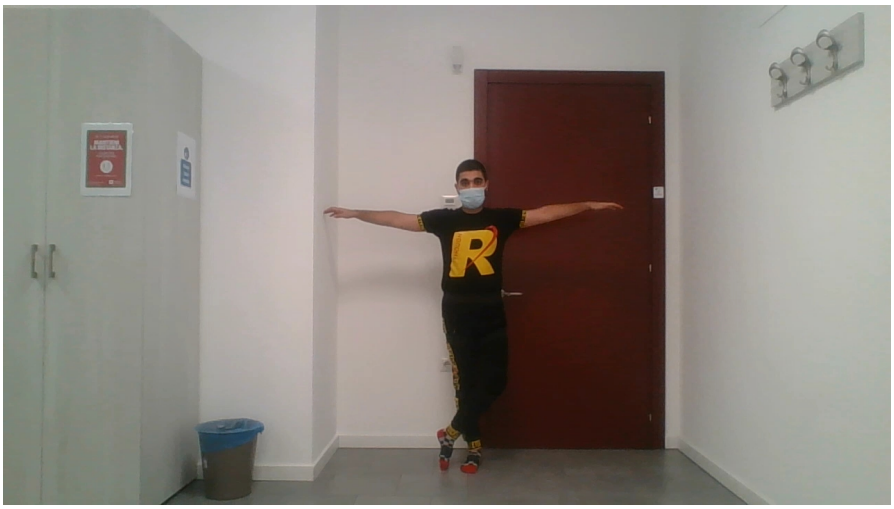
It is important to note that the model's initial posture is the T-Pose posture. During the experiments, all body parts of the humanoid model are visible, but only those parts which are subjected to the experiment will move. Let us consider the upper body experimental module, during this experiment, a complete humanoid model is represented, but only the body parts corresponding to sensor numbers #1 to #8 will move (left and right hands, left and right wrists, left and right arms, chest, and head).

(a) Test subject's posture

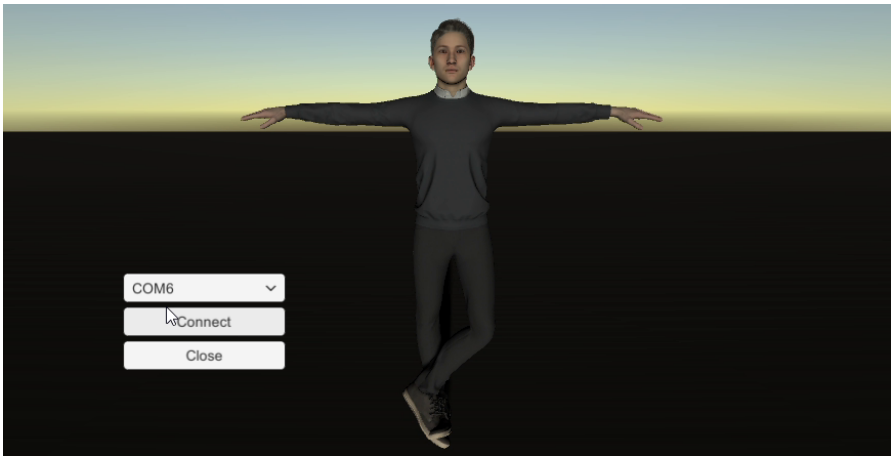


(b) Representation using Unity

Figure 8.10: Upper body experiments - Unity



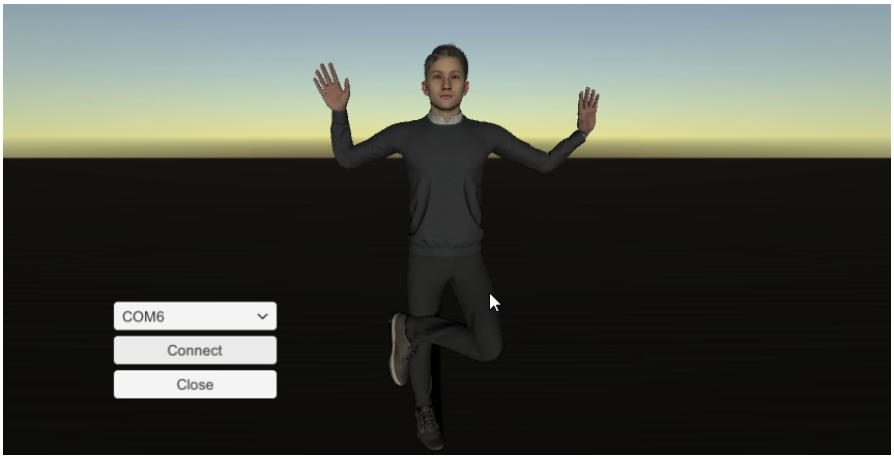
(a) Test subject's posture



(b) Representation using Unity

Figure 8.11: Lower body experiments - Unity

(a) Test subject's posture



(b) Representation using Unity

Figure 8.12: Full body experiments - Unity

8.4 Metrological characterization

Reporting measurement values of a physical quantity requires quantitative implications in order to describe the quality of the measurement procedure. This concept is necessary for reliability assessment of a given piece of data. Therefore, defining a generally accepted parameter for evaluating and characterizing the measurement quality is essential. In many applications, this parameter, also known as *measurement uncertainty*, is expected to provide an interval with a con-

fidence level, in which a large fraction of the measurement values are encompassed by it.

The method for expressing measurement uncertainty should be universal (applicable to all types of data), internally consistent (derivable from the contributing components and unrestrained from the composition or decomposition of expressions), and transferable (the results should be useful for uncertainty evaluation of other measurements).

The basic terminology of measurement systems and related parameters are provided by joint committee for guides in metrology (JCGM) [41]. In this section, based on the cited document, a preliminary attempt has been done to metrologically characterize the measuring units (Wearable devices).

8.4.1 The theory

The measurement uncertainty, which describes the dispersion of the measured values attributed to a measurand, is composed from several components. Some of these components are calculated using statistical methods based on series of measurements, while, the other can be calculated using other presumed probability distribution based on a pool of data (Manufacturer specifications, historical knowledge, experience, etc.). Regardless of the used method, in both cases, these components are characterized by standard deviation.

The uncertainty that is represented as a standard deviation is denoted as standard uncertainty. Generally, there are two main ways to evaluate the standard uncertainty which are provided below:

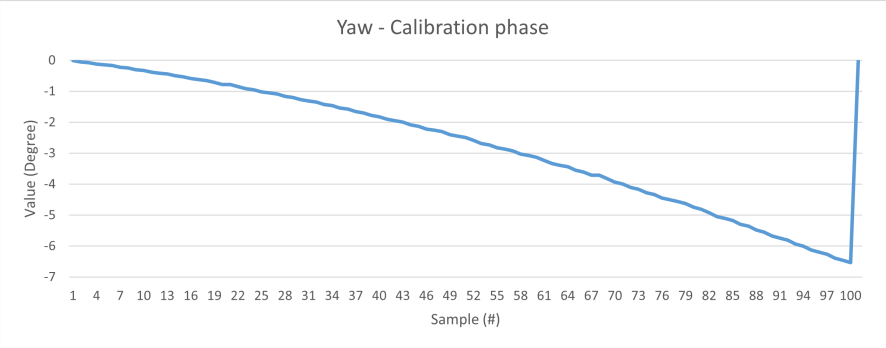
- **Type A evaluation**: An evaluation method using statistical analysis of observed values.
- **Type B evaluation**: An evaluation method based on methods other than statistical analysis (historical data, experience, etc).

When the measureand is affected from several input quantities, the ***combined standard uncertainty*** method is used to combine the evaluations. If there is no correlation between the input quantities, the combined standard uncertainty is equal to square root of sum of all variances (or covariances) of the effective input quantities. Furthermore, it is possible to represent the obtained uncertainty by means of the ***expanded uncertainty***. This parameter defines an interval that a large fraction of the measured values (estimated measurements) are covered during the measurement procedure.

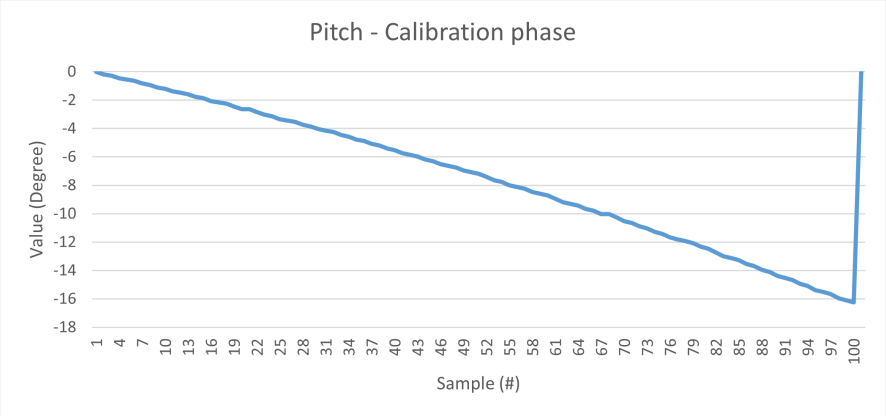
8.4.2 The procedure and results

In this section, by performing an experiment, an attempt to evaluate the Type A standard uncertainty has been done. This preliminary evaluation has been done in order to roughly estimate the standard uncertainty of a single sensor (Sensor #2 corresponding to chest) throughout the whole measurement process. This is a first simple metrological characterization. The future aim is to metrologically characterize the system by comparing it with the Gold Standard, which is present at Politecnico di Milano at Divieti laboratory.

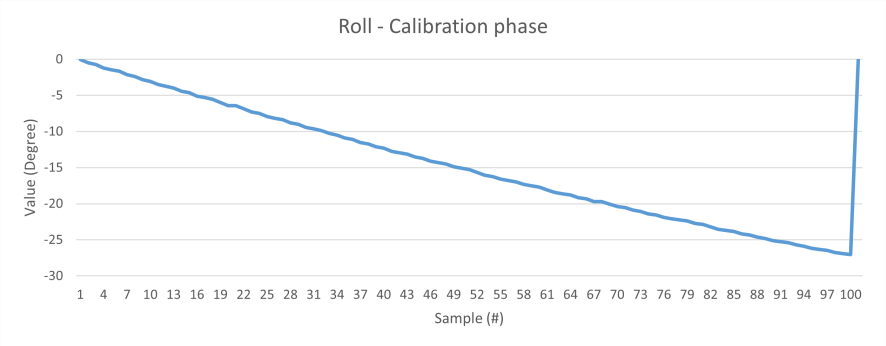
The experiment has been performed by connecting a single sensor tag to the central receiver via BLE, and connecting the central receiver to a personal computer using serial communication. After setting up the devices, the sensor tag was located in a fixed position (On a piece of sponge in order to damp environmental vibrations). Then, the sensor tag was turned on and the calibration process started, which took approximately about 10 seconds. Moreover, the NI LabView application have been used in order to record the incoming data in a text-plain file. Finally, the text-plain file has been converted into an Excel worksheet in order to perform the statistical analysis. Figure 8.13 illustrates the variations of the Euler angles during the calibration phase.



(a) Rotation around Yaw axis during calibration



(b) Rotation around Pitch axis during calibration



(c) Rotation around Roll axis during calibration

Figure 8.13: Variations of the Euler angles during the calibration process

The experiment was done for about 9 minutes, at room temperature (23 ºC) in order to avoid clock frequency variations (See sec 5.5 of [25]), and the PCB Mounting and Cross-Axis sensitivity has been neglected (See sec 11.4.5 of [25]). The first 5000 samples were considered for the analysis (500 seconds of data

acquisition), in which the first 197 samples were before finishing the calibration process, therefore, there were only subjected to visual representation and the uncertainty assessment. The next 100 samples were also discarded to take into account the warm-up time of the measuring unit. Hence, the samples between 297 to 5000 has been used for this analysis.

According to JCGM, the best estimated expected value of a quantity $'q'$ that varies randomly can be obtained by calculating the arithmetic mean of the variable throughout n observations using Eq. 8.1

$$\overline{q} = \frac{1}{n}\sum_{k=1}^{n} q_k \tag{8.1}$$

Where $\overline{q}$ is the arithmetic mean and q_k is the k^{th} observation of quantity q.

Since the measured values differ randomly due to the variations of the influence quantities, it is possible to calculate the experimental variance of the observation. This parameter indicates the variance value for the probability distribution of quantity q using Eq. 8.2.

$$s^2(q_k) = \frac{1}{n-1}\sum_{j=1}^{n} (q_j - \overline{q})^2 \tag{8.2}$$

The positive square root of the experimental variance indicates the experimental standard deviation. This parameter defines the dispersion of the measurement values with respect to their arithmetic mean.

It is also possible to obtain the variance of the mean $s^2(\overline{q})$, by dividing the experimental variance of the observations with number of samples.

$$s^2(\overline{q}) = \frac{s^2(q_k)}{n} \tag{8.3}$$

By calculating the positive square root of the $s^2(\overline{q})$, it is possible to obtain the experimental standard deviation of the mean $s(\overline{q})$, which indicates the quality of $\overline{q}$ estimating the expected value of q. This value evaluates whether if the arithmetic mean is the best estimator of the expected value of q.

Performing the mentioned during the experiment resulted the following values in Table 8.3 for our solution:

Table 8.3: Type A evaluation of standard uncertainty

Induced rotation[o]	**Y**	0	**Experimental SD** $s(q_k)$	**Y**	0.05983
	P	0		**P**	0.03077
	R	0		**R**	0.02622
Best estimate $\overline{q}$	**Y**	0.18821	**Experimental variance of the mean** $s^2(\overline{q})$	**Y**	7.6145e(-7)
	P	-0.09228		**P**	2.0145e(-7)
	R	-0.06463		**R**	1.4622e(-7)
Experimental variance $s^2(q_k)$	**Y**	0.00358	**Experimental SD of the mean** $s(\overline{q})$	**Y**	0.00087
	P	0.00094		**P**	0.00044
	R	0.00068		**R**	0.00038

It is also worth mentioning that the errors due to the measurement equipment, may propagate throughout the whole process. Let us consider Figure 8.14, which represents the sensor and a possible rotation on an angle theta. Assuming the distance between sensor #6 and #8, the R represents the length of the right forearm. Therefore, for rotations on the angle theta, chord C represents the horizontal displacement of the right hand's finger tips; and height h demonstrates the vertical displacement of the finger tips. According to Figure 8.14, the following equations can be written:

$$c = 2R \sin \frac{\theta}{2} \tag{8.4}$$

$$h = R(1 - \cos \frac{\theta}{2}) \tag{8.5}$$

where R is the radius of the circle (in this case, length of the subject's forearm), and θ is the rotation angle).

Considering Eq. 8.4 and 8.5, and assuming the length of the test subject's arm as 32cm, one degree error in measurement of the angle θ might result in 0.558cm horizontal, and 8.726×10^{-3}cm vertical displacement during the 3D reconstruction process. The significance of this error depends on the application of the solution. As for entertainment purposes, such as game development, the results of the Type A evaluation indicate that the solution requires improvement. While, for sports training purposes the indicated results might be within the acceptable margin.

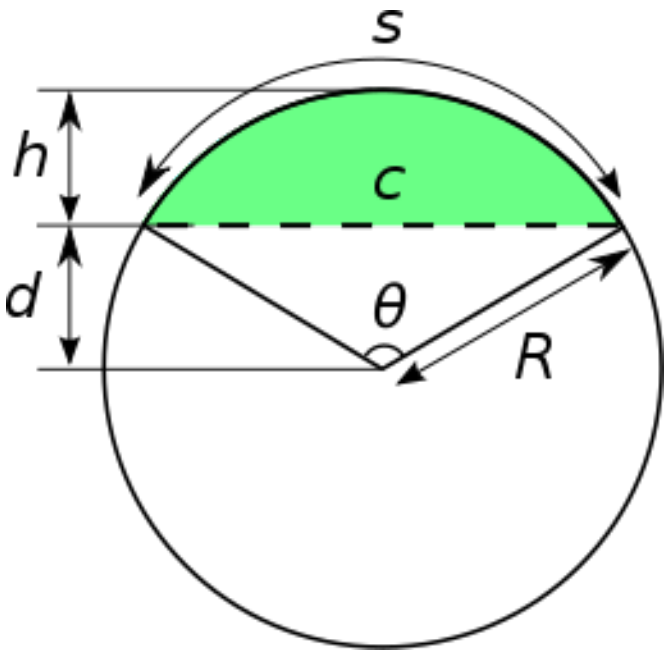


Figure 8.14: Chord and height of a circle's segment

Chapter 9

Conclusion and future scope

9.1 Conclusion

Considering the limitations of machine vision technologies in motion capturing, this project aimed to implement a wireless, real-time and wearable motion capturing solution using inertial measurement units. For this purpose, first, investigations were carried out to demonstrate the spectrum of motion-capture devices. Studies showed that sensor-based motion capture offers a broad range of applications and could be utilized in ambient environments that are not ideal for machine vision motion capturing. These studies also demonstrated that the design requirements of motion capturing systems are correlated with each other and are highly dependent on the application area. Therefore, it is crucial to precisely specify the required features and specifications of the system before implementing these solutions.

To achieve this objective, the working principle of inertial measurement units and the theoretical concepts regarding attitude representation based on the angular acceleration and angular velocity were explained. Two possible methods of attitude representation namely Euler angles, and Quaternions; and the challenges of employing them were discussed. Additionally, the problem of absolute

attitude representation solely based on rotation angles was mentioned. This problem resulted in implementing attitude estimation procedures instead of attitude representation by using Kalman and Complementary filtering.

Moreover, to emphasize the importance of the design factors, a chapter was dedicated to the scientific terminology and the implementation scenario of wearable motion capturing devices. Various design parameters that directly affect the system performance were mentioned and the design specifications of the suggested solution were discussed.

Regarding hardware design as the first step of the implementation process, the procedure for choosing hardware materials and the specifications of each hardware component have been provided. Due to the requirements of the project, a sensor module, named *Sensor_tag*, was selected. This component provided us with an MPU6050 3-axis accelerometer, a 3-axis gyroscope IMU and an nRF51802 SoC as the processing unit. The SoC enabled us to use the Bluetooth® Low Energy wireless communication protocol which fully satisfied the design requirements. In addition, the necessity of employing rechargeable batteries along with wearable devices was highlighted and a new design scheme in the form of a printed circuit board was implemented to provide this feature. As for the software design, the terminology of the BLE communication protocol has been described. This description provided the basic working principles and necessary considerations for implementing an application using BLE. Also, we collaborated with the electronics and telecommunication department of Politecnico di Torino for building the central receiver and the embedded firmware of the wearable devices. The working roles and the detailed application of each component throughout this system is demonstrated using flowcharts and parameter specifications. Then the considerations for data interpretation and the working principle of each software was mentioned. Eventually, multiple software applications for representing data were described and implemented for personal computers.

Finally, the experimental results have demonstrated the functionality of the

system. These experiments were done to represent the measurement data of each software application. The final assessment of the data revealed that although the system performs sufficiently for a single sensor, using a network of multiple sensors may present obstacles that are worth investigating. These obstacles became more challenging as the number of sensors in the network increased. This problem was diagnosed to be due to the communication protocol and the requirement for fast data transmission. In addition, this study attempted to metrologically characterize the measuring units in order to describe the standard uncertainty of the measurement systems.

9.2 Future scope

Consistent product design is a major step in the future development of this solution. The introduced solution is currently at the development stage, and it is considered as a prototype. The functionality and performance of this system can be further improved by employing professional product design methods. The graphical user interface of the representation software applications can also be enhanced to improve user experience.

Moreover, the implementation scenario of the system can be further developed by discarding the central receiver development kit and using a device that is already equipped with a BLE receiver. Removing a hardware component from the whole system could result in bypassing the hardware and software errors of that particular component and the errors of the communication protocol. This adjustment could in turn enhance the functionality, stability, and performance of the measurement system. However, these modifications require a thorough software development process on the desired representation platform.

Additionally, the accuracy and reliability of the system can be further improved by employing a magnetometer along with the 3-axis gyroscope. Although the gyroscope drift errors were sufficiently bypassed using software-level digital

motion processing methods, using a magnetometer can be effectively beneficial for gyroscope measurement corrections. Of course, this component introduces more challenges regarding magnetic isolation of the system when it is used next to ferromagnetic materials.

As previously mentioned, the implemented solution is a real-time measurement system that represents rotation angles based on angular acceleration and angular velocity. The rotation angles can be further analyzed using data processing methods for human activity recognition (HAR). This application is widely used for medical research and diagnostic procedures such as gait analysis and core stability assessment. It is therefore possible to implement HAR by using suitable data analysis methods such as Bayesian classifiers and deep neural networks.

Finally, the metrological characterization of the whole system can be further researched. This process requires modeling and evaluating the uncertainty that is introduced by the multiple measuring units, the acquisition process, the mathematical operations, the communication protocol, and the noise interference at each stage of the measurement operation. Additionally, the implemented solution can be subjected to further evaluation by comparing the measurement results with the gold standard.

Appendices

Appendix A

Code of the central receiver

This is the main code of the central receiver's embedded application (main.c). The full program is available in the GitHub repository of the project [37].

```
#include "app_timer.h"
#include "ble.h"
#include "ble_conn_params.h"
#include "ble_conn_state.h"
#include "ble_db_discovery.h"
#include "ble_nus_c_cust.h"
#include "boards.h"
#include "nordic_common.h"
#include "nrf.h"
#include "nrf_ble_scan.h"
#include "nrf_delay.h"
#include "nrf_log.h"
#include "nrf_log_ctrl.h"
#include "nrf_log_default_backends.h"
#include "nrf_pwr_mgmt.h"
#include "nrf_sdh.h"
```

```
17 #include "nrf_sdh_ble.h"
18 #include <stdint.h>
19 #include <stdio.h>
20 #include <string.h>
21 #define APP_BLE_CONN_CFG_TAG 1
22 #define APP_BLE_OBSERVER_PRIO 3
23 #define NUS_SERVICE_UUID_TYPE BLE_UUID_TYPE_VENDOR_BEGIN
24 #define MAX_SENSORS NRF_SDH_BLE_CENTRAL_LINK_COUNT
25 #define MAX_DATA_LEN 60
26 #define UUID16_SIZE 2
27 #define UUID32_SIZE 4
28 #define UUID128_SIZE 16
29 #define DELTAT (3276 * 1 + 5)
30 BLE_NUS_C_ARRAY_DEF(m_nus_c,
31 NRF_SDH_BLE_CENTRAL_LINK_COUNT);
32 BLE_DB_DISCOVERY_ARRAY_DEF(m_db_disc,
33 NRF_SDH_BLE_CENTRAL_LINK_COUNT);
34 NRF_BLE_SCAN_DEF(m_scan);
35
36 char CursorLine[MAX_DATA_LEN];
37 char DataMatrix[MAX_SENSORS][MAX_DATA_LEN];
38 char LastDataCounter[MAX_SENSORS];
39
40 static char const m_target_periph_name[] = "homeTag";
41 static uint8_t m_scan_buffer_data[BLE_GAP_SCAN_BUFFER_MIN];
42 static ble_data_t m_scan_buffer =
43     {
44         m_scan_buffer_data,
45         BLE_GAP_SCAN_BUFFER_MIN};
```

```
46
47 static const ble_uuid_t m_nus_uuid =
48     {
49         .uuid = BLE_UUID_NUS_SERVICE,
50         .type = NUS_SERVICE_UUID_TYPE};
51
52 static const ble_gap_conn_params_t m_connection_param =
53     {
54         NRF_BLE_SCAN_MIN_CONNECTION_INTERVAL, // Minimum connection
55         NRF_BLE_SCAN_MAX_CONNECTION_INTERVAL, // Maximum connection
56         0,                                    // Slave latency
57         NRF_BLE_SCAN_SUPERVISION_TIMEOUT      // Supervision time-out
58 };
59 static bool is_uuid_present(const ble_uuid_t *p_target_uuid,
60     const ble_gap_evt_adv_report_t *p_adv_report) {
61   uint32_t err_code;
62   uint32_t index = 0;
63   uint8_t *p_data = (uint8_t *)p_adv_report->data.p_data;
64   ble_uuid_t extracted_uuid;
65
66   while (index < p_adv_report->data.len) {
67     uint8_t field_length = p_data[index];
68     uint8_t field_type = p_data[index + 1];
69
70     if ((field_type ==
71     BLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_MORE_AVAILABLE) ||
72     (field_type == BLE_GAP_AD_TYPE_16BIT_SERVICE_UUID_COMPLETE)) {
73       for (uint32_t u_index = 0; u_index <
74       (field_length / UUID16_SIZE); u_index++) {
```

```
75          err_code = sd_ble_uuid_decode(UUID16_SIZE,
76              &p_data[u_index * UUID16_SIZE + index + 2],
77              &extracted_uuid);
78          if (err_code == NRF_SUCCESS) {
79            if ((extracted_uuid.uuid == p_target_uuid->uuid) &&
80            (extracted_uuid.type == p_target_uuid->type)) {
81              return true;
82            }
83          }
84        }
85      }
86
87      else if ((field_type ==
88      BLE_GAP_AD_TYPE_32BIT_SERVICE_UUID_MORE_AVAILABLE) ||
89      (field_type == BLE_GAP_AD_TYPE_32BIT_SERVICE_UUID_COMPLETE)) {
90        for (uint32_t u_index = 0; u_index <
91        (field_length / UUID32_SIZE); u_index++) {
92          err_code = sd_ble_uuid_decode(UUID16_SIZE,
93              &p_data[u_index * UUID32_SIZE + index + 2],
94              &extracted_uuid);
95          if (err_code == NRF_SUCCESS) {
96            if ((extracted_uuid.uuid == p_target_uuid->uuid) &&
97            (extracted_uuid.type == p_target_uuid->type)) {
98              return true;
99            }
100         }
101       }
102     }
103
```

```
104     else if ((field_type ==
105     BLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_MORE_AVAILABLE) ||
106     (field_type == BLE_GAP_AD_TYPE_128BIT_SERVICE_UUID_COMPLETE)) {
107       err_code = sd_ble_uuid_decode(UUID128_SIZE,
108           &p_data[index + 2],
109           &extracted_uuid);
110       if (err_code == NRF_SUCCESS) {
111         if ((extracted_uuid.uuid == p_target_uuid->uuid) &&
112          (extracted_uuid.type == p_target_uuid->type)) {
113           return true;
114         }
115       }
116     }
117     index += field_length + 1;
118   }
119   return false;
120 }
121 void assert_nrf_callback(uint16_t line_num,
122 const uint8_t *p_file_name) {
123   app_error_handler(0xDEADBEEF, line_num, p_file_name);
124 }
125 static ble_gap_scan_params_t const m_scan_param =
126     {
127         .active = 0x01,
128         .interval = NRF_BLE_SCAN_SCAN_INTERVAL,
129         .window = NRF_BLE_SCAN_SCAN_WINDOW,
130         .timeout = 0x00,
131 };
132 ////////////////////////////////////////////////////////////
```

```
133 ////////////////APPLICATION FUNCTIONS/////////////////////
134 ////////////////////////////////////////////////////////////
135 void InsertNewSensorDataIntoMatrix(int m, char *p, int L) {
136
137   int IDX = m;
138   if ((IDX < 1) || (IDX > 20))
139     return;
140
141   IDX--;
142
143   for (int n = 1; n < L; n++)
144     DataMatrix[IDX][n] = p[n];
145   DataMatrix[IDX][0]++;
146   if (!DataMatrix[IDX][0])
147     DataMatrix[IDX][0]++; // Avoid value zero
148 }
149
150 void Init_Sensor_Data(void) {
151 // First byte represents the counter. This is incremented each
152 //time a new packet is received. Is used at main level to
153 //check read coherence and verify fresh packets
154 // Zero value is skipped since zero mean no packet at all
155   for (int n = 0; n < MAX_SENSORS; n++) {
156     DataMatrix[n][0] = 0;
157     LastDataCounter[n] = 0;
158
159     for (int m = 1; m < MAX_DATA_LEN; m++)
160       DataMatrix[n][m] = 0;
161   }
```

```
162 }
163
164 void CoherentReading(int n) {
165
166   do {
167     for (int m = 0; m < MAX_DATA_LEN; m++)
168       CursorLine[m] = DataMatrix[n][m];
169   } while (CursorLine[0] != DataMatrix[n][0]);
170 }
171 ////////////////////////////////////////////////////////////
172 ////////////////END OF APPLICATION FUNCTIONS/////////////
173 ////////////////////////////////////////////////////////////
174
175 ///////////////////////DEBUG SECTION//////////////////////
176 ///////////////Active/Deactive from SDK_CONFIG/////////////
177 ////////////////////////////////////////////////////////////
178 #ifndef DEBUG_application
179 #include "app_uart.h"
180 #if defined(UART_PRESENT)
181 #include "nrf_uart.h"
182 #endif
183 #if defined(UARTE_PRESENT)
184 #include "nrf_uarte.h"
185 #endif
186 //#define UART_TX_BUF_SIZE 128 /**< UART TX buffer size. */
187 //#define UART_RX_BUF_SIZE 128 /**< UART RX buffer size. */
188 #define UART_TX_BUF_SIZE 16384 /**< UART TX buffer size. */
189 #define UART_RX_BUF_SIZE 256 /**< UART RX buffer size. */
190 void uart_error_handle(app_uart_evt_t *p_event) {
```

```
191   if (p_event->evt_type == APP_UART_COMMUNICATION_ERROR) {
192     APP_ERROR_HANDLER(p_event->data.error_communication);
193   } else if (p_event->evt_type == APP_UART_FIFO_ERROR) {
194     APP_ERROR_HANDLER(p_event->data.error_code);
195   }
196 }
197 #define UART_HWFC APP_UART_FLOW_CONTROL_DISABLED
198 void Uart_define() {
199   uint32_t err_code;
200   const app_uart_comm_params_t comm_params =
201       {
202           RX_PIN_NUMBER,
203           TX_PIN_NUMBER,
204           RTS_PIN_NUMBER,
205           CTS_PIN_NUMBER,
206           UART_HWFC,
207           false,
208           NRF_UARTE_BAUDRATE_230400};
209   APP_UART_FIFO_INIT(&comm_params,
210       UART_RX_BUF_SIZE,
211       UART_TX_BUF_SIZE,
212       uart_error_handle,
213       APP_IRQ_PRIORITY_LOWEST,
214       err_code);
215 }
216 #endif
217 ////////////////////END OF DEBUG SECTION/////////////////
218 ////////////////////////////////////////////////////////
219 ////////////////////////////////////////////////////////
```

```
220 static void scan_evt_handler(scan_evt_t const *p_scan_evt) {
221 }
222 /**@brief Function for initializing the
223 scanning and setting the filters.
224  */
225 static void scan_init(void) {
226   if (ble_conn_state_central_conn_count() !=
227   NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
228     ret_code_t err_code;
229     nrf_ble_scan_init_t init_scan;
230     memset(&init_scan, 0, sizeof(init_scan));
231     init_scan.connect_if_match = true;
232     init_scan.conn_cfg_tag = APP_BLE_CONN_CFG_TAG;
233     err_code = nrf_ble_scan_init(&m_scan,
234     &init_scan, scan_evt_handler);
235     APP_ERROR_CHECK(err_code);
236     err_code = nrf_ble_scan_filter_set(&m_scan,
237      SCAN_NAME_FILTER, m_target_periph_name);
238     APP_ERROR_CHECK(err_code);
239     err_code = nrf_ble_scan_filters_enable(&m_scan,
240     NRF_BLE_SCAN_NAME_FILTER, false);
241     APP_ERROR_CHECK(err_code);
242   } else {
243     return;
244   }
245 }
246
247 /**@brief Function for starting scanning. */
248 static void scan_start(void) {
```

```
249   if (ble_conn_state_central_conn_count() !=
250   NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
251     ret_code_t ret;
252     ret = nrf_ble_scan_start(&m_scan);
253     APP_ERROR_CHECK(ret);
254
255   } else {
256     return;
257   }
258 }
259
260 /**@brief Handles events coming from the
261 LED Button central module.
262  *
263  * @param[in] p_lbs_c     The instance of LBS_C
264  that triggered the event.
265  * @param[in] p_lbs_c_evt The LBS_C event.
266  */
267 static void ble_nus_c_evt_handler(ble_nus_c_t *p_ble_nus_c,
268 ble_nus_c_evt_t const *p_ble_nus_evt) {
269   switch (p_ble_nus_evt->evt_type) {
270   case BLE_NUS_C_EVT_DISCOVERY_COMPLETE: {
271     ret_code_t err_code;
272     //err_code = app_button_enable();
273     APP_ERROR_CHECK(err_code);
274     err_code = ble_nus_c_handles_assign(p_ble_nus_c,
275     p_ble_nus_evt->conn_handle, &p_ble_nus_evt->handles);
276     APP_ERROR_CHECK(err_code);
277     // LED Button Service discovered. Enable
```

```
278       //notification of Button.
279       err_code = ble_nus_c_tx_notif_enable(p_ble_nus_c);
280       APP_ERROR_CHECK(err_code);
281     } break; // BLE_LBS_C_EVT_DISCOVERY_COMPLETE
282     default:
283       // No implementation needed.
284       break;
285     }
286   }
287   static void ble_evt_handler(ble_evt_t const *p_ble_evt,
288   void *p_context) {
289     ret_code_t err_code;
290     // For readability.
291     ble_gap_evt_t const *p_gap_evt = &p_ble_evt->evt.gap_evt;
292     ble_gap_evt_adv_report_t const *p_adv_report =
293     &p_ble_evt->evt.gap_evt.params.adv_report;
294     switch (p_ble_evt->header.evt_id) {
295
296     case BLE_GAP_EVT_ADV_REPORT: {
297       if (ble_conn_state_central_conn_count() !=
298       NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
299         scan_start();
300       }
301       if (p_gap_evt->
302       params.adv_report.peer_addr.addr[0] == 0xbd) {
303         InsertNewSensorDataIntoMatrix(1,
304         (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
305         if (ble_conn_state_central_conn_count() !=
306          NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
```

```
307           scan_start();
308         }
309       } else if (p_gap_evt->
310       params.adv_report.peer_addr.addr[0] == 0x3a) {
311         InsertNewSensorDataIntoMatrix(2,
312          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
313         if (ble_conn_state_central_conn_count() !=
314         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
315           scan_start();
316         }
317       } else if (p_gap_evt->
318       params.adv_report.peer_addr.addr[0] == 0xe5) {
319         InsertNewSensorDataIntoMatrix(3,
320          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
321         if (ble_conn_state_central_conn_count() !=
322         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
323           scan_start();
324         }
325       }
326       else if (p_gap_evt->
327       params.adv_report.peer_addr.addr[0] == 0x3d) {
328         InsertNewSensorDataIntoMatrix(4,
329          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
330         if (ble_conn_state_central_conn_count() !=
331         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
332           scan_start();
333         }
334       }
335       else if (p_gap_evt->
```

```
336     params.adv_report.peer_addr.addr[0] == 0x8a) {
337       InsertNewSensorDataIntoMatrix(5,
338        (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
339       if (ble_conn_state_central_conn_count() !=
340       NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
341         scan_start();
342       }
343     }
344     else if (p_gap_evt->
345     params.adv_report.peer_addr.addr[0] == 0x57) {
346       InsertNewSensorDataIntoMatrix(6,
347        (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
348       if (ble_conn_state_central_conn_count() !=
349       NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
350         scan_start();
351       }
352     }
353     else if (p_gap_evt->
354     params.adv_report.peer_addr.addr[0] == 0xc2) {
355       InsertNewSensorDataIntoMatrix(7,
356        (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
357       if (ble_conn_state_central_conn_count() !=
358       NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
359         scan_start();
360       }
361     }
362     else if (p_gap_evt->
363     params.adv_report.peer_addr.addr[0] == 0x6b) {
364       InsertNewSensorDataIntoMatrix(8,
```

```
365          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
366        if (ble_conn_state_central_conn_count() !=
367        NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
368          scan_start();
369        }
370      }
371      else if (p_gap_evt->
372      params.adv_report.peer_addr.addr[0] == 0x9c) {
373        InsertNewSensorDataIntoMatrix(9,
374         (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
375        if (ble_conn_state_central_conn_count() !=
376        NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
377          scan_start();
378        }
379      }
380      else if (p_gap_evt->
381      params.adv_report.peer_addr.addr[0] == 0x1b) {
382        InsertNewSensorDataIntoMatrix(10,
383         (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
384        if (ble_conn_state_central_conn_count() !=
385         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
386          scan_start();
387        }
388      }
389      else if (p_gap_evt->
390      params.adv_report.peer_addr.addr[0] == 0x66) {
391        InsertNewSensorDataIntoMatrix(11,
392         (char *)p_gap_evt->
393         params.adv_report.data.p_data + 5, 26);
```

```
394         if (ble_conn_state_central_conn_count() !=
395         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
396           scan_start();
397         }
398       }    else if (p_gap_evt->
399       params.adv_report.peer_addr.addr[0] == 0xca) {
400         InsertNewSensorDataIntoMatrix(12,
401          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
402         if (ble_conn_state_central_conn_count() !=
403         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
404           scan_start();
405         }
406       }    else if (p_gap_evt->
407       params.adv_report.peer_addr.addr[0] == 0xef) {
408         InsertNewSensorDataIntoMatrix(13,
409          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
410         if (ble_conn_state_central_conn_count() !=
411         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
412           scan_start();
413         }
414       }    else if (p_gap_evt->
415       params.adv_report.peer_addr.addr[0] == 0x8c) {
416         InsertNewSensorDataIntoMatrix(14,
417          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
418         if (ble_conn_state_central_conn_count() !=
419         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
420           scan_start();
421         }
422       }    else if (p_gap_evt->
```

```
423       params.adv_report.peer_addr.addr[0] == 0xe4) {
424         InsertNewSensorDataIntoMatrix(15,
425          (char *)p_gap_evt->params.adv_report.data.p_data + 5, 26);
426         if (ble_conn_state_central_conn_count() !=
427         NRF_SDH_BLE_CENTRAL_LINK_COUNT) {
428           scan_start();
429         }
430       }
431     } break;
432
433     case BLE_GAP_EVT_CONNECTED: {
434       //sd_nvic_SystemReset();
435       //Init_Sensor_Data();
436       //scan_start();
437     } break; // BLE_GAP_EVT_CONNECTED
438
439     case BLE_GAP_EVT_DISCONNECTED: {
440       //sd_nvic_SystemReset();
441       //Init_Sensor_Data();
442       //scan_start();
443     } break; // BLE_GAP_EVT_DISCONNECTED
444
445     case BLE_GAP_EVT_TIMEOUT: {
446     } break; //BLE_GAP_EVT_TIMEOUT
447
448     case BLE_GAP_EVT_CONN_PARAM_UPDATE_REQUEST: {
449     } break; //BLE_GAP_EVT_CONN_PARAM_UPDATE_REQUEST
450
451     case BLE_GAP_EVT_PHY_UPDATE_REQUEST: {
```

```
452     } break; //BLE_GAP_EVT_PHY_UPDATE_REQUEST
453     default:
454       break;
455     }
456   }
457   /**@brief LED Button collector initialization. */
458   static void nus_c_init(void) {
459     ret_code_t err_code;
460     ble_nus_c_init_t nus_c_init_obj;
461     nus_c_init_obj.evt_handler = ble_nus_c_evt_handler;
462     for (uint32_t i = 0; i <
463     NRF_SDH_BLE_CENTRAL_LINK_COUNT; i++) {
464       err_code = ble_nus_c_init(&m_nus_c[i],
465        &nus_c_init_obj);
466       APP_ERROR_CHECK(err_code);
467     }
468   }
469   /**@brief Function for initializing the BLE stack.
470    *
471    * @details Initializes the SoftDevice and the BLE event interrupts.
472    */
473   static void ble_stack_init(void) {
474     ret_code_t err_code;
475     err_code = nrf_sdh_enable_request();
476     APP_ERROR_CHECK(err_code);
477     // Configure the BLE stack using the default settings.
478     // Fetch the start address of the application RAM.
479     uint32_t ram_start = 0;
480     err_code = nrf_sdh_ble_default_cfg_set(
```

```
481   APP_BLE_CONN_CFG_TAG, &ram_start);
482   APP_ERROR_CHECK(err_code);
483   // Enable BLE stack.
484   err_code = nrf_sdh_ble_enable(&ram_start);
485   APP_ERROR_CHECK(err_code);
486   // Register a handler for BLE events.
487   NRF_SDH_BLE_OBSERVER(m_ble_observer,
488    APP_BLE_OBSERVER_PRIO, ble_evt_handler, NULL);
489 }
490 static void db_disc_handler(
491 ble_db_discovery_evt_t *p_evt) {
492
493   ble_nus_c_on_db_disc_evt(
494   &m_nus_c[p_evt->conn_handle], p_evt);
495 }
496
497 /** @brief Database discovery initialization.
498  */
499 static void db_discovery_init(void) {
500   ret_code_t err_code =
501   ble_db_discovery_init(db_disc_handler);
502   APP_ERROR_CHECK(err_code);
503 }
504
505 /**@brief Function for initializing power management.
506  */
507 static void power_management_init(void) {
508   ret_code_t err_code;
509   err_code = nrf_pwr_mgmt_init();
```

```
510     APP_ERROR_CHECK(err_code);
511 }
512
513 /**@brief Function for handling the idle state (main loop).
514  *
515  * @details This function handles any pending
516  log operations, then sleeps until the next event occurs.
517  */
518 static void idle_state_handle(void) {
519 }
520
521 /** @brief Function for initializing the log module.
522  */
523 static void log_init(void) {
524     ret_code_t err_code = NRF_LOG_INIT(NULL);
525     APP_ERROR_CHECK(err_code);
526
527     NRF_LOG_DEFAULT_BACKENDS_INIT();
528 }
529
530 /** @brief Function for initializing the timer.
531  */
532 static void timer_init(void) {
533     ret_code_t err_code = app_timer_init();
534     APP_ERROR_CHECK(err_code);
535 }
536 int main(void) {
537 // Initialize.
538 ////////////////////DEBUG/////////////////////
```

```
539 #ifndef DEBUG_application
540   Uart_define();
541 #endif
542 ////////////REFER TO SDK_CONFIG////////////
543   Init_Sensor_Data();
544   log_init();
545   timer_init();
546   db_discovery_init();
547   ble_stack_init();
548   nus_c_init();
549   ble_conn_state_init();
550   uint32_t lastt = 0;
551   lastt = NRF_RTC0->COUNTER;
552   for (;;) {
553     scan_init();
554     scan_start();
555     uint32_t ticks = NRF_RTC0->COUNTER;
556     if (ticks >= lastt + DELTAT) {
557       lastt += DELTAT;
558       char NewData;
559       char LineCounter;
560       for (int n = 0; n < MAX_SENSORS; n++) {
561         NewData = 0;
562         CoherentReading(n);
563         LineCounter = CursorLine[0];
564         if (LastDataCounter[n] != LineCounter)
565           NewData = 1;
566         if (LineCounter == 0)
567           NewData = 255;
```

```
568           LastDataCounter[n] = LineCounter;
569           printf("%02x%02x", n + 1, NewData);
570           for (int m = 1; m < 26; m++)
571             printf("%02x", CursorLine[m]);
572           printf("\n");
573         }
574         //printf("\n");
575       }
576       if (NRF_LOG_PROCESS() == false) {
577         nrf_pwr_mgmt_run();
578       }
579     }
580 }
```

Appendix B

MATLAB data Representation software

This is the code of the MATLAB data representation program. This program requires the file hex2singlenum.m which is available in the GitHub repository of the project [37].

```
%Initialize
clear all
close all
clc
%Global variable definitions for numerical representation
global q1 q2 q3 q4;
q1 = zeros(1,15);
q2 = zeros(1,15);
q3 = zeros(1,15);
q4 = zeros(1,15);
global r1 r2 r3;
r1 = zeros(1,15);
r2 = zeros(1,15);
```

```
14 r3 = zeros(1,15);
15 %% Opening serial port
16 if exist('PORT','var')
17     flushinput(PORT);
18     fclose(PORT);
19 end
20 % Serial port parameters
21 PORT = serial('COM6');
22 PORT.BaudRate = 230400;
23 PORT.InputBufferSize = 54*16;
24 % PORT.InputBufferSize = 1024*50;
25 fopen(PORT);
26
27 %Main cycle
28 while(1)
29    %Obtain serial incoming packets
30    s = fgetl(PORT);
31    %Analyze incoming data for every matrix row 54 columns
32    %(exclude termination character)
33    if (length(s) == 54)
34        %COL 3 and 4, declaring the sensor state (ON/OFF)
35        STATE=s(3:4);
36        %Check sensor state
37        if (STATE == '01')
38        %COL 1 and 2, indicating the sensor number
39        inp = s(1:2);
40        %Split the task for every sensor data
41        switch inp
42            case '01'
```

```
43                  ProcessSensorData(1,s);
44              case '02'
45                  ProcessSensorData(2,s);
46              case '03'
47                  ProcessSensorData(3,s);
48              case '04'
49                  ProcessSensorData(4,s);
50              case '05'
51                  ProcessSensorData(5,s);
52              case '06'
53                  ProcessSensorData(6,s);
54              case '07'
55                  ProcessSensorData(7,s);
56              case '08'
57                  ProcessSensorData(8,s);
58              case '09'
59                  ProcessSensorData(9,s);
60              case '0a'
61                  ProcessSensorData(10,s);
62              case '0b'
63                  ProcessSensorData(11,s);
64              case '0c'
65                  ProcessSensorData(12,s);
66              case '0d'
67                  ProcessSensorData(13,s);
68              case '0e'
69                  ProcessSensorData(14,s);
70              case '0f'
71                  ProcessSensorData(15,s);
```

```
72          end
73          end
74      end
75  end
76
77  %Function for processing the data matrix
78  function ProcessSensorData(ns, s)
79
80  %Declare global variables to be
81  %used inside the function
82  global q1 q2 q3 q4;
83  global r1 r2 r3;
84
85  %Split the to obtain quaternions
86  q1(ns) = hexsingle2num([s(11:12) s(9:10)
87  s(7:8) s(5:6)]);
88  q2(ns) = hexsingle2num([s(19:20) s(17:18)
89  s(15:16) s(13:14)]);
90  q3(ns) = hexsingle2num([s(27:28) s(25:26)
91  s(23:24) s(21:22)]);
92  q4(ns) = hexsingle2num([s(35:36) s(33:34)
93  s(31:32) s(29:30)]);
94
95  %Obtain Euler angles based on Quaternion values
96  [r1(ns) r2(ns) r3(ns)] = quat2angle([q1(ns)
97  q2(ns) q3(ns) q4(ns)]);
98
99  %Figure for 3D representation
100 figure(1)
```

```
101 subplot(4,4,ns)
102
103 %Make the disk object
104 [x,y,z] = sphere;
105 z = z / 20;
106 h = surf(x,y,z);
107 axis('square');
108 title('SENSOR ' + string(ns));
109 xlabel('x'); ylabel('y'); zlabel('z');
110 xlim([-1 1]);
111 ylim([-1 1]);
112 zlim([-1 1]);
113
114 %Rotate the disk object based on Euler angles
115 rotate(h,[1,0,0],(r3(ns))*180/pi)
116 rotate(h,[0,1,0],(r2(ns))*180/pi)
117 rotate(h,[0,0,1],(r1(ns))*180/pi+90)
118
119 %Represent the disk object
120 view(80,10);
121 drawnow
122
123 %Call the function for terminal
124 %representation of the numerical data
125 represent(ns)
126
127 end
128
129 %Function for terminal representation of
```

```
130 %the numerical data
131 function represent(num)
132
133 %Declare the global variables which have the
134 %previously processed numerical values
135 ns = num;
136 global q1 q2 q3 q4;
137 global r1 r2 r3;
138
139 %Represent on the terminal output
140 clc
141 disp(sprintf('quatW_s1 : %f quatW_s2 : %f quatW_s3 :
142 %f quatW_s4 : %f quatW_s5 : %f',q1(1),
143 q1(2),q1(3),q1(4),q1(5)));
144 disp(sprintf('quatX_s1 : %f quatX_s2 : %f quatX_s3 :
145 %f quatX_s4 : %f quatX_s5 : %f',q2(1),
146 q2(2),q2(3),q2(4),q2(5)));
147 disp(sprintf('quatY_s1 : %f quatY_s2 : %f quatY_s3 :
148 %f quatY_s4 : %f quatY_s5 : %f',q3(1),
149 q3(2),q3(3),q3(4),q3(5)));
150 disp(sprintf('quatZ_s1 : %f quatZ_s2 : %f quatZ_s3 :
151 %f quatZ_s4 : %f quatZ_s5 : %f',q4(1),
152 q4(2),q4(3),q4(4),q4(5)));
153 disp (' ')
154 disp(sprintf('quatW_s6 : %f quatW_s7 : %f quatW_s8 :
155 %f quatW_s9 : %f quatW_s10 : %f',q1(6),
156 q1(7),q1(8),q1(9),q1(10)));
157 disp(sprintf('quatX_s6 : %f quatX_s7 : %f quatX_s8 :
158 %f quatX_s9 : %f quatX_s10 : %f',q2(6),
```

```
159 q2(7),q2(8),q2(9),q2(10)));
160 disp(sprintf('quatY_s6 : %f quatY_s7 : %f quatY_s8 :
161 %f quatY_s9 : %f quatY_s10 : %f',q3(6),
162 q3(7),q3(8),q3(9),q3(10)));
163 disp(sprintf('quatZ_s6 : %f quatZ_s7 : %f quatZ_s8 :
164 %f quatZ_s9 : %f quatZ_s10 : %f',q4(6),
165 q4(7),q4(8),q4(9),q4(10)));
166 disp (' ')
167 disp(sprintf('quatW_s11 : %f quatW_s12 : %f quatW_s13 :
168 %f quatW_s14 : %f quatW_s15 : %f',q1(11),
169 q1(12),q1(13),q1(14),q1(15)));
170 disp(sprintf('quatX_s11 : %f quatX_s12 : %f quatX_s13 :
171 %f quatX_s14 : %f quatX_s15 : %f',q2(11),
172 q2(12),q2(13),q2(14),q2(15)));
173 disp(sprintf('quatY_s11 : %f quatY_s12 : %f quatY_s13 :
174 %f quatY_s14 : %f quatY_s15 : %f',q3(11),
175 q3(12),q3(13),q3(14),q3(15)));
176 disp(sprintf('quatZ_s11 : %f quatZ_s12 : %f quatZ_s13 :
177 %f quatZ_s14 : %f quatZ_s15 : %f',q4(11),
178 q4(12),q4(13),q4(14),q4(15)));
179 disp (' ')
180 disp (' ')
181 r1(ns) = r1(ns) * pi*180; %Yaw deg
182 r2(ns) = r2(ns) * pi*180; %Pitch deg
183 r3(ns) = r3(ns) * pi*180; %Roll deg
184 disp(sprintf('Yaw_s1   : %f ÃÂ° Yaw_s2   : %f ÃÂ° Yaw_s3   :
185 %f ÃÂ° Yaw_s4   : %f ÃÂ° Yaw_s5   : %f ÃÂ°',
186 r1(1),r1(2),r1(3),r1(4),r1(5)));
187 disp(sprintf('Pitch_s1 : %f ÃÂ° Pitch_s2 : %f ÃÂ° Pitch_s3 :
```

```
188 %f ÃÂ° Pitch_s4 : %f ÃÂ° Pitch_s5 : %f ÃÂ°',
189 r2(1),r2(2),r2(3),r2(4),r2(5)));
190 disp(sprintf('Roll_s1  : %f ÃÂ° Roll_s2  : %f ÃÂ° Roll_s3  :
191 %f ÃÂ° Roll_s4  : %f ÃÂ° Roll_s5  : %f ÃÂ°',
192 r3(1),r3(2),r3(3),r3(4),r3(5)));
193 disp (' ')
194 disp(sprintf('Yaw_s6   : %f ÃÂ° Yaw_s7   : %f ÃÂ° Yaw_s8   :
195 %f ÃÂ° Yaw_s9   : %f ÃÂ° Yaw_s10  : %f ÃÂ°',
196 r1(6),r1(7),r1(8),r1(9),r1(10)));
197 disp(sprintf('Pitch_s6 : %f ÃÂ° Pitch_s7 : %f ÃÂ° Pitch_s8 :
198 %f ÃÂ° Pitch_s9 : %f ÃÂ° Pitch_s10: %f ÃÂ°',
199 r2(6),r2(7),r2(8),r2(9),r2(10)));
200 disp(sprintf('Roll_s6  : %f ÃÂ° Roll_s7  : %f ÃÂ° Roll_s8  :
201 %f ÃÂ° Roll_s9  : %f ÃÂ° Roll_s10 : %f ÃÂ°',
202 r3(6),r3(7),r3(8),r3(9),r3(10)));
203 disp (' ')
204 disp(sprintf('Yaw_s11  : %f ÃÂ° Yaw_s12  : %f ÃÂ° Yaw_s13  :
205 %f ÃÂ° Yaw_s14  : %f ÃÂ° Yaw_s15  : %f ÃÂ°',
206 r1(11),r1(12),r1(13),r1(14),r1(15)));
207 disp(sprintf('Pitch_s11: %f ÃÂ° Pitch_s12: %f ÃÂ° Pitch_s13:
208 %f ÃÂ° Pitch_s14: %f ÃÂ° Pitch_s15: %f ÃÂ°',
209 r2(11),r2(12),r2(13),r2(14),r2(15)));
210 disp(sprintf('Roll_s11 : %f ÃÂ° Roll_s12 : %f ÃÂ° Roll_s13 :
211 %f ÃÂ° Roll_s14 : %f ÃÂ° Roll_s15 : %f ÃÂ°',
212 r3(11),r3(12),r3(13),r3(14),r3(15)));
213
214 end
```

Appendix C

Unity data representation software

This is the code of the Unity data representation program. This code handles the incoming data from the central receiver as events and applies the rotation angles to the corresponding body part (game object).

```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using System.Collections;
4  using System.IO.Ports;
5  using System;
6  using System.Linq;
7  using TMPro;
8  //using System.index;
9
10
11
12 public class portsConn : MonoBehaviour
13
```

```
14 {
15
16
17     //Body Parts as game objects    //OBJ Number
18     public GameObject forehead;     //1
19     public GameObject chest;        //2
20     public GameObject leftArm;      //3
21     public GameObject rightArm;     //4
22     public GameObject leftWrist;    //5
23     public GameObject rightWrist;   //6
24     public GameObject leftHand;     //7
25     public GameObject rightHand;    //8
26     public GameObject scaral;        //9
27     public GameObject leftKnee;     //10
28     public GameObject rightKnee;    //11
29     public GameObject leftAnkle;    //12
30     public GameObject rightAnkle;   //13
31     public GameObject leftFoot;     //14
32     public GameObject rightFoot;    //15
33     //================================
34     public Quaternion initialForehead;
35     public Quaternion initialChest;
36     public Quaternion initialLeftArm;
37     public Quaternion initialRightArm;
38     public Quaternion initialLeftWrist;
39     public Quaternion initialRightWrist;
40     public Quaternion initialLeftHand;
41     public Quaternion initialRightHand;
42     public Quaternion initialScaral;
```

```
43      public Quaternion initialLeftKnee;
44      public Quaternion initialRightKnee;
45      public Quaternion initialLeftAnkle;
46      public Quaternion initialRightAnkle;
47      public Quaternion initialLeftFoot;
48      public Quaternion initialRightFoot;
49      //==================================
50      public Quaternion inverseQuaternion;
51
52      public Button closeButton;
53      public Button connetButton;
54      public bool connectionstate = false;
55      public Dropdown DropdownPORTS;
56      SerialPort sp;
57      // Start is called before the first frame update
58
59      //Obtain initial rotation from body parts in quaternion
60      private Quaternion initial(GameObject bodypart)
61      {
62          Quaternion initialValue = new Quaternion();
63          initialValue = bodypart.transform.rotation;
64          return initialValue;
65      }
66      void applyRotation(GameObject bodypart,
67          Quaternion rotations,
68          Quaternion initialRotations)
69      {
70          bodypart.transform.rotation =
71              initialRotations
```

```
72              * inverseQuaternion
73              * rotations;
74      }
75
76      void Start()
77      {
78          inverseQuaternion = Quaternion.identity;
79          string[] ports = SerialPort.GetPortNames();
80          DropdownPORTS.AddOptions(ports.ToList());
81          Button btn = connetButton.GetComponent<Button>();
82          Button btnClose = closeButton.GetComponent<Button>();
83          btn.onClick.AddListener(Connect);
84          btnClose.onClick.AddListener(Close);
85          //obtaining initial rotations
86          initialForehead = initial(forehead)
87              * Quaternion.Euler(0, 180, 0);
88          initialChest = initial(chest)
89              * Quaternion.Euler(0, 180, 0);
90          initialLeftArm = initial(leftArm)
91              * Quaternion.Euler(0, 180, 0);
92          initialRightArm = initial(rightArm)
93              * Quaternion.Euler(0, 180, 0);
94          initialLeftWrist = initial(leftWrist)
95              * Quaternion.Euler(0, 180, 0);
96          initialRightWrist = initial(rightWrist)
97              * Quaternion.Euler(0, 180, 0);
98          initialLeftHand = initial(leftHand)
99              * Quaternion.Euler(0, 180, 0);
100         initialRightHand = initial(rightHand)
```

```
101             * Quaternion.Euler(0, 180, 0);
102         initialScaral = initial(scaral)
103             * Quaternion.Euler(0, 180, 0);
104         initialLeftKnee = initial(leftKnee)
105             * Quaternion.Euler(0, 180, 0);
106         initialRightKnee = initial(rightKnee)
107             * Quaternion.Euler(0, 180, 0);
108         initialLeftAnkle = initial(leftAnkle)
109             * Quaternion.Euler(0, 180, 0);
110         initialRightAnkle = initial(rightAnkle)
111             * Quaternion.Euler(0, 180, 0);
112         initialLeftFoot = initial(leftFoot)
113             * Quaternion.Euler(0, 180, 0);;
114         initialRightFoot = initial(rightFoot)
115             * Quaternion.Euler(0, 180, 0);
116         //sp.Close();
117     }
118     public float hexstring2quaternion(string hexString)
119     {
120         byte[] raw = new byte[hexString.Length / 2];
121         for (int i = 0; i < raw.Length; i++)
122         {
123             // THEN DEPENDING ON ENDIANNESS
124             raw[i] = Convert.ToByte(
125                 hexString.Substring(i * 2, 2), 16);
126             // OR
127             // raw[raw.Length - i - 1] =
128             //Convert.ToByte(q1.Substring(i * 2, 2), 16);
129         }
```

```
130         float f = BitConverter.ToSingle(raw, 0);
131         return f;
132     }
133 
134     public float[] hex2angleQT(string datas)
135     {
136         char[] x = new char[8];
137         for (int i = 0; i < 8; i++)
138         {
139             x[i] = datas[5 + i];
140         }
141         string q1 = new string(x).ToUpper();
142         float angleQ1 = hexstring2quaternion(q1);
143         for (int i = 0; i < 8; i++)
144         {
145             x[i] = datas[13 + i];
146         }
147         string q2 = new string(x).ToUpper();
148         float angleQ2 = hexstring2quaternion(q2);
149         for (int i = 0; i < 8; i++)
150         {
151             x[i] = datas[21 + i];
152         }
153         string q3 = new string(x).ToUpper();
154         float angleQ3 = hexstring2quaternion(q3);
155         for (int i = 0; i < 8; i++)
156         {
157             x[i] = datas[29 + i];
158         }
```

```
159         string q4 = new string(x).ToUpper();
160         float angleQ4 = hexstring2quaternion(q4);
161 
162         float[] angles = { angleQ1, angleQ2, angleQ3, angleQ4 };
163         return angles;
164 
165     }
166     public float[] quaternion2YPWradian(float Q1, float Q2,
167         float Q3, float Q4)
168     {
169         float roll = Mathf.Atan2(2 * Q2 * Q4 - 2 * Q1 * Q3,
170             1 - 2 * Q2 * Q2 - 2 * Q3 * Q3);
171         float pitch = Mathf.Atan2(2 * Q1 * Q4 - 2 * Q2 * Q3,
172             1 - 2 * Q1 * Q1 - 2 * Q3 * Q3);
173         float yaw = Mathf.Asin(2 * Q1 * Q2 + 2 * Q3 * Q4);
174         float[] YPW = { yaw, pitch, roll };
175         return YPW;
176     }
177     // Update is called once per frame
178     void Update()
179     {
180         if (connectionstate == true)
181         {
182             string datas = sp.ReadLine();
183             if (datas[2].ToString() == "1" &&
184                 datas[4].ToString() == "1")
185             {
186                 float[] anglesQt;
187                 anglesQt = hex2angleQT(datas);
```

```
188                 var rotations = new Quaternion(anglesQt[3],
189                     anglesQt[0], anglesQt[1], anglesQt[2]);
190                 applyRotation(forehead, rotations,
191                     initialForehead);
192
193             }
194             else if (datas[2].ToString() == "2" &&
195                 datas[4].ToString() == "1")
196             {
197                 float[] anglesQt;
198                 anglesQt = hex2angleQT(datas);
199                 var rotations = new Quaternion(anglesQt[3],
200                     anglesQt[0], anglesQt[1], anglesQt[2]);
201                 applyRotation(chest, rotations,
202                     initialChest);
203
204             }
205             else if (datas[2].ToString() == "3" &&
206                 datas[4].ToString() == "1")
207             {
208                 float[] anglesQt;
209                 anglesQt = hex2angleQT(datas);
210                 var rotations = new Quaternion(anglesQt[3],
211                     anglesQt[0], anglesQt[1], anglesQt[2]);
212                 applyRotation(leftArm, rotations,
213                     initialLeftArm);
214
215             }
216             else if (datas[2].ToString() == "4" &&
```

```
217                 datas[4].ToString() == "1")
218             {
219                 float[] anglesQt;
220                 anglesQt = hex2angleQT(datas);
221                  var rotations = new Quaternion(anglesQt[3],
222                      anglesQt[0],
223                      anglesQt[1],anglesQt[2] );
224                 applyRotation(rightArm, rotations,
225                     initialRightArm);
226
227             }
228             else if (datas[2].ToString() == "5" &&
229                 datas[4].ToString() == "1")
230             {
231                 float[] anglesQt;
232                 anglesQt = hex2angleQT(datas);
233                  var rotations = new Quaternion(anglesQt[3],
234                      anglesQt[0],
235                      anglesQt[1],anglesQt[2] );
236                 applyRotation(leftWrist, rotations,
237                     initialLeftWrist);
238
239             }
240             else if (datas[2].ToString() == "6" &&
241                 datas[4].ToString() == "1")
242             {
243                 float[] anglesQt;
244                 anglesQt = hex2angleQT(datas);
245                  var rotations = new Quaternion(anglesQt[3],
```

```
246                     anglesQt[0],
247                     anglesQt[1],anglesQt[2] );
248                 applyRotation(rightWrist, rotations,
249                     initialRightWrist);
250
251             }
252             else if (datas[2].ToString() == "7" &&
253                 datas[4].ToString() == "1")
254             {
255                 float[] anglesQt;
256                 anglesQt = hex2angleQT(datas);
257                  var rotations = new Quaternion(anglesQt[3],
258                     anglesQt[0],
259                     anglesQt[1],anglesQt[2] );
260                 applyRotation(leftHand, rotations,
261                     initialLeftHand);
262
263             }
264             else if (datas[2].ToString() == "8" &&
265                 datas[4].ToString() == "1")
266             {
267                 float[] anglesQt;
268                 anglesQt = hex2angleQT(datas);
269                  var rotations = new Quaternion(anglesQt[3],
270                     anglesQt[0],
271                     anglesQt[1],anglesQt[2] );
272                 applyRotation(rightHand, rotations,
273                     initialRightHand);
274
```

```
275             }
276             else if (datas[2].ToString() == "9" &&
277                 datas[4].ToString() == "1")
278             {
279                 float[] anglesQt;
280                 anglesQt = hex2angleQT(datas);
281                  var rotations = new Quaternion(anglesQt[3],
282                      anglesQt[0],
283                      anglesQt[1],anglesQt[2] );
284                 applyRotation(scaral, rotations,
285                     initialScaral);
286 
287             }
288             else if (datas[2].ToString() == "a" &&
289                 datas[4].ToString() == "1")
290             {
291                 float[] anglesQt;
292                 anglesQt = hex2angleQT(datas);
293                  var rotations = new Quaternion(anglesQt[3],
294                      anglesQt[0],
295                      anglesQt[1],anglesQt[2] );
296                 applyRotation(leftKnee, rotations,
297                     initialLeftKnee);
298 
299             }
300             else if (datas[2].ToString() == "b" &&
301                 datas[4].ToString() == "1")
302             {
303                 float[] anglesQt;
```

```
304                 anglesQt = hex2angleQT(datas);
305                  var rotations = new Quaternion(anglesQt[3],
306                      anglesQt[0],
307                      anglesQt[1],anglesQt[2] );
308                 applyRotation(rightKnee, rotations,
309                     initialRightKnee);
310
311             }
312             else if (datas[2].ToString() == "c" &&
313                 datas[4].ToString() == "1")
314             {
315                 float[] anglesQt;
316                 anglesQt = hex2angleQT(datas);
317                  var rotations = new Quaternion(anglesQt[3],
318                      anglesQt[0],
319                      anglesQt[1],anglesQt[2] );
320                 applyRotation(rightAnkle, rotations,
321                     initialRightAnkle);
322
323             }
324             else if (datas[2].ToString() == "d" &&
325                 datas[4].ToString() == "1")
326             {
327                 float[] anglesQt;
328                 anglesQt = hex2angleQT(datas);
329                  var rotations = new Quaternion(anglesQt[3],
330                      anglesQt[0],
331                      anglesQt[1],anglesQt[2] );
332                 applyRotation(leftFoot, rotations,
```

```
333                     initialLeftFoot);
334
335             }
336             else if (datas[2].ToString() == "e" &&
337                 datas[4].ToString() == "1")
338             {
339                 float[] anglesQt;
340                 anglesQt = hex2angleQT(datas);
341                  var rotations = new Quaternion(anglesQt[3],
342                      anglesQt[0],
343                      anglesQt[1],anglesQt[2] );
344                 applyRotation(rightFoot, rotations,
345                     initialRightFoot);
346
347             }
348         }
349     }
350
351     //Serial communication initialization
352     public void Connect()
353     {
354         string port =
355             DropdownPORTS.options[DropdownPORTS.value].text;
356         sp = new SerialPort(port, 230400)
357         {
358             ReadBufferSize = 8192,
359             ReadTimeout = 2000
360         };
361         if (!sp.IsOpen)
```

```
362         {
363             sp.Open();
364             connectionstate = true;
365         }
366     }
367     public void Close()
368     {
369         sp.Close();
370         connectionstate = false;
371     }
372 }
```

Bibliography

[1] Wei-Hua Tan et al. "ePet: A Physical Game Based on Wireless Sensor Networks". In: *International Journal of Distributed Sensor Networks* 5.1 (2009), pp. 68–68. DOI: 10.1080/15501320802555262. eprint: https://doi.org/10.1080/15501320802555262. URL: https://doi.org/10.1080/15501320802555262.

[2] Filippo Casamassima et al. "A Wearable System for Gait Training in Subjects with Parkinsonâs Disease". In: *Sensors* 14.4 (2014), pp. 6229–6246. ISSN: 1424-8220. DOI: 10.3390/s140406229. URL: https://www.mdpi.com/1424-8220/14/4/6229.

[3] Le Nguyen Ngu Nguyen et al. "Basketball Activity Recognition Using Wearable Inertial Measurement Units". In: *Proceedings of the XVI International Conference on Human Computer Interaction*. Interacción '15. Vilanova i la Geltru, Spain: Association for Computing Machinery, 2015. ISBN: 9781450334631. DOI: 10.1145/2829875.2829930. URL: https://doi.org/10.1145/2829875.2829930.

[4] D. Kelly et al. "Automatic detection of collisions in elite level rugby union using a wearable sensing device". In: *Sports Engineering* 15 (2012), pp. 81–92. DOI: 10.1007/S12283-012-0088-5. URL: https://doi.org/10.1007/s12283-012-0088-5.

[5] Graham D. Finlayson. "Colour and illumination in computer vision". In: *Interface Focus* 8.4 (2018), p. 20180008. DOI: 10.1098/rsfs.2018.0008.

eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsfs.2018.0008. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsfs.2018.0008.

[6] Cheong, Yun Zhe and Chew, Wei Jen. "The Application of Image Processing to Solve Occlusion Issue in Object Tracking". In: *MATEC Web Conf.* 152 (2018), p. 03001. DOI: 10.1051/matecconf/201815203001. URL: https://doi.org/10.1051/matecconf/201815203001.

[7] Igor Khokhlov et al. "Design of activity recognition systems with wearable sensors". In: *2018 IEEE Sensors Applications Symposium (SAS)*. 2018, pp. 1–6. DOI: 10.1109/SAS.2018.8336752.

[8] Javier Marin, Teresa Blanco, and Jose J. Marin. "Octopus: A Design Methodology for Motion Capture Wearables". In: *Sensors* 17.8 (2017). ISSN: 1424-8220. DOI: 10.3390/s17081875. URL: https://www.mdpi.com/1424-8220/17/8/1875.

[9] Stefano Canali. "Towards a Contextual Approach to Data Quality". In: *Data* 5.4 (2020). ISSN: 2306-5729. DOI: 10.3390/data5040090. URL: https://www.mdpi.com/2306-5729/5/4/90.

[10] Peng-zhan Chen et al. "Real-Time Human Motion Capture Driven by a Wireless Sensor Network". In: *International Journal of Computer Games Technology* 2015 (Feb. 2015), p. 695874. DOI: 10.1155/2015/695874. URL: https://doi.org/10.1155/2015/695874.

[11] Agnieszka SzczÄsna et al. "Inertial Motion Capture Costume Design Study". In: *Sensors* 17.3 (2017). ISSN: 1424-8220. DOI: 10.3390/s17030612. URL: https://www.mdpi.com/1424-8220/17/3/612.

[12] Karol JÄdrasiak, Krzysztof Daniec, and Aleksander Nawrat. "The low cost micro inertial measurement unit". In: *2013 IEEE 8th Conference on In-*

dustrial Electronics and Applications (ICIEA). 2013, pp. 403–408. DOI: 10.1109/ICIEA.2013.6566403.

[13] A.M. Sabatini. "Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing". In: *IEEE Transactions on Biomedical Engineering* 53.7 (2006), pp. 1346–1356. DOI: 10.1109/TBME.2006.875664.

[14] Robert Mahony, Tarek Hamel, and Jean-Michel Pflimlin. "Nonlinear Complementary Filters on the Special Orthogonal Group". In: *IEEE Transactions on Automatic Control* 53.5 (2008), pp. 1203–1218. DOI: 10.1109/TAC.2008.923738.

[15] URL: https://jcgm.bipm.org/vim/en/3.8.html.

[16] URL: http://viml.oiml.info/en/0.11.html.

[17] Daniel Tazartes. "An historical perspective on inertial navigation systems". In: *2014 International Symposium on Inertial Sensors and Systems (ISISS)*. 2014, pp. 1–5. DOI: 10.1109/ISISS.2014.6782505.

[18] Jorg F. Wagner and Andor Trierenberg. "The Machine of Bohnenberger: Bicentennial of the Gyro with Cardanic Suspension". In: *PAMM* 10.1 (2010), pp. 659–660. DOI: https://doi.org/10.1002/pamm.201010322. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pamm.201010322. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pamm.201010322.

[19] Norhafizan Ahmad et al. "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: *International Journal of Signal Processing Systems* 1 (Jan. 2013), pp. 256–262. DOI: 10.12720/ijsps.1.2.256-262.

[20] Guo Zhanshe et al. "Research development of silicon MEMS gyroscopes: a review". In: *Microsystem Technologies* 21.10 (Oct. 2015), pp. 2053–2066.

ISSN: 1432-1858. DOI: 10.1007/s00542-015-2645-x. URL: https://doi.org/10.1007/s00542-015-2645-x.

[21] Last Minute Engineers. *In-Depth: Interface MPU6050 accelerometer amp; Gyroscope sensor with Arduino*. Dec. 2020. URL: https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/.

[22] Weimeng Niu et al. "Summary of Research Status and Application of MEMS Accelerometers". In: *Journal of Computer and Communications* 06 (Jan. 2018), pp. 215–221. DOI: 10.4236/jcc.2018.612021.

[23] Hardik Parwana and Mangal Kothari. "Quaternions and Attitude Representation". In: (Aug. 2017).

[24] James Diebel. "Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors". In: *Matrix* 58 (Jan. 2006).

[25] *MPU-6000 and MPU-6050 Product Specification*. PS-MPU-6000A-00. Rev. 3.4. InvenSense. Aug. 2013.

[26] *nRF51802 Multiprotocol Bluetooth low energy/2.4 GHz RF System on Chip Product Specification*. Ver. 1.2. Nordic Semiconductor. May 2016.

[27] *CE6208 SERIES Ultra-Fast High PSRR 1A CMOS Voltage Regulator*. Ver. 1.1. Nanjing Chipower Electronics Inc.

[28] *TP4056 1A Standalone Linear Li-Ion Battery Charger with Thermal Regulation in SOP-8*. NanJing Top Power ASIC Corp.

[29] *nRF52840 Development kit PCA10056 V1.0.0 user guide*. Ver. 1.3. Nordic Semiconductor. Feb. 2019.

[30] *S110 nRF51 Bluetooth low energy Peripheral SoftDevice specifications*. Ver. 2. Nordic Semiconductor. Feb. 2015.

[31] *Understanding the generic access Profile (GAP)*. URL: https://developer.qualcomm.com/hardware/qca4020-qca4024/learning-resources/understanding-generic-access-profile.

[32] *Developer help*. URL: https://microchipdeveloper.com/wireless:ble-gap-security.

[33] *Bluetooth core specifications*. Ver. 5.3. Bluetooth SIG Proprietary. July 2021.

[34] Kevin Townsend et al. *Getting started with Bluetooth low energy*. URL: https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch04.html.

[35] *Getting started with nRF5 SDK and SES (nRF51 nRF52 Series)*. Ver. 1.4. Nordic Semiconductor. Apr. 2020.

[36] "IEEE Standard for Floating-Point Arithmetic". In: *IEEE Std 754-2019 (Revision of IEEE 754-2008)* (2019), pp. 1–84. DOI: 10.1109/IEEESTD.2019.8766229.

[37] SINA RONAGHI. *PC software for attitude representation*. June 2021. URL: https://github.com/SinaRonaghi/PoliTesi-BLE-MoCap-applications.

[38] URL: https://www.mixamo.com/#/.

[39] Nordic semiconductor. *Configuring placement of the SoftDevice*. URL: https://infocenter.nordicsemi.com/topic/ug_gsg_ses/UG/gsg/program_sd_ses.html.

[40] Sina Ronaghi. *PoliTesi - BLE MoCap experiments*. Youtube. 2021. URL: https://youtube.com/playlist?list=PLEQDgukQzWZFPCwNrCT3x3Xi42EmWvrvs.

[41] *GUM: Guide to the Expression of Uncertainty in Measurement*. Sept. 2008. URL: https://www.bipm.org/documents/20126/2071204/JCGM_100_2008_E.pdf.