Politecnico di Milano

School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science in Biomedical Engineering

DESIGN OF A WRIST WEARABLE DEVICE FOR GESTURE RECOGNITION IN THE CONTEXT OF ADHERENCE MEASUREMENT

Supervisor:

Prof. Enrico Gianluca Caiani

Co-Supervisor:

Prof. Andrea Aliverti

Master Thesis by:

Diana Maritza González Ramírez

Matr. 894383

A.A. 2018-2019

Mamá estoy triunfando

Abstract

Poor adherence in medical treatments is one of the biggest problems to tackle in the medical field. Poor adherence reduces clinical benefit provided by the treatment and increases the likelihood of hospitalization, especially in chronic illnesses, while good adherence increases the effectiveness of the treatment and promotes healthy lifestyles. This affects directly not only the quality of life of the patient but also the costs and quality of the healthcare system. Wearable technologies are a great tool to measure and promote medical adherence because they can monitor the at-home behavior of the patient and can also give reminders and feedback.

This project addresses poor adherence by monitoring human activities through the movement of an arm, that could allow physicians to understand whether their patient is following the recommendations, by taking a pill. The approach we gave to this project is the design of a system that can acquire accelerometric signals from the forearm and send them to a smart-phone. In order to allow the data to be processed for gesture tracking, the device is connected to a smart-phone application that sends the data to the cloud.

The designed system includes software and hardware. The wearable device thought for this application is a wrist band. The hardware design included a 6-axis IMU featuring an accelerometer and a gyroscope. This device performs an acquisition only when requested by the smartphone with which communicates via Bluetooth. Considering that the software for gesture tracking analysis (non developed in this thesis) is based on the knowledge of the gesture specific motion pattern that needs to be detected, customized to the specific patient, a training function is needed by which such gesture can be acquired in an univocal way in a dedicated session, the software design includes a systems interface that is adapted for two types of users: regular app user and expert user. Each type of user has specific interactions in the system. The expert user (physician) specifies the gestures to be tracked and the relevant training mode, meaning it can create and train gestures as well as managing the alarms for the regular app users. While the regular app user (patient) can train the already existing gestures and set alarms according to when the medication is needed to be taken. For storing all the data and information a cloud database was also designed.

In the document, all the considerations are taken for the circuit design, the schematics and layouts of the circuit, the state machine that was programmed as firmware in the MCU, as well as a suggested method for testing the device are written. There is also presented the system's use case scenario, activity diagrams; the database class diagram,

entity-relationship diagram, technical description of tables, and queries; and the app environments overview. And will explain overall how the system was built employing a database and a smartphone-app, and how this system works. At the end of the document, there is a discussion about the applicability in the market, the possible improvements, and future work for the designed device.

Sommario

La scarsa aderenza ai trattamenti medici è uno dei maggiori problemi da affrontare in campo medico perché riduce il beneficio clinico fornito dal trattamento e aumenta la probabilità di ospedalizzazione, soprattutto nelle malattie croniche, mentre una buona aderenza aumenta l'efficacia del trattamento e promuove stili di vita sani. Ciò influisce direttamente non solo sulla qualità della vita del paziente, ma anche sui costi e sulla qualità del sistema sanitario. Le tecnologie wearable sono un ottimo strumento per misurare e promuovere l'aderenza medica perché possono monitorare il comportamento del paziente mentre è a casa e possono anche dargli indicazioni e feedback.

Questo progetto affronta la scarsa aderenza monitorando le attività umane attraverso il movimento di un braccio, che potrebbe consentire ai medici di capire quando il loro paziente sta seguendo le raccomandazioni, relative al prendere una pillola. L'approccio che abbiamo avuto con questo progetto è la progettazione di un sistema in grado di acquisire segnali accelerometrici dall'avambraccio e inviarli ad uno smartphone. Per consentire l'elaborazione dei dati per fare gesture tracking, il dispositivo è collegato a un'applicazione per smartphone che invia i dati al cloud.

Il sistema progettato include software e hardware. Il dispositivo wearable pensato per questa applicazione è un bracciale. La progettazione hardware include un sensore IMU a 6 assi con un accelerometro e un giroscopio. Questo dispositivo esegue un'acquisizione solo quando richiesto dallo smartphone con cui comunica tramite Bluetooth. Considerando che il software di analisi dei gesti (non incluso in questa tesi) presuppone la conoscenza del pattern motorio relativo al gesto che si vuole riconoscere, customizzato per il singolo paziente, è necessario predisporre una funzione di "allenamento" in cui tale gesto può essere registrato in modo univoco in una opportuna sessione, il design include un'interfaccia di sistema che è adattata per due tipi di utenti: utente normale dell'app e utente esperto. Ogni tipo di utente ha interazioni specifiche nel sistema. L'utente esperto (medico) specifica i gesti di cui tenere traccia e la modalità di allenamento pertinente, il che significa che può creare e addestrare gesti e gestire gli allarmi per gli utenti normali dell'app, mentre l'utente normale (paziente) dell'app può allenare i gesti già esistenti e impostare allarmi in base a quando è necessario prendere delle medicine. Per l'archiviazione di tutti i dati e le informazioni è stato progettato anche un database sul cloud.

Nel documento, sono presenti tutte le considerazioni che sono state effettuate per la progettazione del circuito, gli schemi e i layout del circuito, la macchina a stati programmata come firmware nell'MCU e un metodo suggerito per testare il dispositivo.

Viene inoltre presentato lo scenario del caso d'uso del sistema, i diagrammi di attività; il diagramma di classe del database, il diagramma entità-relazione, la descrizione tecnica delle tabelle e le query; e la panoramica degli ambienti delle app. Inoltre si spiegherà come è stato creato il sistema utilizzando un database e un'app per smartphone e come esso funzioni. Alla fine del documento, una discussione sull'applicabilità al mercato, i possibili miglioramenti e il lavoro futuro per il dispositivo progettato conclude il documento.

Author's Comments and Acknowledgments

Thanks to Professors Andrea Aliverti and Enrico Caiani, for giving me the opportunity to work in this project, because their guidance and disposition were essential for its development. Also, to Politecnico di Milano for providing me the resources that made this possible, and for all the knowledge I acquired during these years in the master course that was useful in my thesis journey. Thanks a lot, to all the teachers that contributed to my learning process as a professional and a human being.

To my alma mater, Pontificia Universidad Javeriana that honored me with the opportunity of participating in the double degree program with Polimi and that throughout my career gave me essential tools that helped me to carry out this master thesis. Very special thanks to my dear professors Germán Yamhure, Daniel Jaramillo and Rafael Uribe which not only thought me a lot in academics but also encouraged me to pursue my goals and to seek excellence in my profession, following your legacy has been a huge motivation during my journey so far.

Thanks to my dear friends Damián Martínez, Diego Mora, Luis Montaño, Camilo Hurtado, Alejandro Ramírez, Catalina Riaño, Nicky Borrero, Alessandro Cacciatore, and Rabie Mottahedeh, that helped me in moments of distress during my thesis. Also, to my dear friends Sergio Losada, Simón Morales, Diego Parra, Daniela Vaca, Alexandra Ramos, María Libreros, Andrés Hernández, Andrea Rozo, Andrea Dorizza and Bárbara Coloma, because you enlightened me in many different ways and without you, I wouldn't have been able to reach this goal.

But more importantly thanks to my beloved family that has supported in all the possible ways and loved me through thick and thin. Thanks to my parents Dora Emma and Wilson. To all my uncles Migue, Tycko, Rafa, David, Boris, Héctor and Stoyan, to my aunt Perla, to my godmother Lyna, to all my cousins, to my grandparents Tere, Dora and Orlando. Special thanks to my brother, Daniel Julián, because you have been my rock all along, my biggest support, the person that I can always rely on and that helped me not only thought this thesis and this master course but though all my life to go ahead and achieve everything that I aim for. I love you and admire you deeply!

Above all thanks my Lord, my good and loving Father, God.

Contents

List of Figures

List of Tables

1. Introduction

In this section, some concepts will be explained to give a better understanding of the problem this project is trying to address, to understand its relevance in the healthcare context, and to understand better the design choices, the solution development, and the results. Then a description of the problem will be presented, together with an explanation of how this will be addressed.

1.1 Adherence

The main problem we are trying to tackle with this project is the assessment of adherence, so let's first define what adherence is, what are some problems related to poor adherence and how it can be measured.

According to the World Health Organization (WHO), adherence is the extent to which a person's behavior corresponds with agreed recommendations from a health care provider(Sabaté, 2003) (Seeking medical attention, filling prescriptions, taking medication appropriately, following a diet, obtaining immunizations, attending follow-up appointments, executing behavioral modifications that address personal hygiene, self-management of asthma or diabetes, smoking, contraception, risky sexual behaviors, involving in physical activity or any other lifestyle changes). Adherence requires the patient's agreement to the recommendations, so communication between patient and health professional plays a huge role in adherence to treatment.

Adherence among patients suffering from chronic diseases in developed countries was about 50% in 2003, is estimated to be much lower in developing counties. There is strong evidence that many patients with chronic illnesses have difficulty adhering to their recommended regimens, especially in situations where the treatment is required to be self-administrated, regardless of the type of disease, severity, and access to health resources. This causes medical and psycho-social complications of the disease, reduces patients' life quality, and wastes health care resources, making more difficult

for health systems to achieve the population health goals. This a major issue for healthcare systems and patients because adherence is a primary determinant of the effectiveness of a treatment, poor adherence reduces clinical benefit provided by the treatment, while good adherence increases the effectiveness of the treatment and promotes healthy lifestyles. (Sabaté, 2003) Since most of the regimes for chronic conditions are self-managed by the patient, patients face several potentially life-threatening risks if health recommendations are not followed as they were prescribed, such as more intense relapses, increased risk of dependence, increased risk of abstinence and rebound effect, increased risk of developing resistance to therapies, increased risk of toxicity and increased likelihood of accidents. (Sabaté, 2003) In fact, on a 2014 study in the UK approximately a third of asthma deaths could have been avoided by taking the medication appropriately, and in 2015 about 60% of asthma-related hospitalizations were related to poor adherence.(Normansell, Kew, & Stovold, 2017) Increasing the effectiveness of adherence might have a greater impact on health than improvements in specific medical treatments and also might save costs in interventions when there are investments in improving adherence.(Sabaté, 2003)

Many different factors contribute to lack of adherence like socioeconomic status, literacy, forgetfulness, treatment anxiety, misunderstood instructions, lack of motivation, comorbidities, level of disability, patient-provider relationship, among others.(Sabaté, 2003) The ability of patients to follow their treatment correctly may be compromised by several of the factors above, the implementation of this project aims to tackle mainly patient-related factors such as forgetfulness and misunderstanding of instructions and patient-provider relationship, for this we will introduce how adherence is measured.

Measuring adherence is essential for planning effective and efficient treatments, it also makes possible to attribute changes in health outcomes to the recommended behavior. Health outcomes can be predicted more easily and accurately by treatment efficacy if there is enough data, which is only achievable when there is good adherence. Even though there is no "gold standard" for measuring adherence, there are several methods reported in literature that we can broadly divide into objective and subjective measurements. Subjective measurements consist primarily of self-report performed by the patient, that can be given in an assessment with the provider or through standardized questionnaires. A major problem with this kind of measurement is that the patients tend to under-report the non-adherence, causing an overestimation in the measured adherence. Objective measurements initially appeared to improve subjective approaches, but they also had several drawbacks. This kind of measure encompasses many different

measuring methods like dosage units counting, databases analysis, biochemical measurements and electronic monitoring.(Sabaté, 2003)

In this context, the aim of this project is to design a tool for having objective measurements of adherence through electronic monitoring and telemedicine. Telemedicine is part of the expanding use of communications technology in health care being used in prevention, disease management, home health care, long-term care, emergency medicine, and other applications including adherence monitoring.(Costin et al., 2008)Some electronic monitoring that can provide adherence monitoring in telemedicine is event monitoring systems, which records the time and date when medication container was opened. The main drawback of this approach is that is very easy to cheat because it is not certain that the pill is taken when the container is opened. There are also a lot of m-health and alarm-based solutions which support the patient on accomplishing their treatment by providing reminders and interactive automated patient-specific messages.(Sabaté, 2003) Several studies state that electronic trackers or reminders used for adherence measurement showed 20% better adherence than medical controls, they also show some evidence that complex tools that include reminders and feedback for adherence may be more effective than simpler tools.(Normansell et al., 2017) In the next section we are going to introduce wearables: A useful tool that can be used as an m-health or alarm-based solution to measure and enhance adherence.

1.2 Wearable Devices in Medicine

A wearable system is a mobile electronic device that can be worn by the user as part of its outfit or as an accessory. Like mentioned previously an important application of wearable systems is that of monitoring physiological parameters during regular day activity.(Anliker et al., 2004)

Health care had been having an ongoing transformation where previously patient information was stored and processed by healthcare practitioners, hospitals, and other industry stakeholders which managed all medical processes. But every time data, information, and knowledge are more concentrated in centralized IT systems. The current system's structure along with an increase in chronic and noncommunicable diseases due to an aging society contributes to a lift on the health care system's cost. This situation had been creating a need for prevention and long-term care with cost-effective health services. In this context, wearable technology is contributing to identifying health risks, monitor disease progress, and provide treatment or advice outside clinics or hospitals.

Now relevant data can be acquired and monitored continuously for longer periods and without the need for a controlled clinical environment. Wearables are also bringing a new health information model, where data will no longer be centralized in a few places and where data will no longer be generated mainly by the hospitals and practitioners, but where providers, promoters, insurers, and device manufacturers will store the patient's health data and where the data will mainly be generated by wearable devices .(Amft, 2018) Self-tracking and digital health are global mega-trends, in fact, the annual growth of this marked it expected to continue rising dynamically with an annual growth rate of more than 20%.(Feldmann, 2017) According to Forbes wearable device sales will double by 2022, becoming a $27 billion+ market with 233 million unit sales.(Lamkin, 2018) More and more wearables will become an important part of daily life and essential for medical data collection.

Even when the wearable market is growing very fast lately, the development of this kind of technology starts in the late 1990s, where researchers engaged in developing technologies that could lead to significant improvements in the diagnosis and treatment of various diseases by performing long-term monitoring. This technology began growing in two separate major approaches to implementing it: wireless technology and e-textile solutions, which later was integrated into the wearable systems that we see nowadays which in many cases fulfill the aim of having at-home long-term monitoring for the subjects. The miniaturization of sensors, the availability of low-power radios and the development of dedicated operating systems was essential for the use of small sensor units and networks of sensor units in wearable technologies. These networks called body sensor networks, allowed researchers to monitor vital signs unobtrusively. The first medical applications for such devices were in cardiology, with photoplethysmographic sensors that could transmit data wirelessly, which then evolved to wristwatch type monitors. Advances in sensors like accelerometers, gyroscopes, and magnetometers, combined with progress in short-range communication technologies like, Bluetooth, ZigBee, RFID, and ultrawideband radio researchers and clinicians can now monitor movement patterns and even reconstruct trajectories of a subject. Meanwhile, the advances in e-textile pursued transforming the clothing items into an equivalent of a computer bus by attaching sensors. Some approaches embedded traditional sensor technology into clothes, while others attempted to integrate sensing elements into clothes using new materials and techniques to integrate sensors and fabric. In the end, both wireless technology and e-textile solutions are very promising for long-term monitoring outside hospitals, and both have a lot of applications such as recording

electrocardiogram data, monitor respiratory rate, track changes in blood oxygenation, and monitor sweat rate, energy consumption, muscle stretch, body temperature, body motion, and many other parameters. (Bonato, 2010) In fact nowadays is very usual to find different types of wearables such as wrist clocks, glasses, headbands, hand gloves, clothes, and many others not only for medical purposes but also for wellness and fitness.

In medical applications, we can divide the sensors employed for wearables into two main categories based on the type of signal they measure. One category focuses on real-time signal acquisition, which demands a high power consumption for the transmission of data because they are collecting continuous time-varying signals. This category can be called continuous signals which include accelerometers, gyroscopes, ECG sensors, electroencephalograph (EEG) sensors, electromyography (EMG) sensors, visual sensors, and auditory sensors. The other category collects signals that have a slower variation, so the amount of data that needs to be transmitted is much lower, this makes the power consumption lower because the device can use a sleep mode. This can be called of discrete signals which category includes glucose sensors, temperature sensors, humidity sensors, blood pressure monitors, and sensors monitoring blood oxygen saturation.(Lai et al., 2013) In table1.1 there is a summary of some sensors that are commonly used in wearable devices for medicine. Since this project used an IMU featuring accelerometers and gyroscopes it would fall into the continuous signal category, but differently from many of the devices that use sensors in this category, the designed device will try to reduce the power consumption by keeping the IMU sensor in sleep mode when the acquisition of the signal is not needed.

Table 1.1: Commonly used sensors in wearable technologies

(Lai et al., 2013)

Sensors	**Function**	**Signal Type**
Accelerometer	Obtaining acceleration on each spatial axis of three-dimensional space.	Continuous
Blood-pressure sensor	Measuring the peak pressure in the systole and the minimum pressure in the diastole.	Discrete
Carbon dioxide sensor	Measuring the content of carbon dioxide from mixed gas by infrared technique.	Discrete
ECG/EEG/EMG sensor	Measuring voltage difference between two electrodes which are placed on the surface of the body.	Continuous
Gyroscope	Measuring angular velocity of rotating object according to the principle of angular momentum conservation.	Continuous
Humidity sensor	Measuring humidity according to the changes of resistivity and capacitance caused by humidity changes.	Discrete
Blood oxygen saturation sensor	StartFragmentMeasuring blood oxygen saturation by absorption ratio of red and infrared light passing through a thin part of the body.	Discrete
Pressure sensor	Measuring pressure value according to the piezoelectric effect of dielectric medium.	Continuous
Respiration sensor	Obtaining respiration parameters indirectly by detecting the expansion and contraction of the chest or the abdomen.	Continuous
Temperature sensor	Measuring temperature according to the changes in materials' physical properties.	Discrete
Visual sensor	Capturing features of the subject, including length, count, location, and area.	Continuous /Discrete

An architecture for a body sensor network consists of small wireless nodes positioned on or inside the patient's body, that contains sensors and actuators providing timely data. These are called Wireless Body Area Networks(WBANs)(Movassaghi, Abolhasan, Lipman, Smith, & Jamalipour, 2014), these body sensor networks have different, network topologies, communication protocols, and standards. The topology of a network determines how different devices of the network are arranged and how they communicate with each other, while the protocols and standards are conventions between many parties that allow performing the communication. Right now we are going to detail the most important characteristics of some standards that are used for body sensor network communication.(Espina et al., 2014)

- Bluetooth: Operates in the 2.4 GHz with a data transfer up to 3Mbps band and is a a short range(ranging from 1 to 100 m) wireless communication standard, intended to maintain high levels of security.(Negra, Jemili, & Belghith, 2016)

- Bluetooth Low Energy: This is a standard that derives from Bluetooth itself and it is explained more deeply in subsection1.2.1.1.

- Zigbee: Operates in 868 MHz, 915 MHz, and 2.4 GHz frequency bands and is very useful for applications that require a low data rate,long battery life and secure networking.(Negra et al., 2016)

- IEEE 802.11:8 Is a set of standards for wireless local area network that operates in different bands including 2.4 GHz, 5 GHz, and 60 GHz. It is ideally suited for large data transfers by providing high-speed wireless connectivity and allowing videoconferencing, voice calls and video streaming.(Negra et al., 2016)

- IEEE 802.15.6: Defines three physical layer specifications, which are: Narrowband, Ultrawideband, and Human Body Communications. Each one operates at different frequency. Human Body Communications physical layer operates in the 21 MHz frequency band, characterised by 5.25 MHz bandwidth. The supported data rate ranges from 164 kbps to 1.3125 Mbps.(Espina et al., 2014)

There are also different types of nodes in a WBAN. A node is defined as an independent device with communication capability. And can be classified based on functionality, implementation, and role in the network. Functionality-wise a node in a WBAN can either be a personal device, a sensor, or an actuator. Implementation-wise it can be an implant node, a body surface node, or an external node. Also, it can assume the

role of coordinator, end node, or relay.(Movassaghi et al., 2014) The communication architecture is also separated into three different levels:

- Intra-WBAN communication: The network interaction of nodes and around the human body. Sensors are used to forward body signals to a personal server, located inside the Intra-WBAN.(Movassaghi et al., 2014)
- Inter-WBAN communication: This communication level is between the personal server and one or more access points. Inter-WBAN communication aims to interconnect WBANs with various networks, which can easily be accessed in daily life as well as cellular networks and the Internet. (Movassaghi et al., 2014)
- Beyond-WBAN communication: It includes the medical history and profile of the user and allows restoring all necessary information of a patient which can be used for their treatment. Depending on the application Beyond-WBAN communication can be between either the personal server and an access point, or between the personal server and GPRS/3G/4G.(Movassaghi et al., 2014)

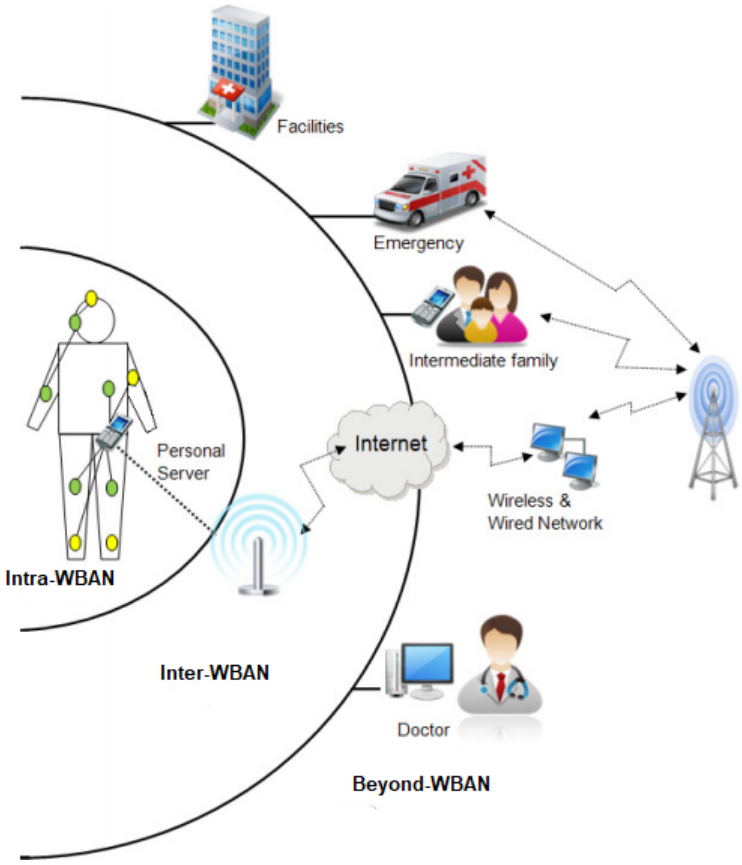

Figure 1.1: WBAN Architecture
(Movassaghi et al., 2014)

With everything mentioned above, we know that a wearable medical system encompasses many different components such as sensors, smart textiles, actuators, power supplies, wireless communication modules and links, control and processing units, interface for the user, software, and advanced algorithms for data extracting and decision making.(Pantelopoulos & Bourbakis, 2010) Then a basic architecture for the system can be one as depicted in figure1.2

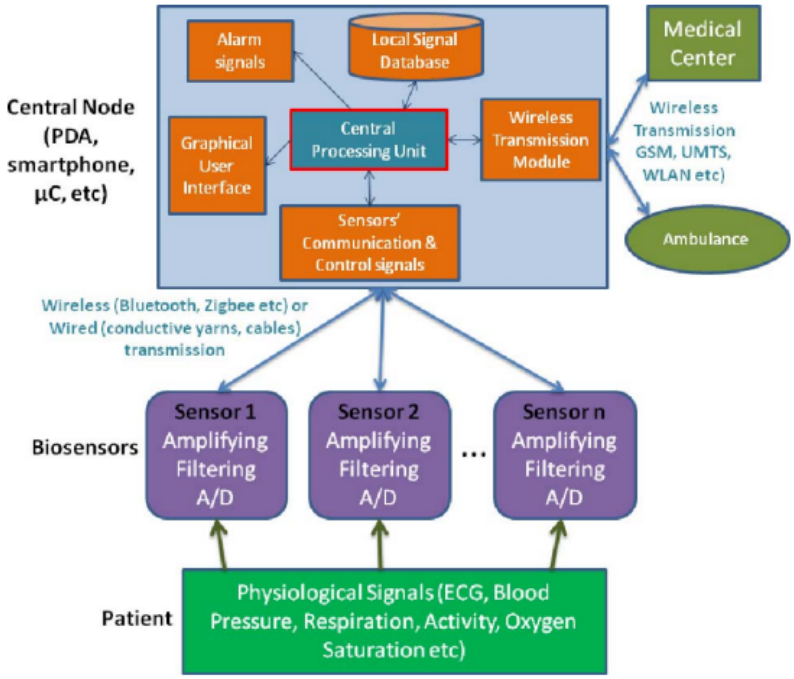

Figure 1.2: Architecture of a wearable health-monitoring system (Pantelopoulos & Bourbakis, 2010)

In the wearable market there are a lot of tracking devices for fitness. However there are very few commercial devices that are meant for analyzing kinematics and more specifically forearm kinematics, the table1.2 shows the specification three devices that can be used with for this purpose.

These wearables are not specific for adherence applications, but they provide forearm motion data. The Xsens DOT is an integrated sensor meant for human kinematic analysis and can be used as stand-alone or connected to up to five sensors and has a mobile app where the acquired signals can be visualized. The Metamotion is a clinical-grade sensing wrist band meant for motion analysis, it has a mobile application where multiple units can be used simultaneously (up to 7), and where the acquired signals can be visualized, it also offers the option to develop customized APIs. The E4 sensors integrates different types of sensors like PPG sensor, EDA sensor (GSR sensor), infrared

Device	Brand	Battery Duration	Data Transfer	Axes	Dimensions	Full Scale	App
DOT	Xsens	6h	BLE 5.0	3	36.30 x 30.35 x 10.80 mm	2000/s 16g 8 Gauss	Yes
MetaMotion	Mbientlab	No info	BLE 4.0	10	17 x 16 x 5mm	2000/s 16 g 810 uT 1100 hPa	Yes
E4 Sensors	Empatica	24 or 32+ h	BLE 5.0	3	44x40 x16 mm	No info	Yes

thermopile, 3-axis accelerometer and an event mark button that allows the user to tags events and link them to physiological signals, this wearable is not specific for motion tracking or analysis, but the motion signals can be retrieved via USB after being stored or can be monitored in real-time through a mobile app.

1.2.1 Embedded Systems

In this subsection it will be presented the concept of embedded systems and some items that were used to build up the project, mainly the communication protocols that were used within the devices and the type of sensor that was used.

An embedded system is a digital system, that uses hardware and software to perform a function, and works with minimal or no human interaction. Most of these systems interact directly with the process or the environment. Despite the applications an embedded system has two main components: hardware and software, and the interaction between them are also very relevant. The hardware includes a central processing unit (generally provided by a microcontroller) and peripheral devices such as displays, buttons, and different kinds of actuators and sensors that provide the system with parameters and variables with which it interacts with the environment and generates outputs in form of control actions. The software is on the other hand, is a series of programs, including the firmware which is the program that gives functionality to the hardware. The firmware is stored in a non-volatile memory that is not modifiable by users or by the environmental interactions with the system.(Jiménez, Palomera, & Couvertier, 2014) Microcontrollers are a very useful tool in embedded systems because they already contain components that allow it to operate stand-alone and being the control unit that is integrated into the system to perform all the monitoring and con-

trol tasks. In general, the architecture of such microcontroller includes a processor core (microprocessor or CPU), memory (register, data, and instruction), digital I/O, analog I/O(including ADC and DAC), interrupt controller, interface controller, timers, communication interfaces, among other functions. This architecture allows the chip to be embedded into other electronic or mechanical devices as a control module that needs very few other external devices to work.(Gridling & Weiss, 2007)

1.2.1.1 Bluetooth Low Energy (BLE)

In an embedded system, it is very important to choose the protocol that will communicate with the outside world; such protocol must suit the application and at the same time should work in the long-term. Here there will be explained why BLE was chosen over classic Bluetooth and some important features of this standard.

BLE stands for Bluetooth low energy, which is a standard for wireless data exchange between devices. This Bluetooth specification is very useful in mobile applications because it gives access to external hardware and provides reliable access to the devices in a mobile operating system. BLE is part of Bluetooth 4.0 focusing in having low power consumption, cost, bandwidth and complexity. One of its main differences with classic Bluetooth is that BLE was designed to serve as an extensible framework to exchange data, while classic BT is focused on a strict set of use cases. This allows BLE to be very usable and talk to different type of devices such as smartphones or tablets.

Like classic Bluetooth, BLE uses the concept of profiles to ensure interoperability between different devices, but these two standards are not directly compatible. To explain this incompatibility we should introduce the three main building blocks of every Bluetooth device:

Application: Being the highest layer of the protocol stack, is the one responsible for containing the logic, user interface, and data handling of everything related to the actual use-case that the application implements. The architecture of an application is highly dependent on the implementation.

Host: The upper layers of the Bluetooth protocol stack. In the Bluetooth Low Energy the host includes the following layers:

- Generic Access Profile (GAP)
- Generic Attribute Profile (GATT)

- Logical Link Control and Adaptation Protocol (L2CAP)
- Attribute Protocol (ATT)
- Security Manager (SM)
- Host Controller Interface (HCI), host side.

While in the classical Bluetooth it includes the following layers:

- Serial Port Profile (SPP)
- Radio frequency communication (RFCOMM)
- Logical Link Control and Adaptation Protocol (L2CAP)
- Host Controller Interface (HCI), host side.

Controller: The lower layers of the Bluetooth protocol stack, including the radio. In the Bluetooth Low Energy the controller includes the following layers:

- Host Controller Interface (HCI), controller side
- Link Layer (LL)
- Physical Layer (PHY)

While in the classical Bluetooth it includes the following layers:

- Host Controller Interface (HCI), controller side
- Link Manager (LM)
- Physical Layer (PHY).

To make it clearer the figure1.3 illustrates the protocol stacks for each standard.

A Bluetooth Low Energy device can communicate in two ways: broadcasting or by connections. In this case we will focus in connections because it is the communication that suits best for this solution, given that we will be needing to transmit data in both directions as it will be explained later on, in the hardware design. A connection is a permanent, periodical data exchange of packets between two devices. This makes a

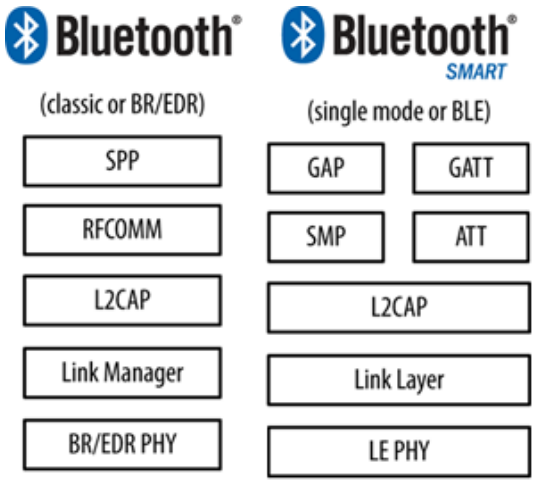

Figure 1.3: Bluetooth versions configurations
(Townsend, Cufí, Akiba, & Davidson, 2014)

connection to be private by definition, because a peering must be done to send and receive data through the connection. Connections involve two roles: central or master, and peripheral or receiver. The master is the one that initiates the connection, so it scans the frequencies for connectable advertising packets repeatedly until its suitable to connect to another device. To initiate a connection, the master picks up the connectable advertising packets from a slave device and then sends a request to the slave to establish an exclusive connection between the two devices. After the connection is established, the master manages the timing and initiates a bidirectional data exchange. Meanwhile, the slave periodically sends advertising packets and accepts incoming connections. After the accepting the masters request, the connection is established, and the slave stops advertising. The slave follows the timing of its master and exchanges data. Both devices can begin exchanging data in both directions, data can be sent independently either by the master or slave device during each connection event, and their roles do not imply a priority or restrictions in the data exchange.(Townsend et al., 2014) In this case the connection selected as so to perform transparent UART between the master and the slave, even though this approach is not significantly less power consuming than traditional Bluetooth protocol.

1.2.1.2 Inertial Measurement Units and MEMS

Inertial measurement unit (IMU), is a sensor package that integrates at least two inertial sensors: accelerometer, gyroscope, and magnetometer. It can measure gravity,

angular rate, and can also act as a compass. Each sensor may be composed from 1 to 3 axes, most IMUs use 3 axes devices. So, it can be used to track the movement and orientation of objects. The most interesting feature of IMU sensors over other kinds of sensors is that they can integrate different technologies sensors such as MEMS into one single device that also includes on-chip ADC, programmable digital filters, and programmable interrupts; this makes the data extraction easier. (Kok, Hol, & Schön, 2017) Since IMUs are noninvasive can be easily integrated to a wearable strap, which makes them an appropriate sensor to use for healthcare and rehabilitation wearable technologies, allowing the assessment and monitoring of activities of an individual, enabling an evaluation of their quality of life.(X & C.Menon, 2018)

The kinematics of the human movement can be obtained from the signals of 3D inertial measurement units. (Luinge, 2002)The IMU chosen for this project uses microelectromechanical systems (MEMS) motion tracking sensors, which measures the acceleration or angular velocity with small masses and springs that change the capacitance of the system as they move. MEMS components are small, light, inexpensive, have low power consumption, and short start-up times. Their accuracy has significantly increased over the years, for this reason, they are very suitable to track human motion.(Kok et al., 2017)

1.2.1.3 I^2C Protocol

I^2C is a broadly implemented serial protocol used for two-wire communication between a master or multiple masters and single or multiple slave devices. This protocol was originally designed by Philips to link a small number of devices on a single circuit board, but nowadays it can also connect devices in different boards via cable and has a communication speed up to 3.4 Mbit per second. Two wires, serial data (SDA) and serial clock (SCL) carry information bidirectionally between the devices connected to the bus. Each device is recognized by a unique address and can operate as either a transmitter or receiver, depending on the function of the device. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfers. A master is a device that initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

Table 1.2: Definition of I^2C-bus terminology(Phillips, 2000)

Term	Description
Transmitter	The device which sends data to the bus
Receiver	The device which receives data from the bus
Master	The device which initiates a transfer, generates clock signals and terminates a transfer
Slave	The device addressed by a master
Multi-master	More than one master can attempt to control the bus at the same time without corrupting the message
Arbitration	Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

These relationships are not permanent and only depend on the direction of data transfer at that time. In this application we will not be dealing with multi-master configuration so we are not going to deepen into this specification.

All masters generate their own clock on the SCL line to transfer messages on the I^2C-bus. Data is only valid during the HIGH period of the clock. Since in this project we are working with a single master we will not deepen in the synchronization and arbitration process of I^2C protocol, but we will introduce the handshaking process.(Phillips, 2000)

START and STOP Conditions

START and STOP are conditions that indicate whether a bus is busy or not. After having a START condition the bus is considered to be busy and after having a STOP condition the bus is considered to be free. These conditions are always generated by the master. The START condition is given when the SDA wire is in high to low transi-

tion and SCL is high, while STOP condition is given when SDA wire is in low to high transition and SCL is high, as shown in Figure 1.4.

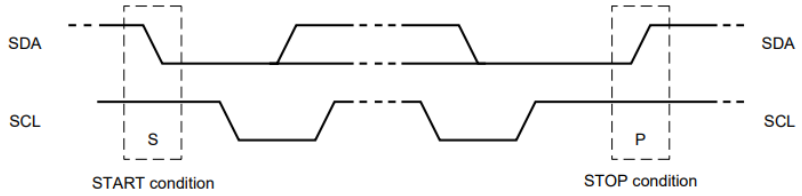

Figure 1.4: START and STOP conditions in I^2C (Phillips, 2000)

Data Transfer

Messages may have and arbitrary structure and the number of bytes transmitted per transfer is unlimited, but the length of the bytes in the SDA line must be of 8-bits and the data is transferred with the most significant bit (MSB) first. Each byte must be followed by an acknowledgment bit, which indicates the receiver has accepted or rejected the message. An acknowledgment is given when the receiver pulls down the SDA line so that it remains in LOW during the HIGH period of the clock pulse, as shown in Figure 1.5.

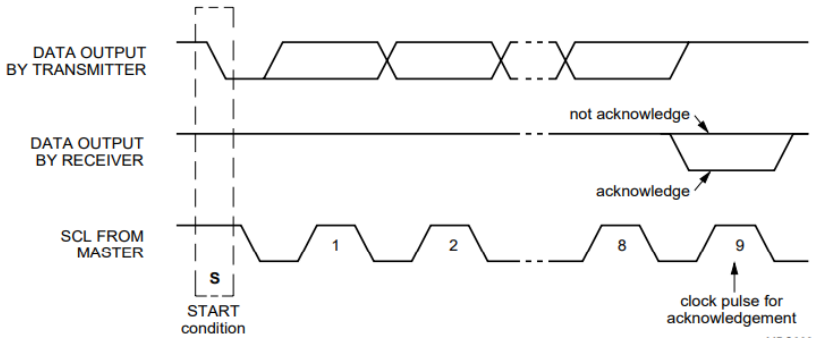

Figure 1.5: Acknowledge in I^2C (Phillips, 2000)

When a slave does not acknowledge the slave address, the data line must be left HIGH by the slave. After that, the master can generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer. If the slave-receiver generates the not-acknowledge on following bytes, it means that for some

reason is not capable of receiving any more data bytes and the master will also generate either a STOP condition or a repeated START condition. If a master-receiver is involved in a transfer, it must show it is the end of the data to the slave- transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.(Phillips, 2000)

Handshaking Process

The handshaking is a process between participants in the communication which establishes the communication protocol. In the case of I^2C, the handshaking is performed as follows:

1. Master start communication: Asserts START condition on the bus
2. Master transmits the address byte which is composed by a 7-bit address and a read (1) or write (0) bit(R/W). This 7-bit address correspond to the slave address (each slave should have a unique 7-bit address)
3. Master releases the SDA line putting it on HIGH.
4. (a) If the address matches a receiver, the receiver pulls down SDA line to LOW for giving acknowledgement (ACK) and then releases the line. Then the master reads the ACK bit and continues to the R/W operation.
   (b) If it doesn't match with any receiver there is no acknowledgement (NACK). The master can either assert a STOP bit or a repeated START.

(Phillips, 2000)

1.3 Problem Definition

This project aims to contribute to the monitoring of human activities as an adherence assessment measure, that can be detected through the movement of an arm, by means of an accelerometer and a gyroscope of 3 axes each.

In clinical practice, several ways to measure or monitor adherence are already available. It can be done through subjective measurements that include self-report and health-care professional assessments using questionnaires, but in this approach, patients tend

to under-report non-adherence to avoid disapproval from their health-care providers. It can also be done through objective measurements, such as pill counts, electronic monitoring, secondary database, and biochemical measures. But even objective measures are not completely infallible, because patients can cheat, so it is suggested to use several approaches in combination. In the electronic monitoring category, there exist many motion tracking and gesture recognition devices.

1.4 Aims and Scope

- Design and build a hardware device that can acquire accelerometric signals from the forearm and send them to a smart-phone. In order to allow the data to be processed for gesture tracking, the device is connected to a smart-phone application that sends the data to the cloud. For this general aim, two states have been defined for the device: A state in which it acquires the signal and a sleep mode state, where it must remain while the acquisition of the signal is not needed.
- Design a smart-phone application that can receive signals from the hardware and serves as a bridge to send data to the cloud.
- Inside the application, design an environment in which the hardware device can be programmed so that it can make acquisitions of different duration. The duration can be determined according to the need of the training routine or recommended activity and can be set to change at different times of the day, which can be programmed by the user giving them different labels.
- Discuss the applicability in the market of the designed device.

2. Materials and Methods

2.1 Solution Characterization

- Hardware solution design:

  Different sensors and wireless communication modules were evaluated to choose the one with most suitable characteristics for wearable and human gesture recognition devices, and in particular, to our application. We chose the IMU sensor ICM 20600, and the BLE module RN4871 and the micro-controller ATTiny1617. With these components an electronic circuit was designed including the power supply, buttons and outputs to communicate with the user, debugger and programmer interface input for the embedded system design and PCB layout. The device was be programmed to comply the requirement of low energy sleeping mode, and acquisition mode. The software used to model the PCB was Altium Designer 19.0.14 . To program the components the software MPLAB X IDE v5.20 in complement of the debugger and programmer PICKit4 were used.

- Smart-phone app design:

  The app will be designed to have two different modes with their respective environment: One environment will be designed for the training phase, where it aims to acquire data regarding to an activity with a given label such as, eating or drinking. The other environment will be designed for tracking the labeled gestures as defined in the training phase; to achieve this, in the environment the user will be able to set several alarms with their respective label and acquisition duration. When this alarm activates the app will send a flag to the coupled device indicating the acquisition should start; when the acquisition duration of the alarm is over the app will send a flag to the device to make the acquisition

stops. In both environments, the acquired data will be uploaded and saved in the cloud. We will not address the signal processing and data analysis to treat the obtained data. The database along with the stored routines were developed with the relational database management system. The app is not implemented, but a simpler app was developed to test the hardware.

- Bracelet design:

  The bracelet was designed to be safe and comfortable for the user and to protect the circuit. The bracelet is meant to be worn in the user's wrist.

- Testing:

  A testing method is suggested for the different devices used in the bracelet and the data it acquires.

2.2 Solution Development

2.2.1 Hardware Design

In this section, we will explain how the problem was defined and approached hardware-wise, including context analysis, the block diagram of the circuit, the circuit configuration along with the boards schematic and the state machine that represents the programming of the embedded system, how the circuit was built, how it works and the final result of the designed circuit.

2.2.1.1 Context Analysis

The device starts a data acquisition from a 6-axis IMU including a 3-axial accelerometer and a 3-axial gyroscope. When it receives a specific signal from the smartphone that indicates an alarm has been triggered. Right after, the data from the IMU sensor is transmitted to the smartphone via Bluetooth. The smart-phone will transfer the information to the cloud where it is stored in a database. Subsection 2.2.3 will explain the data transfer and storage in detail. When the acquisition is done, the smartphone sends another signal to the device indicating it to stop the acquisition. To save energy, when the device is not acquiring data, most of the components will be in sleeping mode if possible. The device has some LEDs to indicate the status of the state machine that

would let the user know if the device is working properly. The device also has a reset button, for restarting the device at any given moment.

2.2.1.2 Block Diagram

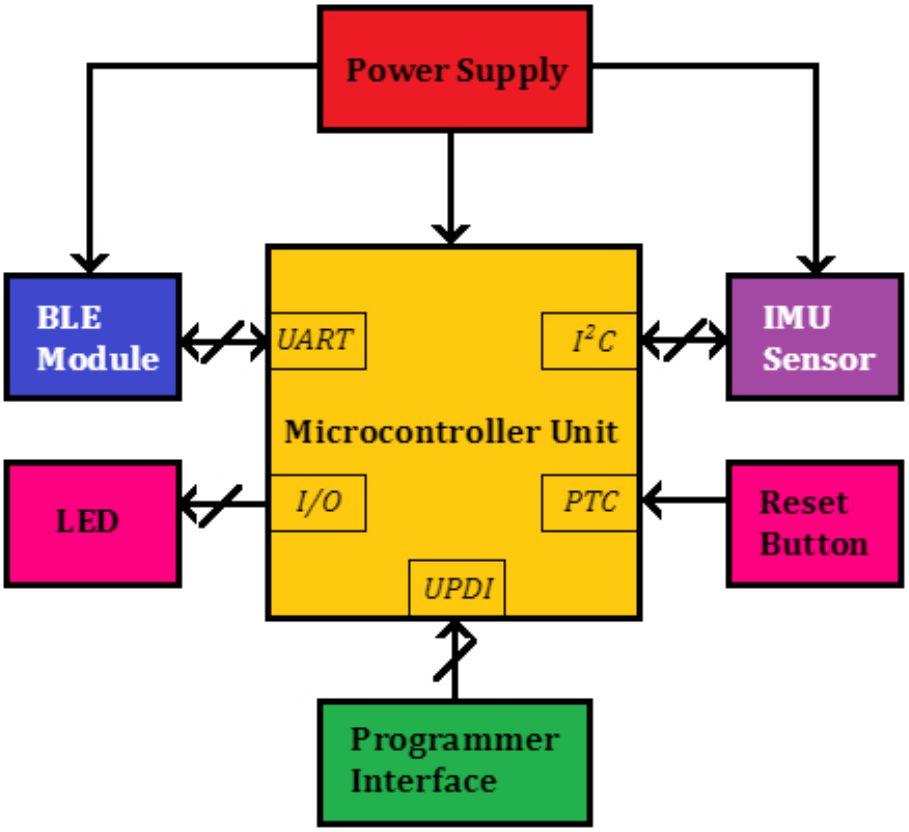

Figure 2.1: Block Diagram

The blocks that build up the device and the connections between them are shown in figure 2.1. The device is powered by a supply voltage of 3.3V which should have a switch to turn on and off the device. The RN4871 BLE module is connected via UART to the ATTiny1617 microcontroller unit, this connection fulfills the functions of control and data exchange. The ICM 20600 IMU sensor is connected to the MCU through the I^2C ports and performs its communication through this protocol. In this way, the MCU works as a link between the IMU and the BLE module when acquiring and transmitting the data captured by the sensor. The LEDs is connected to the input/output pins of the MCU that will light up for 3 seconds during the initialization of the device and then will blink indicating the device is in the acquisition, the other LED is connected to the BLE and it blinks differently depending on the BLE connection state with other

devices. The device final board has a header that connects the MCU with the debugger and programmer PICKit4 so that it can be programmed a debugged on-chip. Each block will be explained in more detail in the following subsection, Circuit Configuration.

2.2.1.3 Circuit Configuration

For visualization purposes, the circuits schematic is presented in four different figures. Figure 2.2 represents the MCU connections, figure 2.3 represents the BLE module connections, figure 2.4 represents the IMU sensor connections and the figure 2.5 represents the power supply for the circuit. The links among the different figures are represented by ports that show the data connections between the components, and nets that represent the power connections between the components.

Microcontroller Unit

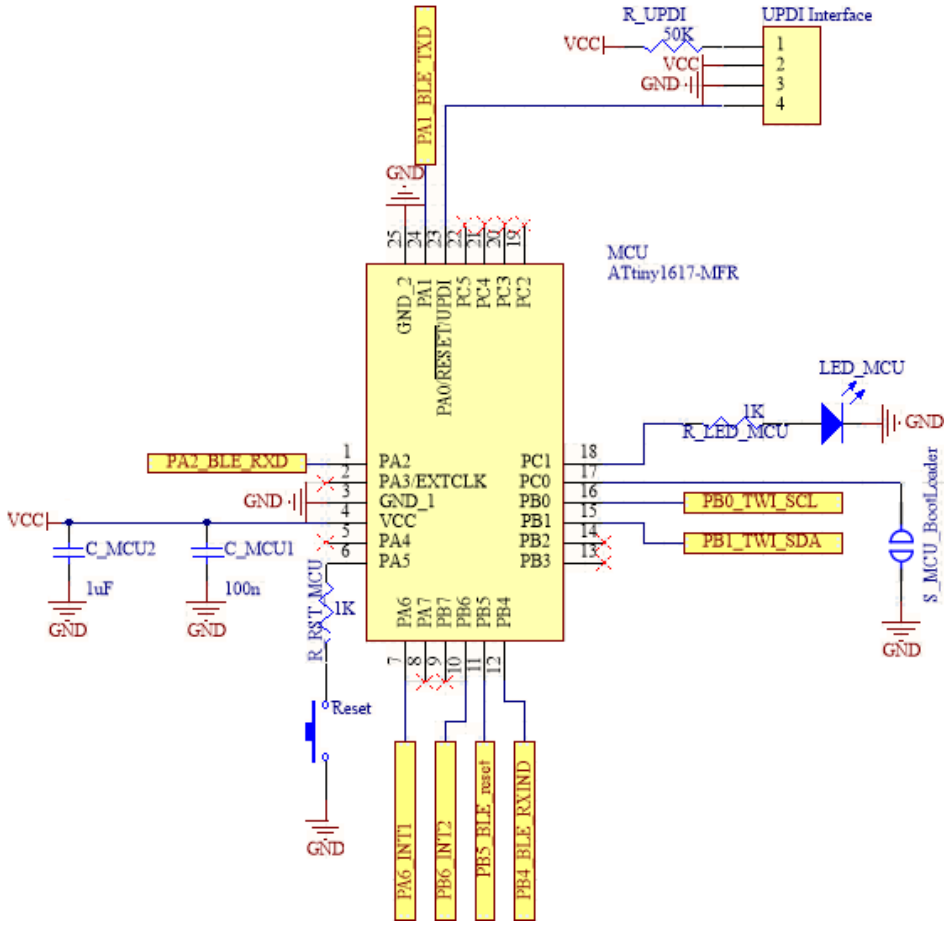

Figure 2.2: Schematic for the Microcontroller Unit

The microcontroller unit manages the state machine and performs the communication between the sensor and the Bluetooth module, and between the device and the user. It

communicates via UART to the BLE module which connections are better explained later in the document and communicates via I^2C with the sensor which is also better explained later in the document. It has a reset button connected to the peripheral touch controller, which includes an internal capacitor that avoids debounce. It also has a LED that gives an idea to the user of which state of the state machine the device is in.

During the system initialization or re-initialization, the LED will be off, then on for 2 seconds, and then it will be off again. While the device is waiting for a Bluetooth connection it will be off, then when it is connected but not acquiring it will blink, and when acquiring it will be on.

The UPDI interface is necessary for the MCU debugging and programming, this is performed with the MPLab PICkit 4.

The bootloader is there for an emergency where no UPDI compatible programmer is available, it is used for device self-programming. This enables the user to download application code into Flash without the need for an external programmer. For further understanding in this feature, you may refer to the ATTiny1617 Datasheet or to the application note AN2634 (Microchip, 2018).

Bluetooth Low Energy Module

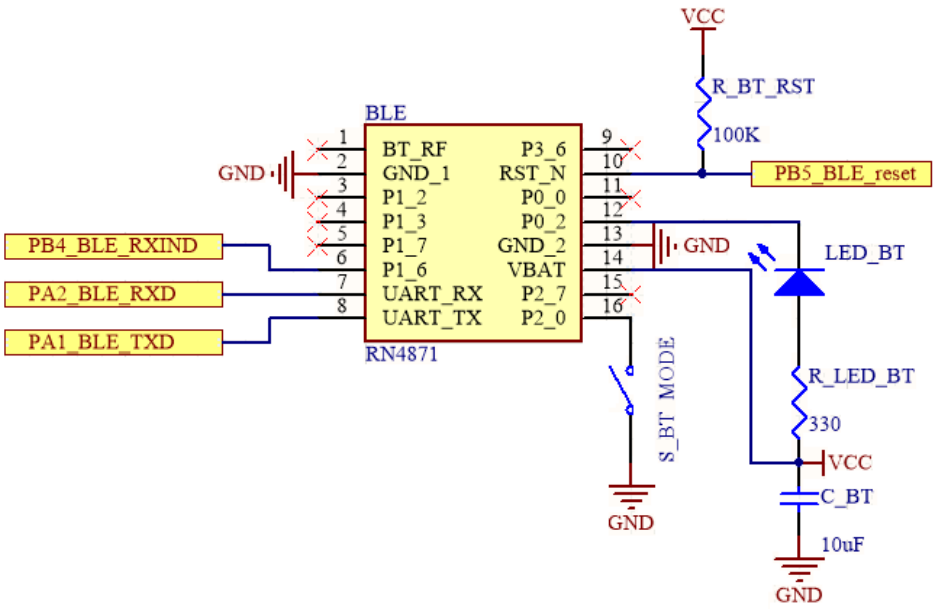

Figure 2.3: Schematic for the Bluetooth Low Energy Module

The RN4871 performs the communication with the microcontroller unit via UART,

Signal name	MCU pin	BLE pin
PA1_BLE_TXD	PA1	UART_TDX
PA2_BLE_RXD	PA2	UART_RDX
PB4_BLE_RXIND	PB4	P1_6
PB5_BLE_reset	PB5	RST_N

Table 2.1: Pin Map for the BLE Module

thus the reception and transmission UART pins are connected to the transmission and reception USART pins from the ATTiny1617 as illustrated in the figure 2.3 and specified in the table 2.1. This connection allows the BLE module to communicate with the microcontroller unit when the alarm is activated. This alarm indicates the microcontroller to start the acquisition with the IMU, it also allows the microcontroller unit to send the acquired signal to the smartphone via the BLE module.

The P1_6 pin is connected to analog GPIO of the MCU which can change the power mode, enabling or disabling the UART low power operation.

The P2_0 pin selects the working mode. Internally is pulled up by default, which puts the module in application mode. So, when the switch is closed, it is pulled to low and it switches to test mode.

The LED indicates the mode of the Bluetooth device either if it is in ON/OFF mode.

The switch is the system configuration input. This pin is configured as input pulled-high by default. So, when the switch is open, the device is in application mode and when is closed the device is in test mode/Flash update/EEPROM configuration.

Inertial Measurement Unit Sensor

Signal name	MCU pin	IMU pin
PB0_TWI_SCL	PB0	SCL/SCLK
PB1_TWI_SDA	PB1	SDA/SDI
PA6_INT1	PA6	INT1
PB6_INT2	PB6	INT2

Table 2.2: Pin Map for the IMU Sensor

The LSB of the I^2C slave address is set by pin 1 (AD0). In this project, we are using only one ICM-20600, so this pin can either be pulled up or down. In this case, it is pulled down as shown in the figure 2.4 which makes this bit to be 0, the full address being 1101000.

In this case, we are using I^2C protocol so the frame synchronization digital input and

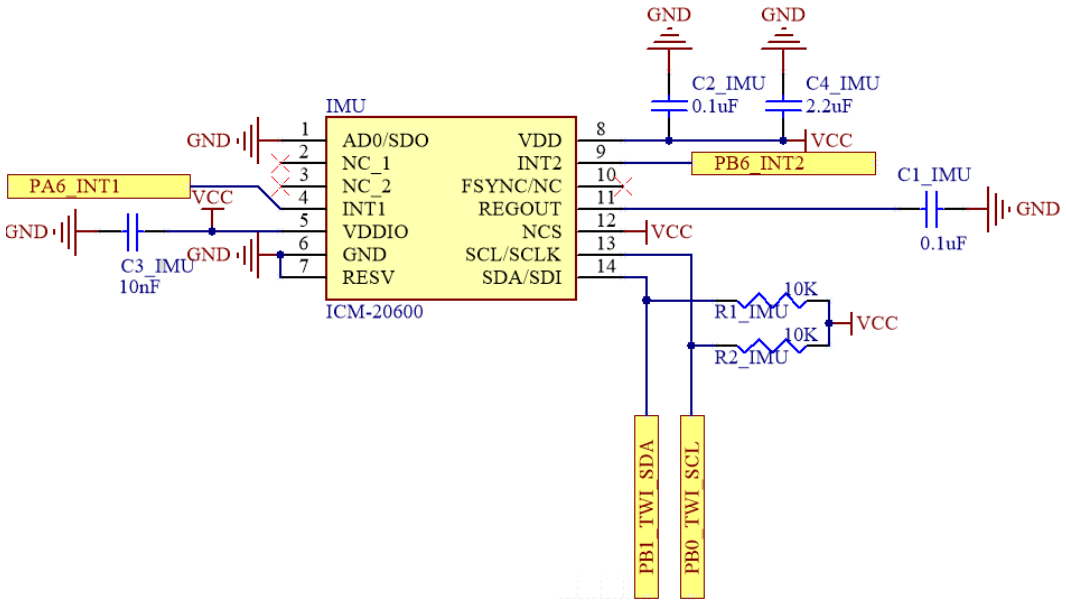

Figure 2.4: Schematic for the Inertial Measurement Unit Sensor

the chip select are left unconnected, while the SCL and SDA pins are connected to the corresponding two-wire interface (TWI) pins in the MCU as shown in the figure 2.4 and specified in the table 2.2. The I^2C lines are open drain and pull-up resistors are required, in this case, 10kΩwere chosen, following the recommendations given in the data-sheet.

INT1 and INT2 pins belong to the interrupt configuration register, where interrupt functionality is configured to allow the interrupt triggering. These pins are connected to I/O port pins from the MCU.

Power Supply

The power supply has as a source a 3.7V, 250mAh LiPo battery cell. Since the nominal supply voltage for this circuit is around 3V, the battery is connected to a 5-3.3V LDO regulator which is connected to the circuits VCC supply voltage. For safety and practical reasons, the current can be interrupted by a switch that can turn the device on and off as shown in the figure 2.5. This switch with the designation S_POW is an On-On type switch, that connects the battery with the charging system or the functional circuit, this allows that the charging circuit will never be directly connected to the device, which is safer for both the user and the device. For the charge management system, a MCP73831 is used, with a Micro-USB connector where the energy will be supplied to the system. To charge the battery chosen, a current of 300mA. The programming resistance was chosen as 3.3kΩ, following the current regulation set from the charge

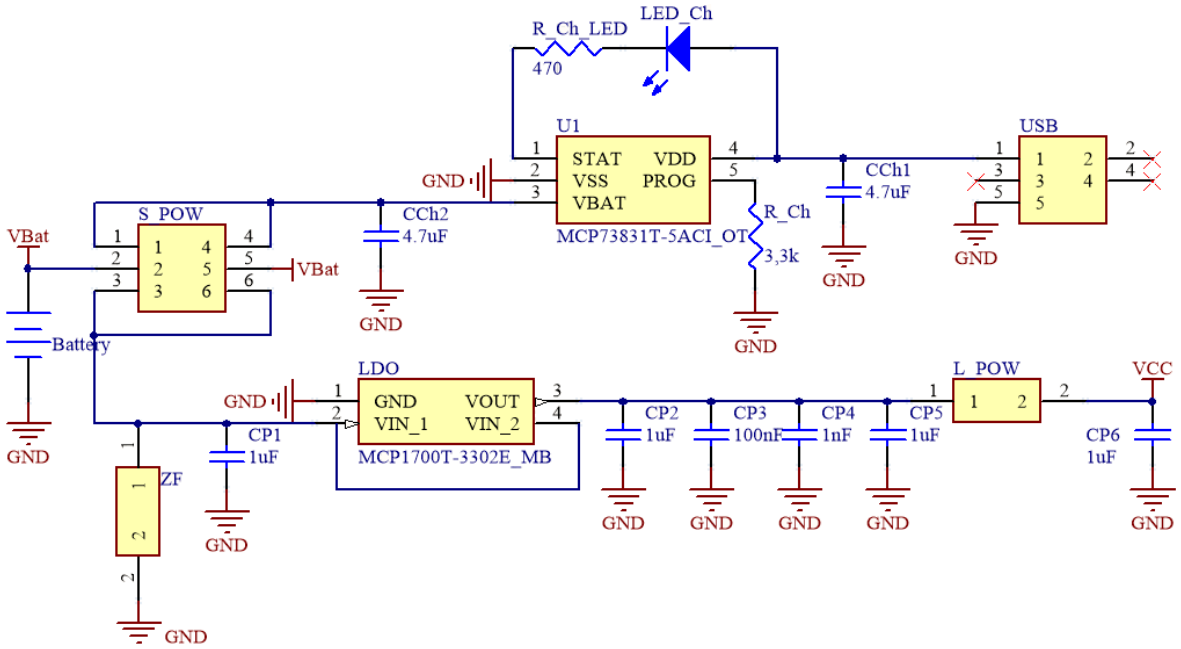

Figure 2.5: Schematic for the Power Supply

management controller, shown in the equation 2.1.

$$I_{REG} = \frac{1000V}{R_{PROG}} \tag{2.1}$$

Where, R_{PROG} is given in kΩand I_{REG} is given in mA.
With this chosen supply current and a battery with an electric charge of 250mAh, the time the device would take to be fully charged is:

$$T_{charge} = \frac{250mAh}{300mA} = 50minutes \tag{2.2}$$

In the section 2.2.5 there is a description in how to determine the charge duration of the working device.

2.2.1.4 State Machine

the flow of the embedded system state machine is depicted in figure 2.6. The black dot represents the state 0 which is the initialization state. Once the device is turned on, the ports are declared and all the initial conditions for the device are settled. The user can recognize the device has performed the initialization when the led turns on for 3 seconds. Right after the initialization is performed the device transitions to the state 1 that was called C0to1. All the other state and transitions descriptions shown in figure2.6 are explained in the tables 2.3 and 2.4.

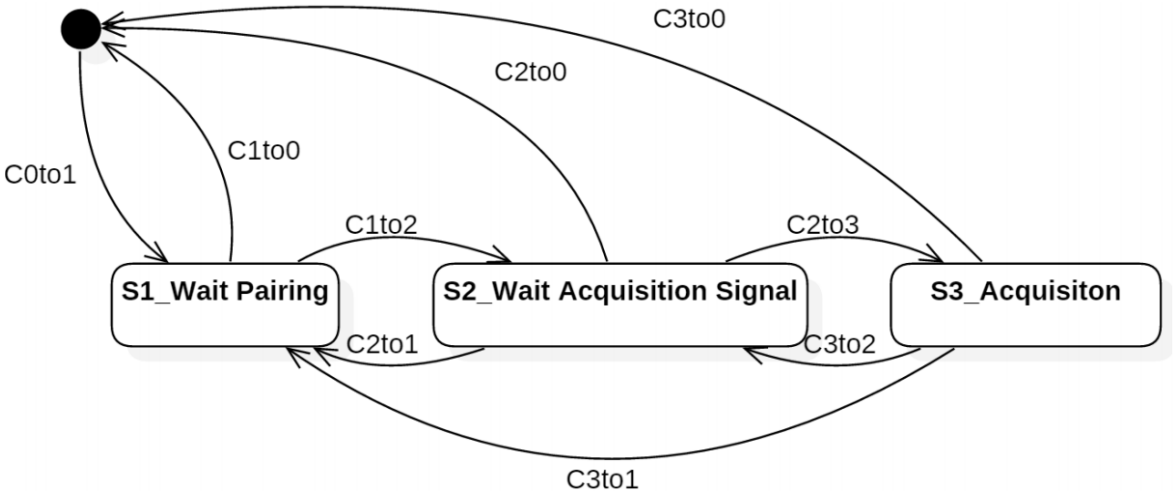

Figure 2.6: State Machine Diagram

Table 2.3: States description

State	**Description**
State 1: S1_Wait Pairing	The sensor is held in low power operation mode until the Bluetooth is paired with another device.
State 2: S2_Wait Acquisition Signal	The Bluetooth module is connected. The sensor is in low power operation mode, waiting to receive a signal from the smartphone to start the acquisition.
State 3: S3_Acquisition	The sensor normal power mode. The data is acquired by the sensor and sent to the smartphone via Bluetooth.

Table 2.4: State transitions description

State	Description
C1to0	Being in state 1, the reset button was pressed.
C1to2	Being in state 1,the device got paired with another BT device.
C2to0	Being in state 2, the reset button was pressed.
C2to1	Being in state 2, the device got disconnected from the BT device it was paired to.
C2to3	Being in state 2, the device receives an signal from the smartphone which indicates the beginning of an acquisition.
C3to0	Being in state 3, the reset button was pressed.
C3to1	Being in state 3, the device got disconnected from the BT device it was paired to.
C3to2	Being in state 3, The time of acquisition has already passed or the device receives an signal from the smartphone which indicates the end of the acquisition.

2.2.2 Hardware Implementation

2.2.2.1 PCB

The circuit board was designed with Altium Designer 19.0.14 and the final layout can be appreciated in figure 2.7 and it was printed and soldered by an external company. The final circuit is shown in figure 2.8.

In the figure 2.9 the shape and dimensions of the final PCB. Where A=36.07mm, B=27.05mm, C=15.62mm, D=9.27mm, E=2.16mm and F=3.94mm, having a total length of 40.01 x 27.05mm, with a thickness of 1,55mm.

2.2.2.2 Device Programming

Even though state machine programming is a matter of software, it is included in this chapter because is relevant to hardware development. To perform the state machine the software MPLAB X IDE v5.30 was used, along with the compiler XC8 v2 and the MPLAB Code Configurator v3. To upload the state machine program it was used the PICKIT 4 in-circuit debugger. All the code developed to program the microcontroller can be found at the appendix in the section 4.1.

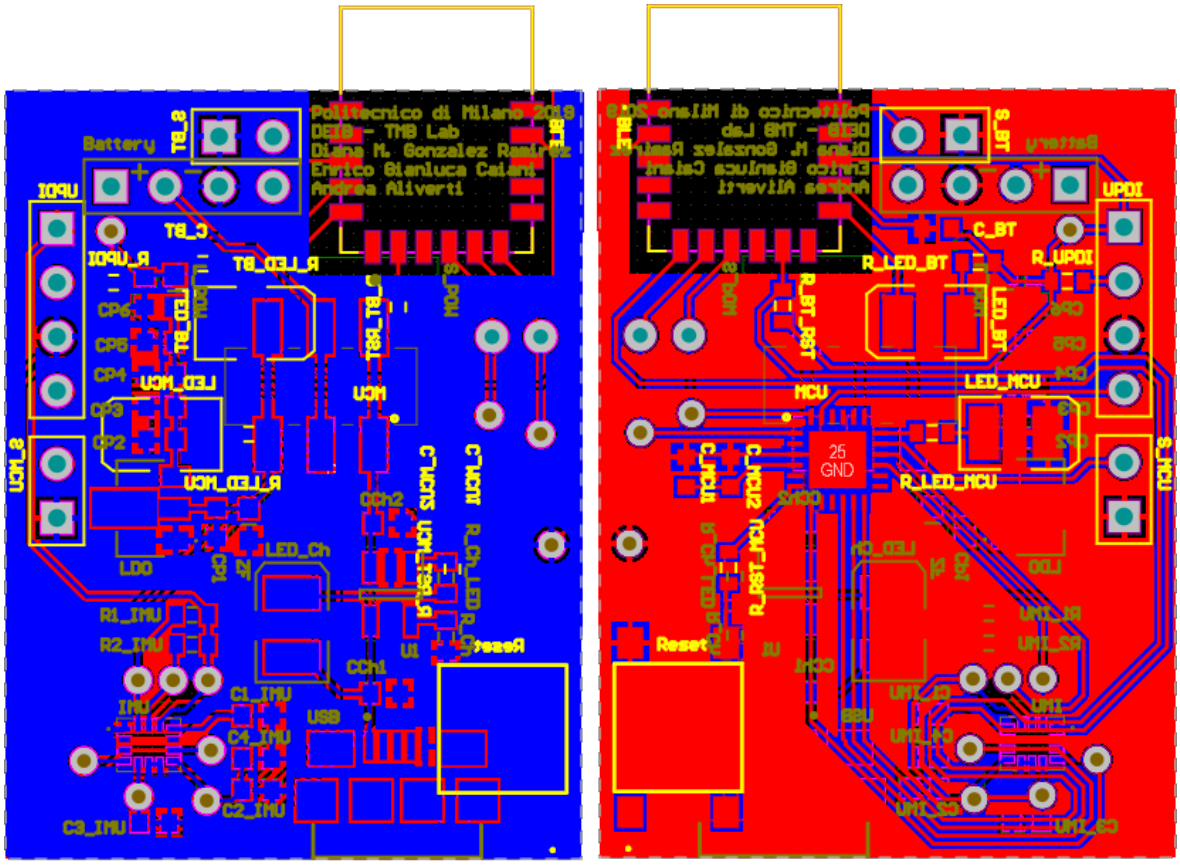

Figure 2.7: PCB Layout

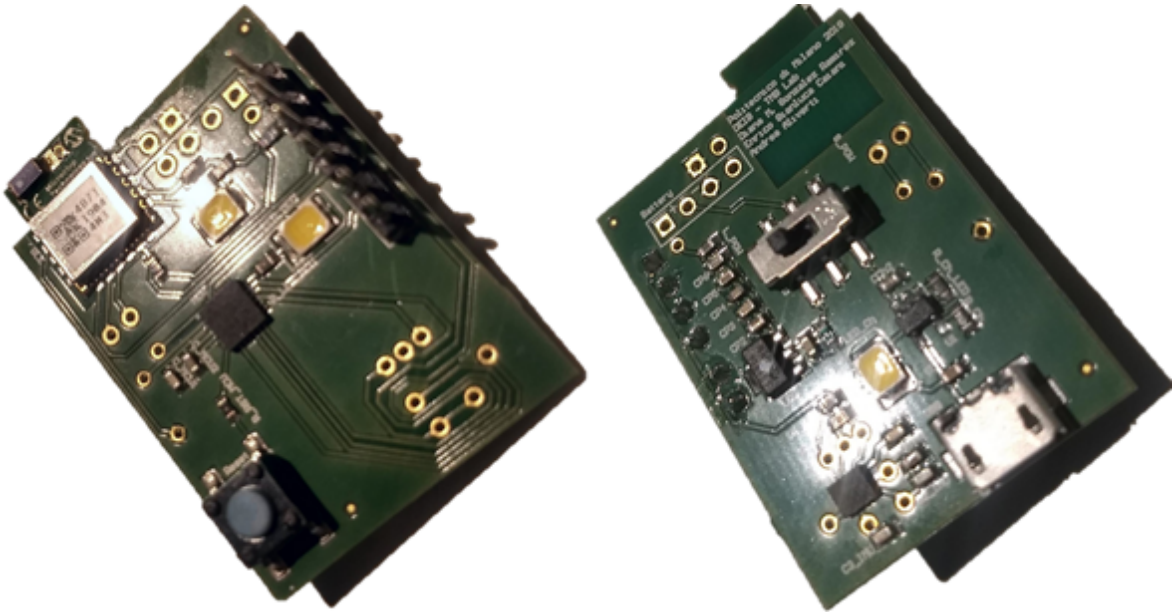

Figure 2.8: Final Circuit Board

The state machine algorithm makes the sampling rate of the system to be 100Hz because of the clock configuration and the interrupt service routine.

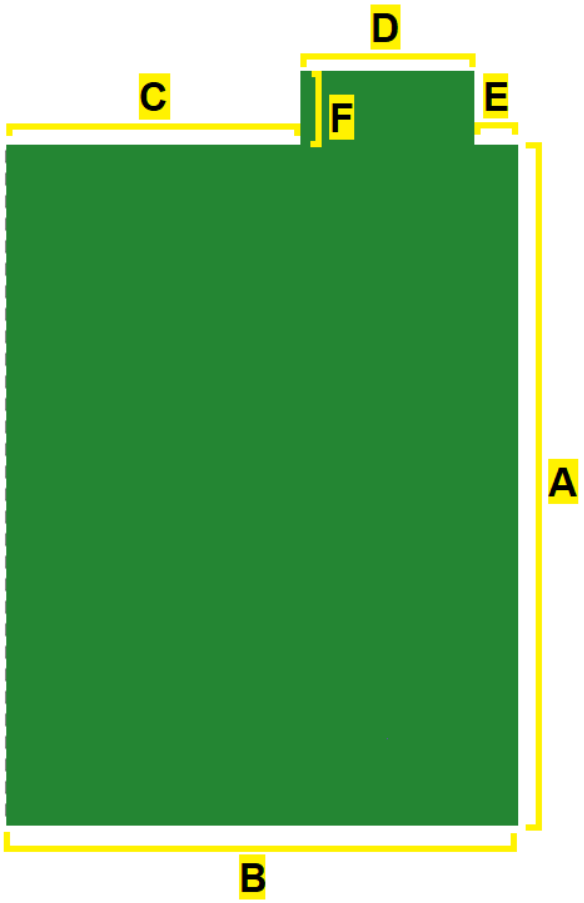

Figure 2.9: Circuit board dimensions

Before loading the state machine to the MCU, the BLE module's firmware was updated through a USB to TTL serial module, and the following configuration was performed using ASCII commands using the same module and the software TeraTerm:

$$$, this command enters command mode

+, this command turns on the echo

SF,1, this command resets the module to factory configuration

SN,ArmTracker, this command sets the module name to "ArmTracker"

SS,C0, this command set the characteristic to support device info and UART transparent services

SR,0100, this command set the service to enable UART transparent without ACK

R,1, this command makes a reboot to make the changes effective

SB,09, this command sets the BAUD rate to 9600

2.2.2.3 Bracelet Design

For the bracelet, a case for the circuit was designed using Fusion 360. The case protects the circuit and, on the sides, has two handles where a strap should pass. The strap will be adjusted to the subject's wrist.

The dimensions of the box are 45 x 34 x 15mm, where the thickness of the material is 4mm; the lid of the box is 45 x 34 x 4mm; and the handles are 3.5 x 24 x 6mm, where the thickness of the material is of 1mm. These dimensions where chosen to fit the circuit board with all the mechanical components and to fix the battery that has dimensions of 8 x 33 x 19mm.

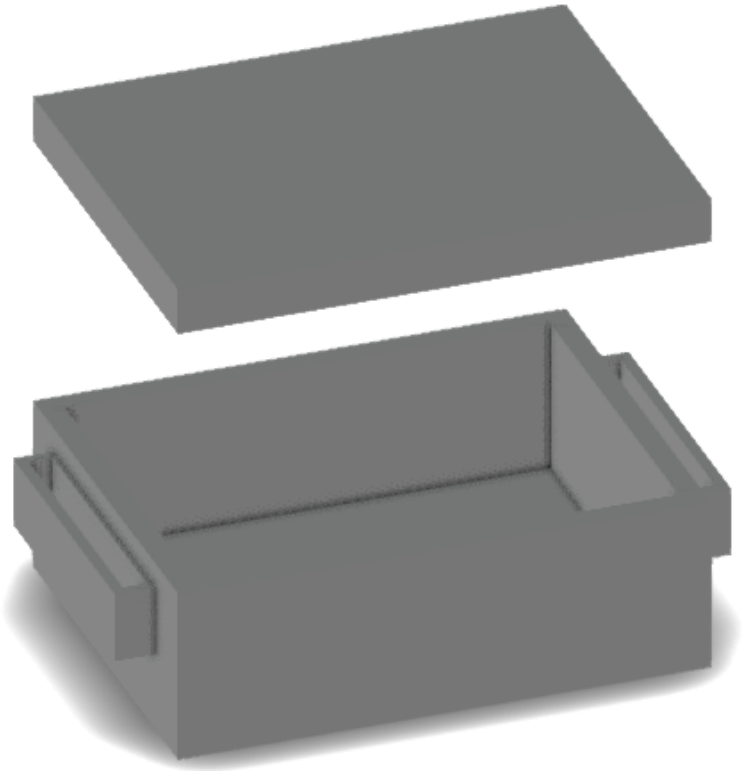

Figure 2.10: Case 3D model

2.2.3 Software Design

In this section we will explain how the problem was defined and approached software-wise, including a context analysis; the system's use case scenario, activity diagrams; the database class diagram, entity-relationship diagram, technical description of tables, and queries; and the app environments overview. In addition, it will explain overall how the system was built by means of a database and a smartphone-app, and how this system works.

2.2.3.1 Context Analysis

This system will be able to monitor the forearm movement of a subject through a wearable device. The system should incorporate a database where we are able to store information related to the users and the measurements and manage all of them. The system's interface is adapted for two types of users: regular app user and expert user. Each type of user has specific interactions in the system. The expert user specifies the gestures to be tracked and the relevant training mode, meaning it can create and train gestures as well as managing the alarms for the regular app users. The regular app user can train the already existing gestures and set alarms according to when the medication is needed to be taken (the alarms will have a space for adding a note in case the user wants to specify something, for example, which medication is supposed to take at that time.)

2.2.3.2 Use Case Scenario

Actors:

1. Regular User
2. Expert User

Regular user:

Figure 2.11: Use Case Diagram Regular User

1. Registers
2. Logs in
3. Pair Bluetooth
4. Train an existing gesture
5. Set an alarm
6. Modify an alarm
7. Delete an alarm

Alternative Scenario:

1. (a) Register fails: There's another user with that username, password confirmation failed or missing information.
   Shows a warning for changing the username, checking the password, or complete the missing information.
2. (a) Login fails
   Ask for username and password again
3. (a) Bluetooth pairing fails
   Show warning: user can continue: ask the user to try the pairing again.
4. (a) There are no gestures to train yet
   Show warning: There are no gestures yet, contact an expert
   (b) The recording failed
   Show warning: Ask the user to perform the recording again.
5. (a) The alarm overlaps with another one
   Show warning: Ask the user to change the time or duration of the settled alarm
6. (a) The alarm overlaps with another one
   Show warning: Ask the user to change the time or duration of the settled alarm

Expert User:

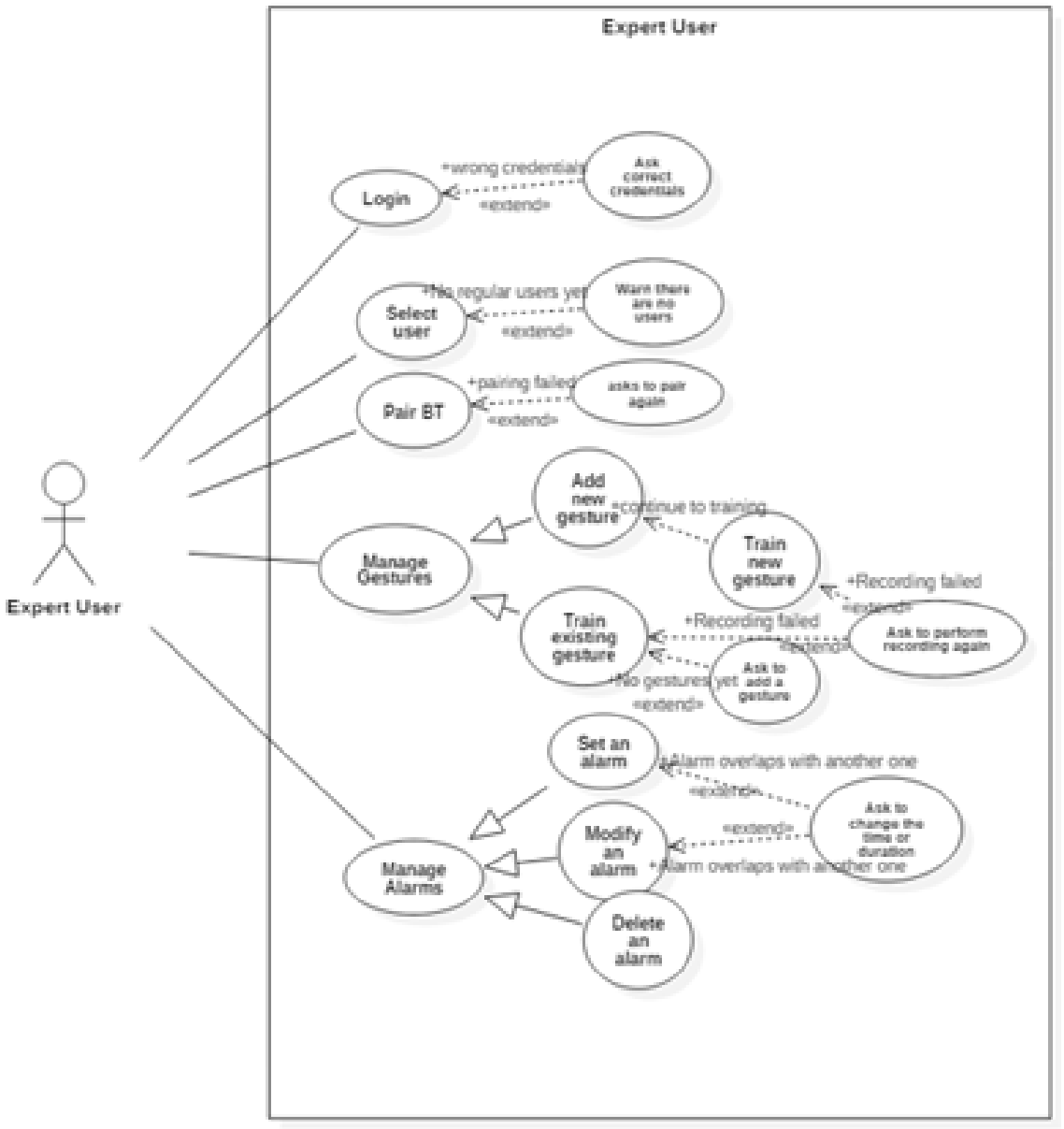

Figure 2.12: Use Case Diagram Expert User

1. Logs in
2. Select a user
3. Set an alarm
4. Modify an alarm
5. Delete an alarm

6. Add new gesture

7. Train new gesture

8. Train old gesture

Alternative scenario:

1. (a) Login fails
   Ask for username and password again

2. (a) There are no users
   show warning: There are no users, go back.

3. (a) The alarm overlaps with another one
   Show warning: Ask to change the time or duration of the settled alarm

4. (a) The alarm overlaps with another one
   Show warning: Ask to change the time or duration of the settled alarm

5. (a) The name of the gesture already exists for the user
   Show warning: Ask to change the name of the gesture or go back and train it.

6. (a) The recording failed
   Show warning: Ask the user to perform the recording again.

7. (a) There are no gestures to train yet
   Show warning: Ask the user to add some gestures first.

2.2.3.3 Activity Diagrams

Regular User:

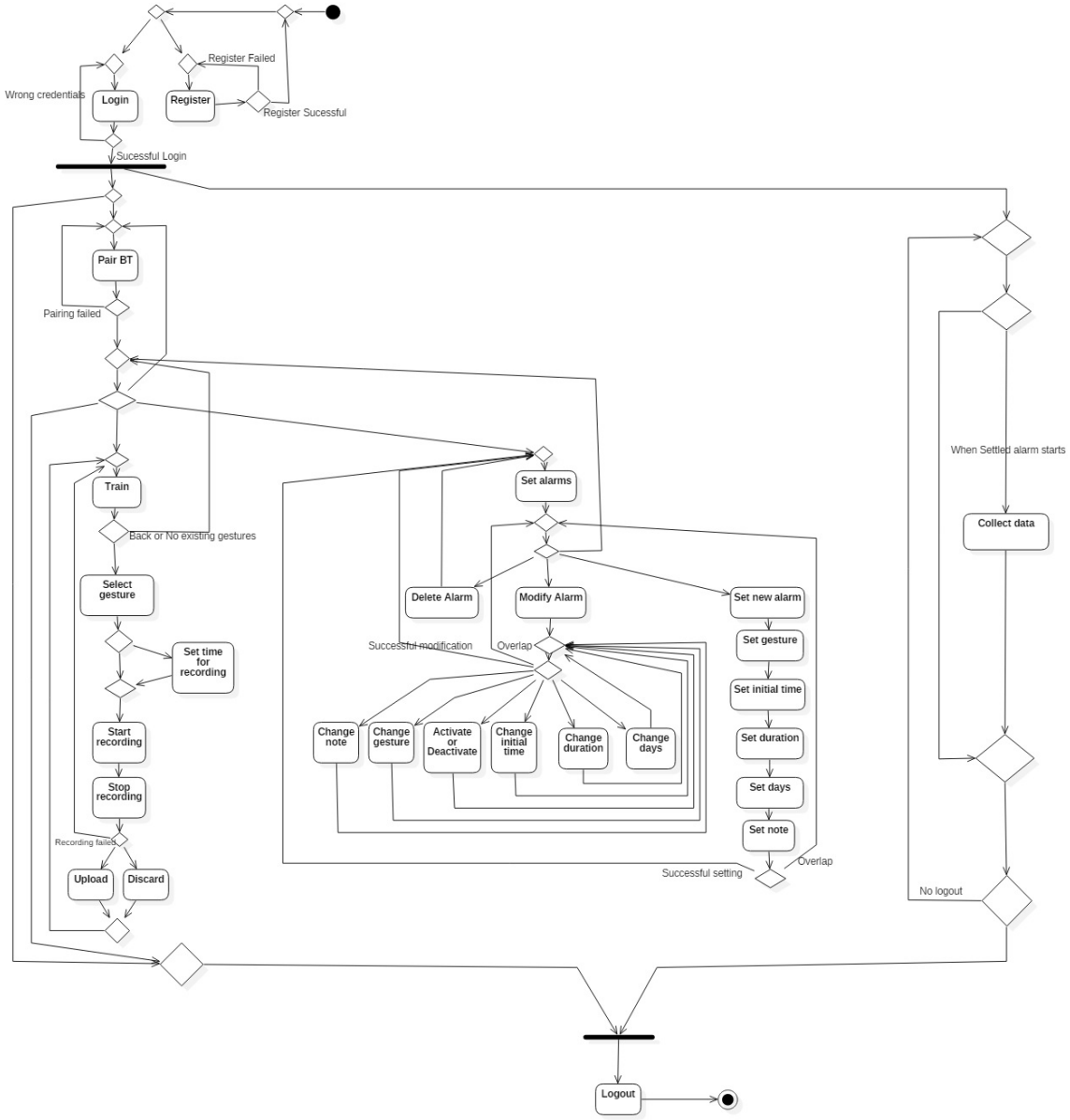

Figure 2.13: Activity Diagram Regular User

Expert User:

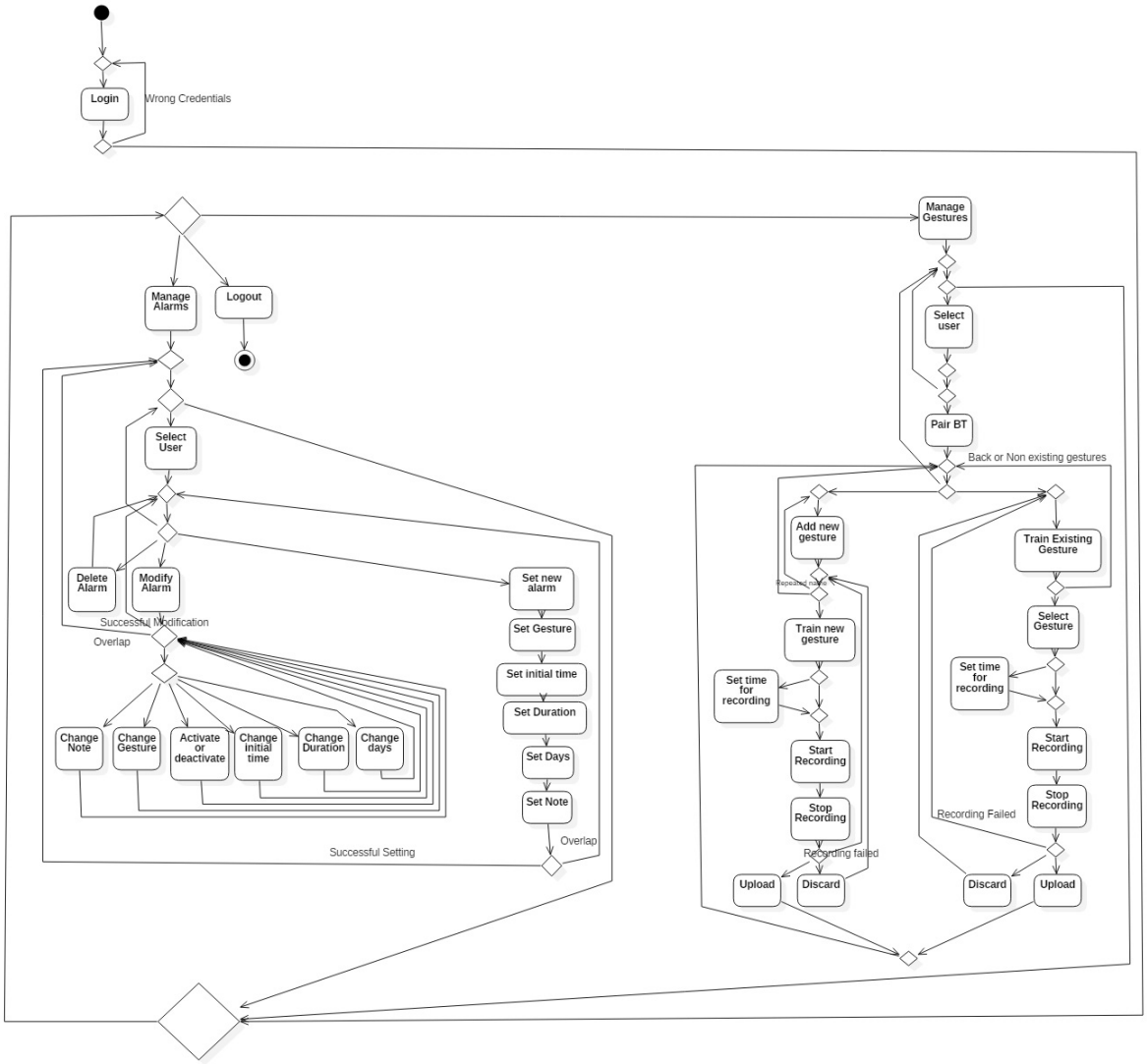

Figure 2.14: Activity Diagram Expert User

2.2.3.4 Class Diagram

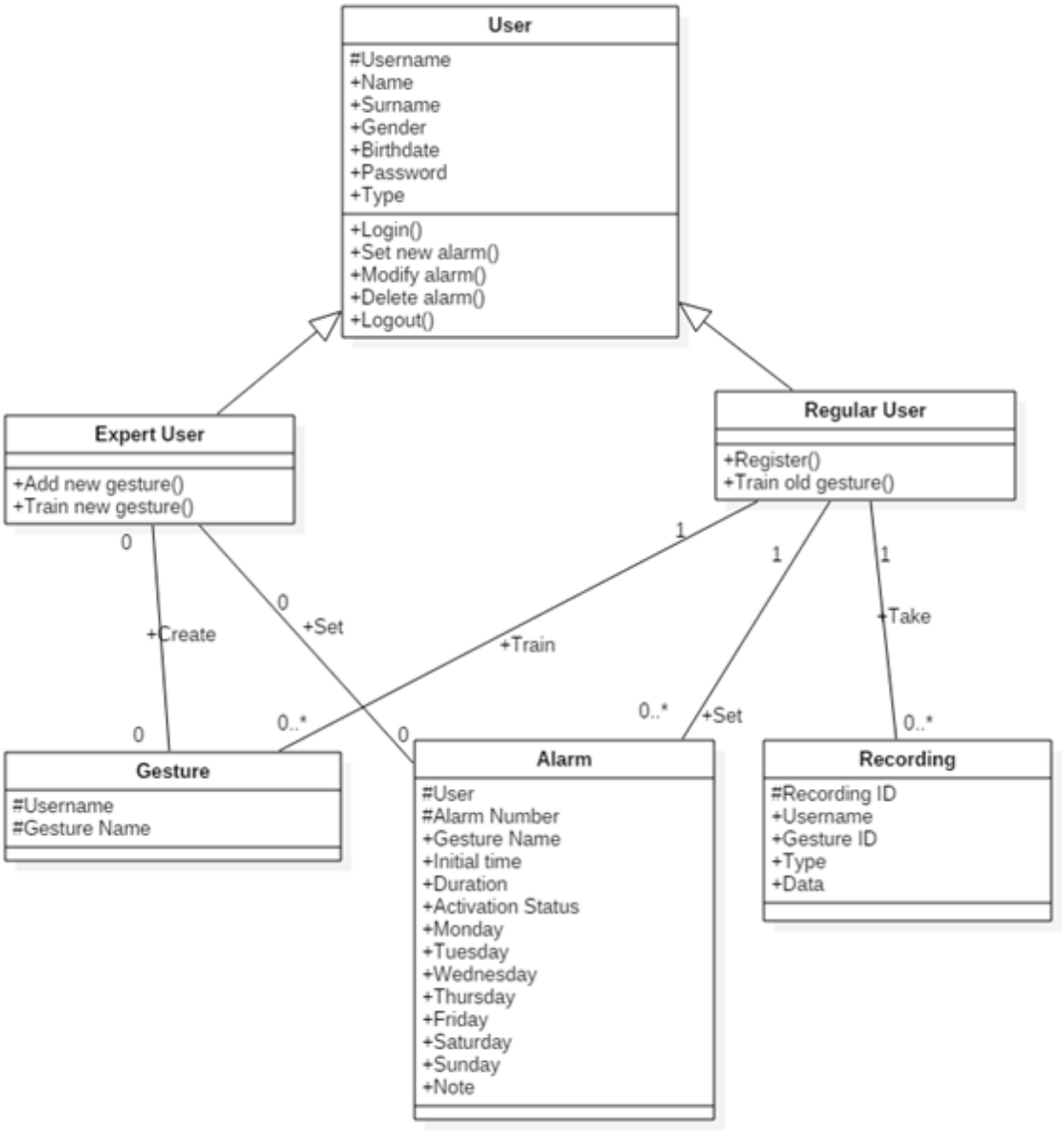

Figure 2.15: Class Diagram

2.2.3.5 Entity Relationship Diagram

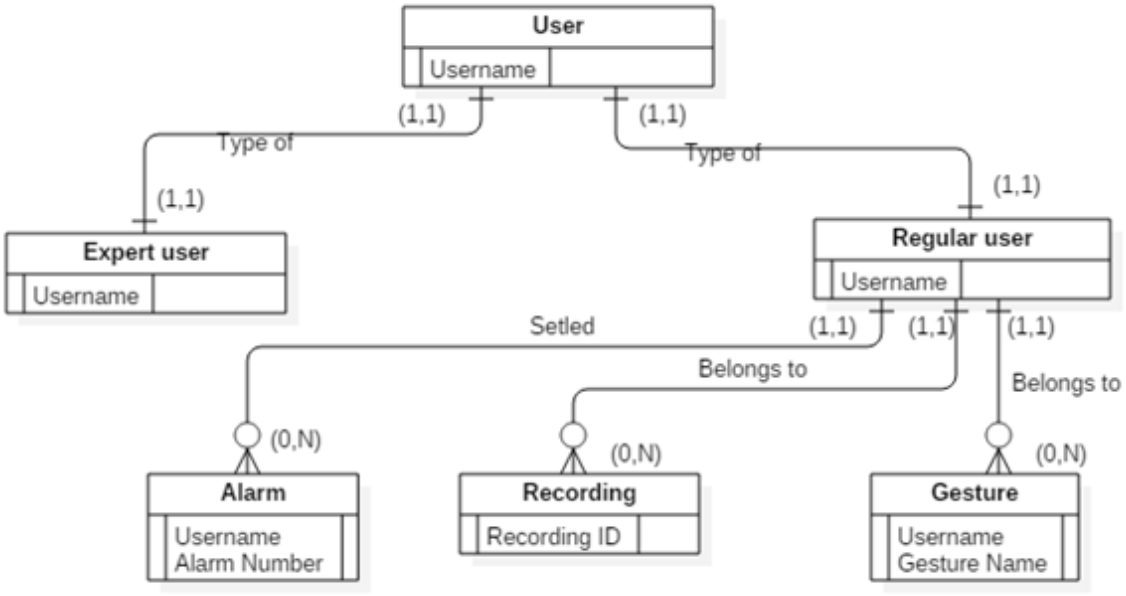

Figure 2.16: Entity Relationship Diagram

2.2.3.6 Technical Description of tables

The variables with (*) represent the key attribute that makes the data univocal.

Table 1: User

In the case of the regular user, this table fills up when the user registers on the platform. In the case of the expert user is previously stored in the system.

- (*) Username: String
- Password: String
- Name: String
- Surname: String
- Gender: Char
- Birthdate: Date

- Type: Bool

The username is the key attribute and is chosen by the user during the register, as well as the password. The gender is of type char and can only take three values male (M), female (F), other(O). The type of user can be either regular user (1) or expert user (0).

Queries for this table:

- Validation of credentials during the login: Given aa username and a password, validate if the user exists and if the password given coincides with the one stored in the database

From the Regular User platform:

- Enter the data: When the register is done
- Username validation: During the register, check if there exist another user with the same username. From the Expert User Platform:
- Retrieve a list with all the usernames: for the "Select User" environment

Table 2: Gestures
This table contains all the gestures each user has. A new gesture is inserted into the database when the user is training a new gesture. And it is only done if the register is done successfully and is uploaded.

- (*) Username: String
- (*) Gesture name(gesName): String
- Duration(dur): Int

The key attributes for this table are the username and the gesture name, meaning no user can have two gestures with the same name. It also has the duration, that indicates in minutes how long should a recording of the gesture last. The default value for the duration is 10 minutes.

Queries for this table:

From the Regular User platform:

- Retrieve all the gesture names from a user: Used for the "Set Alarm" environment and also for choosing a gesture in the "Training Old Gesture" environment.
- Retrieve the duration of a gesture given the gesture name and the user name.(used for the alarm setting)

From the Expert User platform:

- Enter the data: When adding a new gesture
- Name Validation: Given a username and a gesture name, validate that the user doesn't have another gesture named as so.
- Retrieve all the gesture names from a user: Used for the "Set Alarm" environment and also for choosing a gesture in the "Training Old Gesture" environment.

Table 3: Recording

- Username: String
- Gesture Name (gesName): String
- (*) Recording ID de (recordingID): String
- Data (recData): path to a file
- Type: Bool

The recording ID should be generated by the system concatenating the Gesture ID with a count of recordings that gesture has. The type of data refers to if it is testing or training data, for training data the value should be 1, and for testing data 0. The data stores a path to a .csv file stored in a folder in the server, which stores the data from the recording (6 axes of the accelerometer)

Queries for this table:

From the Regular User platform:

- Enter the data: During the training and when the alarms are working.
- Given a Username and a Gesture Name, retrieve if there's any training recording: Used during the alarm setting.

From the Regular User platform:

- Enter the data: During the training
- Given a Username and a Gesture Name, retrieve if there's any training recording: Used during the alarm setting.

Table 4: Alarms

- (*) Username: String
- (*) Alarm number (alarmNum): Int
- Gesture Name (gesName): String
- Activation status (actSt): Bool
- Initial Time (initTime): hour
- Monday (mon): Bool
- Tuesday (tu): Bool
- Wednesday (wed): Bool
- Thursday (th): Bool:
- Friday (fri): Bool
- Saturday (sat): Bool
- Sunday (sun): Bool

- Note: String

This table has a combined key composed of the username and the alarm number. The activation status indicated whether the alarm is activated (1) or not (0). The time indicates the time at which a recording should begin, and the duration of the recording is given depending on the gesture. Each day of the week is represented by a Boolean that indicates is the alarm should work on that day or that day of the week (1) or not (0). The note can be used or not, it is used in case the user wants to specify something, for example, which medication is supposed to take at that time:

We can constraint the maximum possible alarm number a user can have

Queries for this table:

- Enter the data (if it doesn't exceed the maximum alarms per user)
- Validate hour: Given all the attributes of the alarm, validate that no other alarm from that user will overlap with the alarm we are trying to insert.
- Delete an alarm given the alarm and the username.
- Modify any of the other attributes given the user and the number of the alarm (change the days activated, activation status, note or gesture)

App Environments Overview

Starting environment:

1. The user must identify itself to access to the account it has on the cloud.
2. The user inserts the password
3. Login button: If the username and password are valid the app continues opens the account. If it is not, it shows a warning asking for the correct credentials.
4. Register button: It goes to the registering environment

Figure 2.17: Starting environment

ENVIRONMENTS FOR THE REGULAR USER

Registering environment:

This environment can only register regular users. The expert users are inserted directly to the cloud database in the computer.

1. The user inserts its demographic data.
2. The system must validate the username does not already exist in the database.
3. The two password boxes should coincide for validation.
4. If some box is not filled or if the validation goes wrong, there should appear a warning specifying the problem.
5. If the register is successful it returns to the starting environment.

Figure 2.18: Register environment

Bluetooth paring (regular)

1. Select the Bluetooth with which we want to pair the phone with
2. Show whether the BT is connected or not
3. After having the BT connected the user can continue to the next environment.
4. With the Logout button the user returns to the Starting Environment.

Figure 2.19: Bluetooth paring environment (regular)

Main environment (Home)

1. Training button: Goes to the Training Existing Gesture Environment. if there exist gestures already associated to the user. Otherwise, it would show a warning telling the user there are no gestures yet and to please contact the expert.
2. Alarm set button: Goes to the Alarm Set Environment.
   This is only possible if there exist gestures in the database for the user that is

logged in. There will appear any warning regarding the impossibility of performing any action. For example, when they're not existing gestures to get to the alarm set.

3. With the Logout button, the user returns to the Starting Environment.

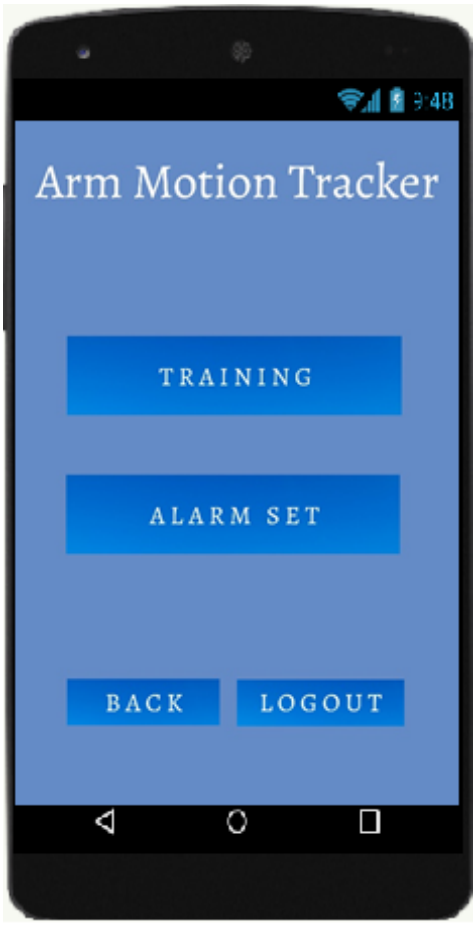

Figure 2.20: Home (regular user)

Train Existing Gesture Environment

While the user is in this activity all the programmed alarms will be deactivated, thus it will not record what it was programmed to, to prevent problems with the recording execution in the device and the transmission and storage of the data.

1. The user can select a gesture from all the existing gestures.

2. Start Recording Button: The phone sends a character to the device via BT to start the recording of the signal.

3. Stop recording button: The phone sends another character to the device to stop the recording. This can also be settled with the Set time button, inserting the time of the recording in minutes.

4. When the phone starts receiving information from the device it should appear a "Recording" sign, and when it stops a "Recording done" sign.

5. The recording can be either discarded or uploaded.

Figure 2.21: Train Existing Gesture

Alarm Set Environment

Show all the existing alarms for the user. When the tab is opened it shows the details of the alarm. Where the user can edit the gesture, the beginning time, the duration, and the repetition days.

The user can also add a new alarm with the + button. When an alarm is activated it

starts the recording of the accelerometric measures at the settled time, but it will give a reminder to the user 5 minutes before the recording begins.

If the user programs an alarm with a gesture that has no training yet, he/she is asked to perform some training.

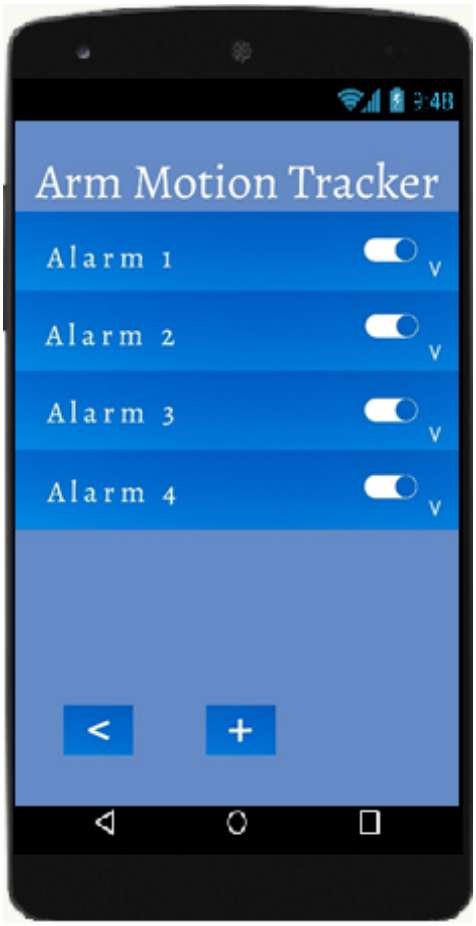

Figure 2.22: Alarm Setting

Details Pop Window

Edit Alarm Pop Windows

ENVIRONMENTS FOR EXPERT USER

Main Environment (Home)

In the mobile app, the expert user does not access the records performed by the users

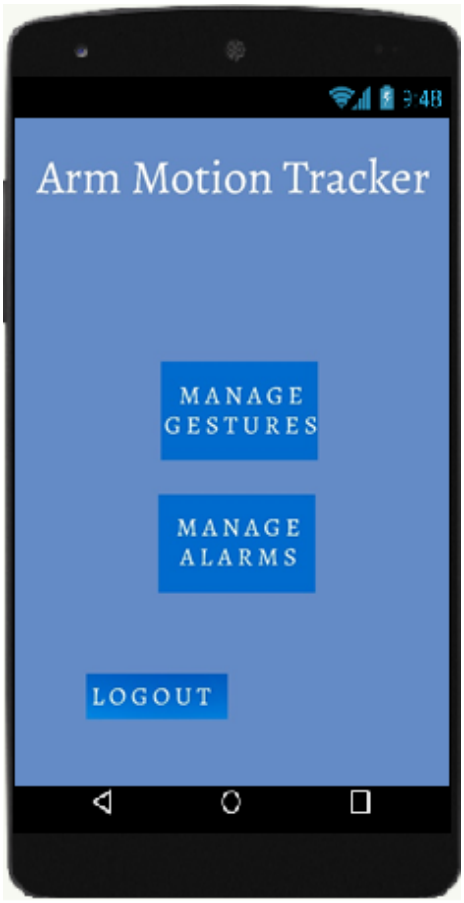

Figure 2.23: Home (expert)

or to their demographic data, but this can be performed directly in the server platform.

With the Manage Alarms button, the user goes to the Select User Environment that will later show the alarms from the selected user, but the alarms are not activated while being on an expert's profile.

With Manage gestures button it also goes to the Select user Environment but then it will go to Bluetooth Pairing (expert) environment.

With the logout button, the user returns to the Starting Environment.

Select User Environment

Here the administrator can choose any user from the database and then with the button "confirm" go to the Alarm Set Environment. Here everything would appear like it would to the selected user, when it presses "manage alarms" button was pressed previously, or to the Bluetooth Paring (expert) environment if the "manage gestures" button was pressed previously.

Figure 2.24: Select user

Bluetooth Pairing (expert)

1. Select the Bluetooth with which we want to pair the phone with
2. Show whether the BT is connected or not
3. After having the BT connected the user can continue to the next environment.
4. With the Logout button the user returns to the Starting Environment.

Figure 2.25: Bluetooth paring environment (expert)

Manage Gestures Environment

Figure 2.26: Manage gestures

1. Train new gesture button: Go to the Add Gesture environment
2. Train old gesture button: Go to Train Existing Gesture environment if there exist gestures already associated to the user. Otherwise, it would show a warning telling the user there are no gestures yet.

Add Gesture Environment

The user inserts the name of the new gesture. Then the user presses the button Check and systems check if the gesture already exists in the database; in that case, it shows a warning and the user can't perform the other actions in the environment but changing the name or going back. Otherwise, the user can press the button "insert". Once the gesture is inserted in the DB the user can either go back to the previous environment or continue to the Train New Gesture environment.

Figure 2.27: Add gesture

Train New Gesture Environment

Start Recording Button: The phone sends a character to the device via BT to start the recording of the signal. Stop recording button: The phone sends another character to the device to stop the recording. This can also be set with the Set time button, inserting the time of the recording in minutes.

When the phone starts receiving information from the device it should appear a "Recording" sign, and when it stops a "Recording done" sign.

The recording can be either discarded or uploaded.

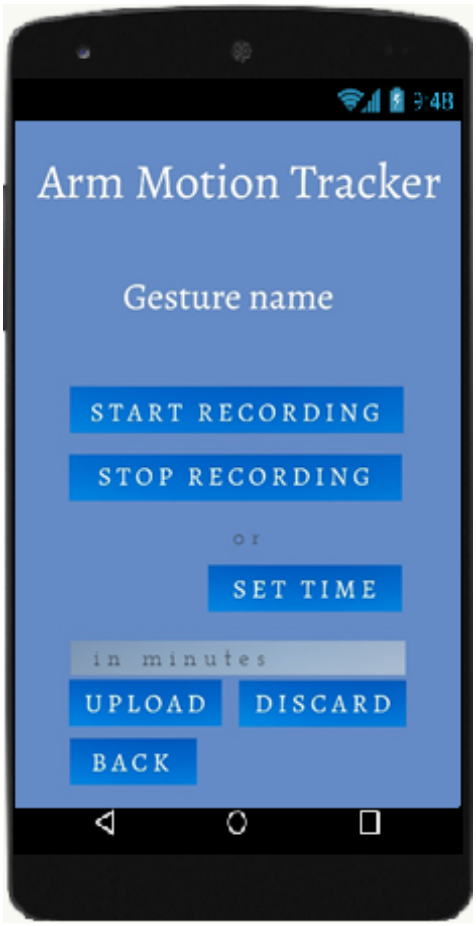

Figure 2.28: Train new gesture

2.2.4 Software Implementation

2.2.4.1 Database

As a hosting for the database, the www.nwksec.com domain was used with a MySql database, which is built over a Linux machine belonging to the company that provides the domain above. There the database with name "Tesis" was created.

Some considerations that were made when creating the database in MySQL given the data-types existing in this platform. Instead of using boolean, tinyint was used. Instead of using string, varchar was used to optimize storage space. For all the tables except for the table "Alarms", all the fields have the non-null characteristic to avoid missing information. "Alarms" table will only allow null data in the field "notes" in case there does not exist any.

All the queries were developed as stored routines, to avoid writing individual statements from the app and repeating queries that are needed during different functions.

The source code of the database can be found in Section 4.2.

2.2.4.2 Signal Pre-processing

Since the signal is sent from the device with a special format, determined by the implemented firmware, a MATLAB script was used to extract the raw data. This script with the proper documentation is included in Section 4.3.

2.2.5 Testing

This subsection illustrates a proposal of a method in which the device could be tested to determine its accuracy and to perform a calibration.

2.2.5.1 Smartphone Application

For the complexity of the designed application and the scope of this project. The full application was not implemented. However, for testing, a much simpler application was developed using MIT App Inventor Designer. This tool is an online development environment for applications in Android and iOS, which uses a visual language that allows the user to program using predetermined blocks.

The application used for testing the device has only one environment, five buttons, and a list picker.
The list picker is used to choose the Bluetooth device with which the smartphone will pair i.e. the wearable designed in this project. While the device is not connected it will show in the connection status "NOT CONNECTED" in red, and while is connected a "CONNECTED" in green.
The button "Start Acquisition" pulls up a flag, if the Bluetooth is connected. While this flag is up the phone sends a character to the device that indicates that the acquisition should start until it receives an acknowledgment character from the wearable.
The button "Stop Acquisition" pulls up a flag, if there's an ongoing acquisition. While this flag is up the phone sends a character to the device that indicates that the acquisition should stop until it receives an acknowledgment character from the wearable.
The button "Save Acquisition" saves in a file the data taken from the acquisition, if available.
The button "Discard Acquisition" discards the data taken from the acquisition.

The button "Exit" exits the application.

Figure 2.29: Application Layout

2.2.5.2 IMU Calibration and Test

For the IMU sensor, two calibration procedures are needed, one for the accelerometer and another one for the gyroscope.

To calibrate the accelerometer, we must do it on a leveled surface and it can be done through static measurements, when a tri-axis accelerometer is stationary, its total measured acceleration is 1g (9.8m/s^2), regardless of orientation. This total acceleration can be calculated from the X, Y, and Z outputs of the accelerometer as:

$$a_{total} = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{2.3}$$

So for the calibration the wearable must be placed in such way that the accelerometer stays steady in every one of the six orientations possible(x facing up, x facing down, y facing up, y facing down, z facing up and z facing down) in which one of the axes should have an acceleration of ±1g while the other axis should have an acceleration of 0g. If any of the axes has a variation from this measurement it should be compensated as an off-set bias error.

To calibrate the gyroscope, we need to have some controlled movement. The physical setting of the sensor should be an arm with a fixed pivot and two mechanical stops for two angular positions at known angles. It could be 90° or 180° to observe a wider range of the sensor's behavior. To have a more accurate measurement an inclinometer can be fixed to the rotating arm as the golden measure. The first step would be orienting the arm in one of the fixed angular position for a known amount of time, for example, 5 seconds. This measurement will be useful because the output should be 0, if it is not 0 we already have the offset and a bias-offset correction can be performed by averaging the output signal during those 5 seconds. The compensation for this error can be done by simple subtraction. The next step would be letting the arm fall to the second fixed angular position, using an inclinometer to measure angular displacement in a time-lapse, this will give a measurement to compare with the gyroscope's measurement. Since the gyroscope's output is angular displacement and the inclinometer's output is the angle, the gyroscope's output should be integrated (making sure in the case of 180° the integration of negative parts of the signal is not done together with the positive parts of the signal). Having the integrated value, the scale-factor error can be estimated as:

$$\frac{expectedvalueorgoldenmeasure}{measuredvalue} \tag{2.4}$$

This can be compensated by multiplying the measurement by the scale-factor error. This can be repeated several times to have more accuracy, averaging the scale-factor errors estimated in each result. This should be repeated placing the sensor in such a way that every axis is tested.

Also, when the gyroscope is steady, the output in every axis should be 0. If any of the axes has a variation from this measurement it should be compensated as an off-set bias error.

The acquisition of the signals for the calibration should be done using the application described in subsection 2.2.5.1. After performing this we can estimate the effective bits

of the sensor with the following formula:

$$ENOB = \frac{SINAD - 1.76}{6.02} \tag{2.5}$$

Where ENOB is the effective number of bits and SINAD can be expressed as:

$$SINAD = \frac{P_{signal} + P_{noise} + P_{distortion}}{P_{noise} + P_{distortion}} \tag{2.6}$$

(LeCroy, 2011) With the proposed testing protocol for the accelerometer, there would not be any information for the distortion, so it would not be considered in the equation. The power of the noise for each axis would be considered as the power of the off-set error, and the power of the signal would be considered as the power of the signal at the full scale.

With the proposed testing protocol for the gyroscope, the power of the noise for each axis would be considered as the power of the off-set error, the power for the distortion of each axis would be considered as the power of the scale-error factor, and the power of the signal would be considered as the power of the signal at the full scale. After performing this protocol the measurements with the calibration may also be compared to those of wearables already existing on the market.

2.2.5.3 Charge Duration Measure

To know the charge duration of the device we divide the time consumption into two different moments, when the device is performing an acquisition and when it is in sleep mode. In each f this moments the supply current needs to be measured. Knowing that the electric charge of the LiPo battery that supplies the circuit is 250mAh the expression to estimate the duration in each of the modes is given by the equation 2.7, that shows the discharge time of the battery as is the circuit was only working in one of the modes. The real duration of the battery will depend in how much time the device is in acquisition mode and sleep mode.

$$T_{discharge} = \frac{250mAh}{I_{supply}} \tag{2.7}$$

In this project we were able to measure the supply current when all the components are in operation mode i.e. when the energy consumption is higher, where the supply current was of 21mA, thus the duration of the battery without a sleep mode would be of 11 hours and 54 minutes. In sleeping mode is expected to be much higher.

3. Conclusions and Future Work

This thesis work focused on the design of the integration of a wrist wearable device for gesture recognition in the context of adherence measurement, based in already existing approaches that help to overcome the challenges that poor adherence present to the healthcare system.

Even though in the market there already exist many electronic devices that assess the patients' adherence to behaviors such as medication intake or help as a reminder, this project tried to design a system that went one step forward and integrate these two features in one single system. The system was designed to be a wrist band with a 6-axis inertial measurement unit to measure the movement of the forearm as daily activities were performed. Many wearables featuring accelerometers and gyroscopes tend to consume a lot of battery because they are constantly sending continuous signals. This characteristic is not very desirable for the monitoring adherence application, since we are only interested in the activities the patient performs at determined times of the day, and we would like the battery to have a long duration. For this reason, the designed device, tried to save energy by remaining in sleep mode unless an alarm indicated an acquisition needed to be done. The alarms are supposed set by a physician according to the recommendations for a specific patient, and the recordings of the arm movement during the activities were designed to be stored in a cloud database. This aims to avoid miss-reporting of the performed activities due to forgetfulness that may happen in a diary, and also to help the patient to remember when to perform the recommended activities

However the design is very limited to controlled environments for now, because this wearable device has only one node to perform gesture recognition. This limits the applicability because it is impossible to know the position of the arm at the beginning of the recording outside of a controlled trial, having no other node serving as reference in the body, and the arm has seven degrees of freedom. A possible way to improve this would be adding a second node that can go inside a pocket, for having a spatial reference. Another thing that could be improved in the design is the physical layout

of the device, since it is a little bit hard to manipulate for a user.

Unfortunately, the hardware implementation of the device is not yet tested, so a future work could be testing the device with the method suggested in the subsections 2.2.5.2 and 2.2.5.1 to determine the accuracy of the device and then comparing it with other arm motion tracking devices in the market as well as the power consumption. The next step of the testing would be that of trying an experimental protocol where different subjects performed daily life hand gestures and see how already trained recognition algorithms perform with data collected with this device compared with data collected from other devices.

In future approximations to this work the software designed could be implemented and test the usability and evaluate if the interconnection choices were the best for this application.

4. Appendices

4.1 Appendix A: Microcontroller Source Code

In this appendix, you will find the source code that was implemented to program the systems state machine. It includes the main and all the libraries that were used for this purpose.

Some of the libraries used for this project were developed from the manufacturer and modified to suit better this application. Those cases include the Copyright licenses fitting the conditions they proposed.

The code is organized in the following way:

- Main: Source file
- USART: Header file and source file
- RN4871: Header file and source file
- ICM20600: Header file and source file
- I^2C Protocol
  - I^2C Master: Header file and source file
  - I^2C Protocol:Header file and source file

Main:

Source File:

```
/*
 * File:   main.c
 * Author: Diana Maritza González Ramírez
 * This project is made for a ATTiny1617 microcontroller, a RN4871 BLE module
 * and an ICM20600 6-axis IMU sensor. For further understanding of the code,
 * the programming choices and the documentation, check out the datasheets.
 */
#define F_CPU 5000000

#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include "rn4871.h"
#include "icm20600.h"
#include "usart.h"
#include "i2c.h"
#include "i2c_master.h"


/*****************************************************************************
                           Global Variable declaration
 *****************************************************************************/
/*Since we will be using the 20MHz Oscillator and prescaling it with 4 the CPU
 * clock will be of 5MHz and we define this valua because it will be usefull in
 * some functions*/

int state=0; //state in the machine
int acqRes=0; //Flag: when in 1 indicates the device was reseted during an acquisition
int acqPair=0;//Flag: when in 1 indicates paring was lost during an acquisition
char rx='0';
bool connectionStatus= false;
icm20600 thesensor;//This variable will contain the sensor information
/*The following variables will serve as pointers yo send the data*/
int16_t accX;
int16_t accY;
int16_t accZ;
int16_t gyrX;
int16_t gyrY;
int16_t gyrZ;
int16_t off_accX;
int16_t off_accY;
int16_t off_accZ;
int16_t off_gyrX;
int16_t off_gyrY;
int16_t off_gyrZ;
int blinkCnt=0;
/*****************************************************************************
                           Interrupt Service Routines
 *****************************************************************************/
ISR(PORTA_PORT_vect)
{
    //Port A: pin 5
    // Ensures the interrupt was triggerd by pin5 (falling edge), meaning reset button was pressed
    if(PORTA.INTFLAGS & PIN5_bm)
    {
     if(PORTA.IN & PIN5_bm)
        {

            if(state==3){//it was reseted during the acquisition
            acqRes=1;  //pulls-up the flag
            }
        state=0;//re-initializes the system
```

```
        }
    }
    PORTA.INTFLAGS = PIN5_bm; // clear interrupt flag

}

ISR(TCA0_OVF_vect)
{

    switch(state)
    {
        case 2:
            //makes the blinking be 100 times slower than the ovf condition (1Hz)
            if(blinkCnt==100)
            {
             //Toogle the val of the LED to make it blink
            PORTC.OUTTGL = PIN1_bm;
            blinkCnt=0;
            }else{
                blinkCnt++;
            }
            if(rn4871_conStat()==false) //if paing is lost
            {
                state=1;//return to the state waiting for pairing
            }

        break;
        case 3:

            if(rn4871_conStat()==false) //if paing is lost
            {
                acqPair=1;//pull acqPair flag up
                state=1;//return to the state waiting for pairing
            }else{ // if the device is still paired
                if (i2c_master_is_enabled() == 0) {
                                    i2c_master_init();
                            }
                /*****************Acquire the IMU signals*******************/
                icm_get_acc(&accX, &accY, &accZ, thesensor);
                icm_get_gyro(&gyrX, &gyrY, &gyrZ, thesensor);
                /****************Send the accelerometer data*****************/
                printf("aX");//Beginning of X acc read
                printf("%x\r\n",accX);//X acc read
                printf(".");//Separates by .

                printf("aY");//Beginning of Y acc read
                printf("%x\r\n",accY);//Y acc read
                printf(".");//Separates by .

                printf("aZ");//Beginning of Z acc read
                printf("%x\r\n",accZ);//Z acc read
                printf(".");//Separates by .

                /*******************Send gyroscope data**********************/
                printf("gX");//Beginning of X gyr read
                printf("%x\r\n",gyrX);//X gyr read
                printf(".");//Separates by .

                printf("gY");//Beginning of Y gyr read
                printf("%x\r\n",gyrY);//Y gyr read
                printf(".");//Separates by .

                printf("aZ");//Beginning of Z acc read
                printf("%x\r\n",gyrZ);//Z gyr read
                printf(".");//Separates by .
            }
        break;
    }
    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;//The interrupt flag has to be cleared manually
```

```
}

/*********************************************************************************
                              Function Declarations
 *********************************************************************************/
void clock_config()
{
    //CLK_MAIN=20MHz and CLK_PER=5MHz
    //Datashhet section 10.5.1
    CLKCTRL.MCLKCTRLA=CLKCTRL_CLKSEL_OSC20M_gc/* 20MHz Internal Oscillator (OSC20M) */
            | 0 << CLKCTRL_CLKOUT_bp /* System clock out: disabled */;
    //Datasheet section 10.5.2: prescaler divition in 4, prescaler enabled
    CLKCTRL.MCLKCTRLB=CLKCTRL_PDIV_4X_gc /* 4 */
                        | 1 << CLKCTRL_PEN_bp /* Prescaler enable: enabled */;

}
void TCA0_init()
{
    TCA0.SINGLE.INTCTRL |= (TCA_SINGLE_OVF_bm); // Enable timer interrupts on overflow on timer A
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc;//No waveform generation
    TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);//Disable event counting
    /*Knowing that the overflow time is given as:
     * TIME_TCA_IQR(s)=(((TCA.PER+1)*TCAprescaler)/CLK_PER)
     * (Microchip TB3217 application note)
     * Having CLK_PER=5MHz, and a desired frecuency of 100Hz, using a prescaler
     * of 1 the count would be 49999 that is reachable by a 16 bits counter, so
     * is chosen to be single and PER set at 0xC34F
     * (The maximum value TCA.PER can take is (2^16)-1 i.e. 65535 or 0xFFFF)
     */
    TCA0.SINGLE.PER=0xC34F;//16bit overflow
    /*the prescaler divides from CLK_PER*/
    TCA0_SINGLE_CTRLA |= (TCA_SINGLE_CLKSEL_DIV1_gc)/*Prescaler with division x*/
            | (TCA_SINGLE_ENABLE_bm);/*Enable the peripheral*/

}
//RTC(real time counter) initialization

void set_ports()
{
    //Unused ports are set as outputs
    /**********************Port initialization******************************/
    //Set pin direction
    //PORTA.DIR[0]=;//UPDI
    PORTA.DIRSET=PIN1_bm;//BLE_TX
    PORTA.DIRCLR=PIN2_bm;//BLE_RX
    PORTA.DIRCLR=PIN5_bm;//Reset button
    PORTA.DIRCLR=PIN6_bm;//IMU_INT1
    /*PB0:input:SCL
     *PB1:input:SDA
     *PB4:output: BLE_RXIND
     *PB5:output:BLE_reset
     *PB6:input:IMU_INT2*/
    //0b00111101=0x3D
    PORTB.DIRSET=0x3D;
    //PORTC.DIR[0]=;//Bootloader
    PORTC.DIRSET=PIN1_bm;//LED
    // set levels fot the outputs
    PORTA.OUTSET=PIN1_bm;//BLE_TX:HIGH
    PORTB.OUTSET=PIN0_bm;//SCL:HIGH
    PORTB.OUTSET=PIN1_bm;//SDA:HIGH
    PORTB.OUTCLR=PIN4_bm;//BLE_RXIND:LOW
    PORTB.OUTSET=PIN5_bm;//BLE_RESET:HIGH
    PORTC.OUTCLR=PIN1_bm;//LED:OFF
    //set pull mode, invertion and input/sense configuration
    //DATASHEET SECTION 16.5.11
    PORTA.PIN2CTRL=PORT_PULLUPEN_bm;//PULL-UP MODE ENABLED
    PORTA.PIN5CTRL=PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc; // enable pull-up and interrupt on both
edges
    PORTA.PIN6CTRL=PORT_PULLUPEN_bm;//PULL-UP MODE ENABLED
```

```
    PORTB.PIN0CTRL=~PORT_PULLUPEN_bm | ~PORT_INVEN_bm |PORT_ISC_INPUT_DISABLE_gc;
    //PULL-UP MODE DISABLED,I/O ON PIN NOT INVERTED,ISC Interrupt disabled but input buffer enabled
    PORTB.PIN1CTRL=~PORT_PULLUPEN_bm | ~PORT_INVEN_bm |PORT_ISC_INPUT_DISABLE_gc;
    //PULL-UP MODE DISABLED,I/O ON PIN NOT INVERTED,ISC Interrupt disabled but input buffer enabled
    PORTB.PIN6CTRL=~PORT_PULLUPEN_bm;//PULL-UP MODE DISABLED
    PORTMUX.CTRLB |= PORTMUX_USART0_bm; //set alternate pin mux for USART

}
void system_init()
{
    i2c_init();
    if (i2c_master_is_enabled() == 0) {
                                        i2c_master_init();
                                    }
    icm_init(thesensor);
    usart_init();
    //rn4871_init();

}
/*converts uint16_t (2 bytes) into to ASCII code numbers stored in a char array,
 * MSB first since the higher bits are stored in the first cell and the lower
 * bits in the second cell
 */
/*const char * ui16_2_ascii(uint16_t num)
{
    char ascii[2];
    ascii[2]=num&0xFF;//gets the 8 lower bits
    ascii[1]=num>>8;//gets the 8 higher bits
    return ascii;
}*/

/*****************************************************************************
                                State Machine
 *****************************************************************************/
int main(void)
{
    clock_config();
    TCA0_init();
    set_ports();
    /*Gobally enable the interrupts*/
    sei();

    while(1)
    {
        switch(state)
        {
            case 0:/**************SYSTEM RE-INIZIALIZATION******************/


              /*indicates the user the system is initializing*/
              PORTC.OUTCLR=PIN1_bm;//Turn off the lED
              _delay_ms(1000);//Wait 1 seconds
              PORTC.OUTSET=PIN1_bm;//Turn on the LED
              _delay_ms(2000);//Wait 2 seconds
              PORTC.OUTCLR=PIN1_bm;//Turn off the lED
              system_init();
              state=1;//pass directly to state 1
            break;
            case 1:/******************WAIT FOR PAIRING**********************/


                PORTC.OUTCLR=PIN1_bm;//Turn off the lED
                //Sensor - sleepmode
                /*If the i2c_master is enabled it initializes it*/
                    if (i2c_master_is_enabled() == 0) {
                                        i2c_master_init();
                                    }
                icm_set_pow_mode(ICM_SLEEP_MODE, thesensor);
                //MCU - low power operation
```

```
    /**************state transitions*****************
     * to 0 if reset button is pressed (ISR:PORTA)
     * to 2 if the BT got paired with another device*/
    connectionStatus=rn4871_conStat();
    if(connectionStatus==true)
    {
        state=2;
    }

break;
case 2:/***************WAIT FOR ACQUISITON SIGNAL*****************/
    //Sensor - sleepmode
    /*If the i2c_master is enabled it initializes it*/
        if (i2c_master_is_enabled() == 0) {
                            i2c_master_init();
                    }
    icm_set_pow_mode(ICM_SLEEP_MODE, thesensor);
    //MCU - low power operation

    /*************Evaluate if the flags are up*******************/
    /*If acqRes was held in 1 sends a message by the BT that
     * indictes it*/
    if(acqRes==1)
    {
     printf("ERROR:RESET_WHILE_ACQUISITON_RUNNING!");
     acqRes=0;//pull acqRes flag down
    }
    /*if acqPair was held in 1 sends a message by the BT that
     *  indictes it*/
    if(acqPair==1)
    {
     printf("ERROR:MISSED_CONNECTION_WHILE_ACQUISITON_RUNNING!");
     acqPair=0;//pull acqPair flag down
    }
    rn4871_dataMode();
    rx= usart_get();//read BT Rx
    /***************state transitions*****************
     * to 0 if reset button is pressed (ISR:PORTA)
     * to 1 if pairing is lost (ISR: TCA0)
     * to 3 if Rx=A(acquisition)*/
    if(rx=='A')
    {
        rx='0';
        printf("K");//Gives an acknowledge to the Smartphone
        printf("START_ACQ");//Marks in the file the beginning of an acquisition
        state=3;//changes to state 3
         /*Calibration*/
           //gets first values
        icm_get_acc(&accX, &accY, &accZ, thesensor);
        icm_get_gyro(&gyrX, &gyrY, &gyrZ, thesensor);
        //gets offset
        icm_get_acc_off(&off_accX, &off_accY, &off_accZ, thesensor);
        icm_get_gyro_off(&off_gyrX, &off_gyrY, &off_gyrZ, thesensor);
        /***************Send the accelerometer data******************/

        printf("acalX");//Beginning of X acc calibration read
        printf("%x\r\n",accX);//X acc calibration read
        printf(".");//Separates by .

        printf("aoffX");//Beginning of X acc read
        printf("%x\r\n",off_accX);//X acc offsetread
        printf(".");//Separates by .


        printf("acalY");//Beginning of Y acc calibration read
        printf("%x\r\n",accY);//Y acc calibration read
        printf(".");//Separates by .

        printf("aoffY");//Beginning of Y acc read
```

```
                    printf("%x\r\n",off_accY);//Y acc offsetread
                    printf(".");//Separates by .

                    printf("acalZ");//Beginning of Z acc calibration read
                    printf("%x\r\n",accZ);//Z acc calibration read
                    printf(".");//Separates by .

                    printf("aoffZ");//Beginning of Z acc read
                    printf("%x\r\n",off_accZ);//Z acc offsetread
                    printf(".");//Separates by .

                    /*******************Send gyroscope data**********************/
                    printf("gcalX");//Beginning of X gyr calibration read
                    printf("%x\r\n",gyrX);//X gyr calibration read
                    printf(".");//Separates by .

                    printf("goffX");//Beginning of X gyr read
                    printf("%x\r\n",off_gyrX);//X gyr offsetread
                    printf(".");//Separates by .


                    printf("gcalY");//Beginning of Y gyr calibration read
                    printf("%x\r\n",gyrY);//Y gyr calibration read
                    printf(".");//Separates by .

                    printf("goffY");//Beginning of Y gyr read
                    printf("%x\r\n",off_gyrY);//Y gyr offsetread
                    printf(".");//Separates by .

                    printf("gcalZ");//Beginning of Z gyr calibration read
                    printf("%x\r\n",gyrZ);//Z gyr calibration read
                    printf(".");//Separates by .

                    printf("goffZ");//Beginning of Z gyr read
                    printf("%x\r\n",off_accZ);//Z gyr offsetread
                    printf(".");//Separates by .

                }
            break;
            case 3:/*****************PERFORM ACQUISITION**********************/
                PORTC.OUTSET=PIN1_bm;//Turn on the LED
                /*If the i2c_master is enabled it initializes it*/
                    if (i2c_master_is_enabled() == 0) {
                                    i2c_master_init();
                                }
                //read BT Rx
                // Sensor-6 axis low noise operation
                icm_set_pow_mode(ICM_6AXIS_LOW_NOISE, thesensor);
                //MCU - normal power operation
                /*Acquires signal: Performed at the ISR:TCA0,
                 *  because it must have a sampling frequency*/
                /***********************state transitions************************
                 * to 0 if reset button is pressed: pull acqRes flag up (ISR:PORTA)
                 * to 1 if pairing is lost: pull acqPair flag up (ISR:TCA0)
                 * to 2 if the acquisition should already stop Rx=S(stop)*/
                rx= usart_get();//reads the BT
                if(rx=='S')//if it receives the stop character
                {
                    printf("K");//Gives an acknowledge to the Smartphone
                    printf("STOP_ACQ");//Marks in the file the end of an acquisition
                    state=2;//changes to state 2
                    rx='0';
                }
            break;
        }

    }
    return 0;
}
```

USART

Header File:

```
/*
 * File:   usart.h
 * Author: DianaM
 *
 */

#ifndef USART_H
#define USART_H

#ifdef  __cplusplus
extern "C" {
#endif
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
// baud rate calculation according to Microchips's Application Note TB3216 "Getting Started with
USART"
#define USART0_BAUD_RATE(BAUD_RATE) ((float)(5000000.0 * 64 / (16 * (float)BAUD_RATE)) + 0.5)

void usart_init(void);
void usart_putchar(uint8_t c);
void usart_putstr(uint8_t *str);
char usart_get(void);
const char * usart_getstr(int maxtam);
#ifdef  __cplusplus
}
#endif

#endif  /* USART_H */
```

Source File:

```
#include "usart.h"
#include <stdbool.h>


void usart_init(void)
{
    USART0.BAUD = (uint16_t)USART0_BAUD_RATE(115200);//Set baud rate
    USART0.CTRLB = 0 << USART_MPCM_bp       /* Multi-processor Communication Mode: disabled */
                     | 0 << USART_ODME_bp     /* Open Drain Mode Enable: disabled */
                     | 1 << USART_RXEN_bp     /* Receiver enable: enabled */
                     | USART_RXMODE_NORMAL_gc /* Normal mode */
                     | 1 << USART_SFDEN_bp    /* Start Frame Detection Enable: enabled */
                     | 1 << USART_TXEN_bp;
    return 0;
}
/*
 * Puts a character c available in the UART TXDATA
 */
void usart_putchar(uint8_t c)
{
        while (!(USART0.STATUS & USART_DREIF_bm)) {
        }
        USART0.TXDATAL = c;
}
/*
 * Puts a string str available in the UART TXDATA
 */
void usart_putstr(uint8_t *str)
{  while(*str)
    {
    usart_putchar(*str);
    ++str;
```

```
    }
}
/*
 * receive a character (polling)
 */
char usart_get(void){
        while ( !(USART0.STATUS & USART_RXCIF_bm) );
        return USART0.RXDATAL;
}
/*
 * recieve a string of maximum length of maxtam returns the string
 */
const char * usart_getstr(int maxtam)
{
    char str[maxtam+1];//+1 to set the '\0' at the end
    bool flag=true;
    int i=0;
    while(i <maxtam){
        str[i]=usart_get();
        if(str[i]=='\0')
        {
            break;
            flag=false;
        }
    }
    if(flag==true)
    {
        str[maxtam+1]='\0';
    }
    return str;
}
```

RN4871

Header File:

```
/*
 * File:   rn4871.h
 * Author: Diana M
 *
 *This file contains all the function declarations for what regards to the BLE module
 * RN4871
 */

#ifndef RN4871_H
#define RN4871_H


void rn4871_init(void);
void rn4871_waitfor(char endchar);
void rn4871_setFuses(void);
void rn4871_commandMode(void);
void rn4871_dataMode(void);
bool rn4871_conStat(void);
void rn4871_sleep(void);
void rn4871_wakeup(void);


#endif  /* RN4871_H */
```

Source File:

```
#include "rn4871.h"
#include "usart.h"
#include "i2c_master.h"
#include "icm20600.h"
#include <string.h>
```

```
#include <stdlib.h>
#include <stddef.h>
#include <util/delay.h>

//Transparent UART Service is set to 49535343-FE7D-4AE5-8FA9-9FAFD205E455
//Transparent UART TX 49535343-1E4D-4BD9-BA61-23C647249616

void rn4871_init(){
        usart_init();
        PORTB.OUTCLR = PIN4_bm; // BLE uart active
        // reset RN4871
        PORTB.OUTSET = PIN5_bm; // BLE RESET HIGH
        PORTB.OUTCLR = PIN5_bm;
        _delay_ms(10);
        PORTC.OUTSET = PIN5_bm;
        _delay_ms(100); // wait for RN4871 to boot up and be ready
        rn4871_setFuses();
}

// wait for a response from the RN4871
void rn4871_waitfor(char endchar){
        char str[10];
        char* str_ptr = str;
        *str_ptr = usart_get();
        while( *str_ptr != endchar ){
                ++str_ptr;
                *str_ptr = usart_get();
        }
        ++str_ptr;
        *str_ptr = '\0';
        //dbg_print_str(str); //print the response for debugging
}

// setup RN4871
void rn4871_setFuses(){

    rn4871_commandMode();
    printf('+');//Echo on
    //RN4871 User guide pages 16-27
    printf("SF,1");//Reset to factory configuration
    rn4871_waitfor('K'); // get response, should be: "AOK"
    _delay_ms(1);
    printf("SN,Diana");//Set device name to Diana
    rn4871_waitfor('K'); // get response, should be: "AOK"
    _delay_ms(1);
    /*SS,<hex8> sets the default services to be supported by the GAP server role*/
    printf("SS,C0");//Support device info and UART transparent services
    rn4871_waitfor('K'); // get response, should be: "AOK"
    _delay_ms(1);
    /*SR,<hex16> stes the supported feature of the device*/
    printf("SR,0100");//Enable UART transparent without ACK
    rn4871_waitfor('K'); // get response, should be: "AOK"
    _delay_ms(1);
    printf("R,1");//Reboot to make changes effective (page 37)
    rn4871_waitfor('T'); // get response, should be: "%REBOOT%"
    _delay_ms(5);
    /**********************SB,<H8> to Set the BAUD rate*************************
     * 00: 921600
     * 01: 460800
     * 02: 230400
     * 03: 115200
     * 04: 57600
     * 05: 38400
     * 06: 28800
     * 07: 19200
     * 08: 14400
     * 09: 9600
     * 0A: 4800
     * 0B: 2400
```

```
     ***************************************************************************/
    printf("SB,03");
    rn4871_waitfor('K'); // get response, should be: "AOK"
    _delay_ms(1);

}
void rn4871_commandMode()
{
        printf("$$$");
        _delay_ms(100);
}
void rn4871_dataMode()
{
        printf("---")
        rn4871_waitfor('D');//wait for "END"
}
bool rn4871_conStat()
{
    bool paired=0;
    rn4871_commandMode();
    //Command GK gets current connection status
    printf("GK");
    const char * cond= usart_getstr(28);
    //28 because is the max length this get command will retrieve
    if(cond=="none")// the output is "none"
    {
       paired=0;
    }else
    {
       paired=1;
    }
    rn4871_dataMode();
    return paired;
}
// setup wakeup and reset pins, reset RN4871
// set wakeup pin high to put RN4871 to sleep (RF still works while sleeping, but UART doesn't)
void rn4871_sleep(){
        PORTB.OUTSET = PIN4_bm; // put RN4871 in sleep mode
}
// wakeup RN4871 from sleep and wait 5ms until UART is ready
void rn4871_wakeup(){
        PORTB.OUTCLR = PIN4_bm; // wake up RN4871
        _delay_ms(5); // RN4871 needs 5ms to wake up until UART is responsive
}
```

ICM20600

This library contains a fair amount of code taken from Seeed Technology inc. Originally developed for Arduino and modified to better suit this application.

Header File:

```
/*
 * ICM20600.h
 * A library for Grove - IMU 9DOF(ICM20600 + AK09918)
 *
 * Copyright (c) 2018 seeed technology inc.
 * Website    : www.seeed.cc
 * Author     : Jerry Yip
 * Create Time: 2018-06
 * Version    : 0.1
 * Change Log :
 *
 * The MIT License (MIT)
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
```

```
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * File:   icm20600.h
 *
 *
 * This file contains all the function declarations for what regards to the 6-axis
 * accelerometer ICM20600 and contains a fair amount of code taken from the library mentioned above
 */

#ifndef ICM20600_H
#define ICM20600_H

#include <stdbool.h>
#include <util/delay_basic.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include "i2c.h"

#ifdef  __cplusplus
extern "C" {
#endif



/*************************************************************
 ICM20600 I2C Register
 *************************************************************/
//Taken from the Register MAP from de ICM-20600 datasheet pages 28-29
#define ICM20600_XG_OFFS_TC_H           0x04
#define ICM20600_XG_OFFS_TC_L           0x05
#define ICM20600_YG_OFFS_TC_H           0x07
#define ICM20600_YG_OFFS_TC_L           0x08
#define ICM20600_ZG_OFFS_TC_H           0x0a
#define ICM20600_ZG_OFFS_TC_L           0x0b
#define ICM20600_SELF_TEST_X_ACCEL      0x0d
#define ICM20600_SELF_TEST_Y_ACCEL      0x0e
#define ICM20600_SELF_TEST_Z_ACCEL      0x0f
#define ICM20600_XG_OFFS_USRH           0x13
#define ICM20600_XG_OFFS_USRL           0x14
#define ICM20600_YG_OFFS_USRH           0x15
#define ICM20600_YG_OFFS_USRL           0x16
#define ICM20600_ZG_OFFS_USRH           0x17
#define ICM20600_ZG_OFFS_USRL           0x18
#define ICM20600_SMPLRT_DIV             0x19
#define ICM20600_CONFIG                 0x1a
#define ICM20600_GYRO_CONFIG            0x1b
#define ICM20600_ACCEL_CONFIG           0x1c
#define ICM20600_ACCEL_CONFIG2          0x1d
#define ICM20600_GYRO_LP_MODE_CFG       0x1e
#define ICM20600_ACCEL_WOM_X_THR        0x20
```

```
#define ICM20600_ACCEL_WOM_Y_THR        0x21
#define ICM20600_ACCEL_WOM_Z_THR        0x22
#define ICM20600_FIFO_EN                0x23
#define ICM20600_FSYNC_INT              0x36
#define ICM20600_INT_PIN_CFG            0x37
#define ICM20600_INT_ENABLE             0x38
#define ICM20600_FIFO_WM_INT_STATUS     0x39
#define ICM20600_INT_STATUS             0x3a
#define ICM20600_ACCEL_XOUT_H           0x3b
#define ICM20600_ACCEL_XOUT_L           0x3c
#define ICM20600_ACCEL_YOUT_H           0x3d
#define ICM20600_ACCEL_YOUT_L           0x3e
#define ICM20600_ACCEL_ZOUT_H           0x3f
#define ICM20600_ACCEL_ZOUT_L           0x40
#define ICM20600_TEMP_OUT_H             0x41
#define ICM20600_TEMP_OUT_L             0x42
#define ICM20600_GYRO_XOUT_H            0x43
#define ICM20600_GYRO_XOUT_L            0x44
#define ICM20600_GYRO_YOUT_H            0x45
#define ICM20600_GYRO_YOUT_L            0x46
#define ICM20600_GYRO_ZOUT_H            0x47
#define ICM20600_GYRO_ZOUT_L            0x48
#define ICM20600_SELF_TEST_X_GYRO       0x50
#define ICM20600_SELF_TEST_Y_GYRO       0x51
#define ICM20600_SELF_TEST_Z_GYRO       0x52
#define ICM20600_FIFO_WM_TH1            0x60
#define ICM20600_FIFO_WM_TH2            0x61
#define ICM20600_SIGNAL_PATH_RESET      0x68
#define ICM20600_ACCEL_INTEL_CTRL       0x69
#define ICM20600_USER_CTRL              0x6A
#define ICM20600_PWR_MGMT_1             0x6b
#define ICM20600_PWR_MGMT_2             0x6c
#define ICM20600_I2C_IF                 0x70
#define ICM20600_FIFO_COUNTH            0x72
#define ICM20600_FIFO_COUNTL            0x73
#define ICM20600_FIFO_R_W               0x74
#define ICM20600_WHO_AM_I               0x75
#define ICM20600_XA_OFFSET_H            0x77
#define ICM20600_XA_OFFSET_L            0x78
#define ICM20600_YA_OFFSET_H            0x7a
#define ICM20600_YA_OFFSET_L            0x7b
#define ICM20600_ZA_OFFSET_H            0x7d
#define ICM20600_ZA_OFFSET_L            0x7e


// Gyroscope scale range
enum gyro_scale_type_t
{
    RANGE_250_DPS = 0,
    RANGE_500_DPS,
    RANGE_1K_DPS,
    RANGE_2K_DPS,
};

// Accelerometer scale range
enum acc_scale_type_t
{
    RANGE_2G = 0,
    RANGE_4G,
    RANGE_8G,
    RANGE_16G,
};

// Gyroscope output data rate
enum gyro_lownoise_odr_type_t
{
    GYRO_RATE_8K_BW_3281 = 0,
    GYRO_RATE_8K_BW_250,
    GYRO_RATE_1K_BW_176,
```

```
    GYRO_RATE_1K_BW_92,
    GYRO_RATE_1K_BW_41,
    GYRO_RATE_1K_BW_20,
    GYRO_RATE_1K_BW_10,
    GYRO_RATE_1K_BW_5,
};

// Accelerometer output data rate
enum acc_lownoise_odr_type_t
{
    ACC_RATE_4K_BW_1046 = 0,
    ACC_RATE_1K_BW_420,
    ACC_RATE_1K_BW_218,
    ACC_RATE_1K_BW_99,
    ACC_RATE_1K_BW_44,
    ACC_RATE_1K_BW_21,
    ACC_RATE_1K_BW_10,
    ACC_RATE_1K_BW_5,
};

// Averaging filter settings for Low Power Accelerometer mode
enum acc_averaging_sample_type_t
{
    ACC_AVERAGE_4 = 0,
    ACC_AVERAGE_8,
    ACC_AVERAGE_16,
    ACC_AVERAGE_32,
};

// Averaging filter configuration for low-power gyroscope mode
enum gyro_averaging_sample_type_t
{
    GYRO_AVERAGE_1 = 0,
    GYRO_AVERAGE_2,
    GYRO_AVERAGE_4,
    GYRO_AVERAGE_8,
    GYRO_AVERAGE_16,
    GYRO_AVERAGE_32,
    GYRO_AVERAGE_64,
    GYRO_AVERAGE_128,
};

// ICM20600 power mode
enum icm20600_power_type_t
{
    ICM_SLEEP_MODE = 0,
    ICM_STANDYBY_MODE,
    ICM_ACC_LOW_POWER,
    ICM_ACC_LOW_NOISE,
    ICM_GYRO_LOW_POWER,
    ICM_GYRO_LOW_NOISE,
    ICM_6AXIS_LOW_POWER,
    ICM_6AXIS_LOW_NOISE,
};

/***************************************
           FUNCTION DECLARATION
***************************************/
void icm_assign_addr(bool AD0,icm20600 sensor);
uint8_t icm_who_am_i(icm20600 sensor);
void icm_init(icm20600 sensor);
void   icm_set_pow_mode(icm20600_power_type_t mode,icm20600 sensor);
void icm_set_sample_rate_div(uint8_t div,icm20600 sensor);
void icm_set_acc_scalerange(acc_scale_type_t range,icm20600 sensor);
void icm_set_acc_avrgsample(acc_averaging_sample_type_t sample, icm20600 sensor);
void icm_set_acc_outp_data_rate(acc_lownoise_odr_type_t odr, icm20600 sensor);
void icm_set_gyro_scalerange(gyro_scale_type_t range,icm20600 sensor);
void icm_set_gyro_avrgsample(gyro_averaging_sample_type_t sample, icm20600 sensor);
void icm_set_gyro_outp_data_rate(gyro_lownoise_odr_type_t odr,icm20600 sensor);
```

```
void icm_get_acc(int16_t* x, int16_t* y, int16_t* z, icm20600 sensor);
int16_t icm_get_acc_x(icm20600 sensor);
int16_t icm_get_acc_y(icm20600 sensor);
int16_t icm_get_acc_z(icm20600 sensor);
int16_t icm_get_acc_rawx(icm20600 sensor);
int16_t icm_get_acc_rawy(icm20600 sensor);
int16_t icm_get_acc_rawz(icm20600 sensor);
void icm_get_gyro(int16_t* x, int16_t* y, int16_t* z, icm20600 sensor);
int16_t icm_get_gyro_x(icm20600 sensor);
int16_t icm_get_gyro_y(icm20600 sensor);
int16_t icm_get_gyro_z(icm20600 sensor);
int16_t icm_get_gyro_rawx(icm20600 sensor);
int16_t icm_get_gyro_rawy(icm20600 sensor);
int16_t icm_get_gyro_rawz(icm20600 sensor);
int16_t icm_get_temp(icm20600 sensor);
void icm_reset(icm20600 sensor);

#ifdef  __cplusplus
}
#endif

#endif  /* ICM20600_H */
```

Source File:

```
#include "icm20600.h"
typedef struct{
    /*
     * This device may have two addresses
     * 0x68 or 0b1101000 when the pin ADO is pulled down
     * 0x69 or 0b1101001 when the pin ADO is pulled up
     */
    uint8_t addr=0x68;//Assign the device this address since in our circuit ADO is pulled down
    uint8_t buffer[16];
    uint16_t acc_scale, gyro_scale;
}icm20600;

/*
 *This function verifies the identity of the device by reading the WHO_AM_I
 * register, getting the ID. The default value of this register is 0x11
 */
uint8_t icm_who_am_i(icm20600 sensor)
{
    //readByte(sensor.addr, ICM20600_WHO_AM_I, sensor.buffer);
    icm_bus_readBytes(sensor,ICM20600_WHO_AM_I,sensor.buffer,1);
    return sensor.buffer[0];
}


/*
 * This function initializes the sensor
 * In the configuration register writes 0x00, dissables the FIFO writing 0x00 in
 * this register, and sets the power, the gyroscope and the accelerometer.
 */
void icm_init(icm20600 sensor)
{
    // configuration
    uint8_t data=0x00;
    icm_bus_writeBytes(sensor,ICM20600_CONFIG,data,1);
    // disable fifo
    data=0x00;
    icm_bus_writeBytes(sensor,ICM20600_FIFO_EN,data,1);

    // set default power mode
    icm_set_pow_mode(ICM_6AXIS_LOW_POWER,sensor);

    // gyro config
    icm_set_gyro_scalerange(RANGE_2K_DPS, sensor);
```

```
    icm_set_gyro_outp_data_rate(GYRO_RATE_1K_BW_176, sensor);
    icm_set_gyro_avrgsample(GYRO_AVERAGE_1, sensor);

    // accel config
    icm_set_acc_scalerange(RANGE_16G, sensor);
    icm_set_acc_outp_data_rate(ACC_RATE_1K_BW_420, sensor);
    icm_set_acc_avrgsample(ACC_AVERAGE_4, sensor);
}
 void   icm_set_pow_mode(icm20600_power_type_t mode,icm20600 sensor)
{
    uint8_t data_pwr1;
    uint8_t data_pwr2 = 0x00;
    uint8_t data_gyro_lp;
    icm_bus_readBytes(sensor,ICM20600_PWR_MGMT_1,sensor.buffer,1);
    data_pwr1 = sensor.buffer[0];
    data_pwr1 &= 0x8f; // 0b10001111
    icm_bus_readBytes(sensor,ICM20600_GYRO_LP_MODE_CFG,sensor.buffer ,1);
    data_gyro_lp = sensor.buffer[0];
    // When set to ?1? low-power gyroscope mode is enabled. Default setting is 0
    data_gyro_lp &= 0x7f;               // 0b01111111
    switch(mode)
    {
        case ICM_SLEEP_MODE:
            data_pwr1 |= 0x40;          // set 0b01000000
        break;

        case ICM_STANDYBY_MODE:
            data_pwr1 |= 0x10;          // set 0b00010000
            data_pwr2 = 0x38;           // 0x00111000 disable acc
        break;

        case ICM_ACC_LOW_POWER:
            data_pwr1 |= 0x20;          // set bit5 0b00100000
            data_pwr2 = 0x07;           //0x00000111 disable gyro
        break;

        case ICM_ACC_LOW_NOISE:
            data_pwr1 |= 0x00;
            data_pwr2 = 0x07;           //0x00000111 disable gyro
        break;

        case ICM_GYRO_LOW_POWER:
            data_pwr1 |= 0x00;          // dont set bit5 0b00000000
            data_pwr2 = 0x38;           // 0x00111000 disable acc
            data_gyro_lp |= 0x80;
        break;

        case ICM_GYRO_LOW_NOISE:
            data_pwr1 |= 0x00;
            data_pwr2 = 0x38;           // 0x00111000 disable acc
        break;

        case ICM_6AXIS_LOW_POWER:
            data_pwr1 |= 0x00;          // dont set bit5 0b00100000
            data_gyro_lp |= 0x80;
        break;

        case ICM_6AXIS_LOW_NOISE:
            data_pwr1 |= 0x00;
        break;

        default:
        break;
    }

    icm_bus_writeBytes(sensor,ICM20600_PWR_MGMT_1,data_pwr1,1);
    icm_bus_writeBytes(sensor,ICM20600_PWR_MGMT_2,data_pwr2,1);
    icm_bus_writeBytes(sensor,ICM20600_GYRO_LP_MODE_CFG,data_gyro_lp,1);
}
```

```
// SAMPLE_RATE = 1KHz / (1 + div)
// work for low-power gyroscope and low-power accelerometer and low-noise accelerometer
void icm_set_sample_rate_div(uint8_t div,icm20600 sensor)
{
    icm_bus_writeBytes(sensor,ICM20600_SMPLRT_DIV,div,1);
}
void icm_set_acc_scalerange(acc_scale_type_t range,icm20600 sensor)
{
    uint8_t data;
    icm_bus_readBytes(sensor,ICM20600_ACCEL_CONFIG,sensor.buffer,1);
    data = sensor.buffer[0];
    data &= 0xe7; // 0b 1110 0111

    switch(range)
    {
        case RANGE_2G:
            data |= 0x00;   // 0bxxx00xxx
            sensor.acc_scale = 4000;
        break;

        case RANGE_4G:
            data |= 0x08;   // 0bxxx01xxx
            sensor.acc_scale = 8000;
        break;

        case RANGE_8G:
            data |= 0x10;   // 0bxxx10xxx
            sensor.acc_scale = 16000;
        break;

        case RANGE_16G:
            data |= 0x18;   // 0bxxx11xxx
            sensor.acc_scale = 32000;
        break;

        default:
        break;
    }
    sensor.buffer=data;
    icm_bus_writeBytes(sensor,ICM20600_ACCEL_CONFIG,data,1);
}


// for low power mode only

void icm_set_acc_avrgsample(acc_averaging_sample_type_t sample, icm20600 sensor)
{
    uint8_t data = 0;
    icm_bus_readBytes(sensor,ICM20600_ACCEL_CONFIG2,sensor.buffer,1);
    data = sensor.buffer[0];

    data &= 0xcf; // & 0b11001111
    switch(sample)
    {
        case ACC_AVERAGE_4:
            data |= 0x00; // 0bxx00xxxx
        break;

        case ACC_AVERAGE_8:
            data |= 0x10; // 0bxx01xxxx
        break;

        case ACC_AVERAGE_16:
            data |= 0x20; // 0bxx10xxxx
        break;

        case ACC_AVERAGE_32:
            data |= 0x30; // 0bxx11xxxx
```

```
        break;

        default:
        break;
    }
    icm_bus_writeBytes(sensor,ICM20600_ACCEL_CONFIG2,data,1);
}


void icm_set_acc_outp_data_rate(acc_lownoise_odr_type_t odr, icm20600 sensor)
{
    uint8_t data;
    icm_bus_readBytes(sensor,ICM20600_ACCEL_CONFIG2,sensor.buffer,1);
    data = sensor.buffer[0];
    data &= 0xf0;  // 0b11110000

    switch(odr)
    {
        case ACC_RATE_4K_BW_1046:
            data |= 0x08;
        break;

        case ACC_RATE_1K_BW_420:
            data |= 0x07;
        break;

        case ACC_RATE_1K_BW_218:
            data |= 0x01;
        break;

        case ACC_RATE_1K_BW_99:
            data |= 0x02;
        break;

        case ACC_RATE_1K_BW_44:
            data |= 0x03;
        break;

        case ACC_RATE_1K_BW_21:
            data |= 0x04;
        break;

        case ACC_RATE_1K_BW_10:
            data |= 0x05;
        break;

        case ACC_RATE_1K_BW_5:
            data |= 0x06;
        break;

        default:
        break;
    }

    icm_bus_writeBytes(sensor,ICM20600_ACCEL_CONFIG2,data,1);
}


void icm_set_gyro_scalerange(gyro_scale_type_t range,icm20600 sensor)
{
    uint8_t data = 0;
    icm_bus_readBytes(sensor,ICM20600_GYRO_CONFIG,sensor.buffer,1);
    data = sensor.buffer[0];
    data &= 0xe7; // 0b11100111

    switch(range)
    {
        case RANGE_250_DPS:
            data |= 0x00;   // 0bxxx00xxx
```

```
            sensor.gyro_scale = 500;
        break;

        case RANGE_500_DPS:
            data |= 0x08;   // 0bxxx00xxx
            sensor.gyro_scale = 1000;
        break;

        case RANGE_1K_DPS:
            data |= 0x10;   // 0bxxx10xxx
            sensor.gyro_scale = 2000;
        break;

        case RANGE_2K_DPS:
            data |= 0x18;   // 0bxxx11xxx
            sensor.gyro_scale = 4000;
        break;

        default:
        break;
    }
    icm_bus_writeBytes(sensor,ICM20600_GYRO_CONFIG,data,1);
}


void icm_set_gyro_avrgsample(gyro_averaging_sample_type_t sample, icm20600 sensor)
{
    uint8_t data = 0;
    icm_bus_readBytes(sensor,ICM20600_GYRO_LP_MODE_CFG,sensor.buffer,1);
    data = sensor.buffer[0];

    data &= 0x8f;           // 0b10001111
    switch(sample)
    {
        case GYRO_AVERAGE_1:
            data |= 0x00; // 0bx000xxxx
        break;

        case GYRO_AVERAGE_2:
            data |= 0x10; // 0bx001xxxx
        break;

        case GYRO_AVERAGE_4:
            data |= 0x20; // 0bx010xxxx
        break;

        case GYRO_AVERAGE_8:
            data |= 0x30; // 0bx011xxxx
        break;

        case GYRO_AVERAGE_16:
            data |= 0x40; // 0bx100xxxx
        break;

        case GYRO_AVERAGE_32:
            data |= 0x50; // 0bx101xxxx
        break;

        case GYRO_AVERAGE_64:
            data |= 0x60;
        break;

        case GYRO_AVERAGE_128:
            data |= 0x70;
        break;


        default:
        break;
```

```
    }
    icm_bus_writeBytes(sensor,ICM20600_GYRO_LP_MODE_CFG,data,1);
}

void icm_set_gyro_outp_data_rate(gyro_lownoise_odr_type_t odr,icm20600 sensor)
{
    uint8_t data;
    icm_bus_readBytes(sensor,ICM20600_CONFIG,sensor.buffer,1);
    data = sensor.buffer[0];
    data &= 0xf8;  // DLPF_CFG[2:0] 0b11111000

    switch(odr)
    {
        case GYRO_RATE_8K_BW_3281:
            data |= 0x07;
        break;
        case GYRO_RATE_8K_BW_250:
            data |= 0x00;
        break;
        case GYRO_RATE_1K_BW_176:
            data |= 0x01;
        break;
        case GYRO_RATE_1K_BW_92:
            data |= 0x02;
        break;
        case GYRO_RATE_1K_BW_41:
            data |= 0x03;
        break;
        case GYRO_RATE_1K_BW_20:
            data |= 0x04;
        break;
        case GYRO_RATE_1K_BW_10:
            data |= 0x05;
        break;
        case GYRO_RATE_1K_BW_5:
            data |= 0x06;
        break;
    }
    icm_bus_writeBytes(sensor,ICM20600_CONFIG,data,1);
}
void icm_get_acc(int16_t* x, int16_t* y, int16_t* z,icm20600 sensor)
{
    *x = icm_get_acc_x(sensor);
    *y = icm_get_acc_y(sensor);
    *z = icm_get_acc_z(sensor);
}
void icm_get_acc_off(int16_t* x, int16_t* y, int16_t* z,icm20600 sensor)
{
    *x = icm_get_off_accX(sensor);
    *y = icm_get_off_accY(sensor);
    *z = icm_get_off_accZ(sensor);
}

int16_t icm_get_acc_x(icm20600 sensor)
{
    int32_t raw_data = icm_get_acc_rawx(sensor);
    raw_data = (raw_data * sensor.acc_scale) >> 16;
    return (int16_t)raw_data;
}

int16_t icm_get_acc_y(icm20600 sensor)
{
    int32_t raw_data = icm_get_acc_rawy(sensor);
    raw_data = (raw_data * sensor.acc_scale) >> 16;
    return (int16_t)raw_data;
}

int16_t icm_get_acc_z(icm20600 sensor)
{
```

```
    int32_t raw_data = icm_get_acc_rawz(sensor);
    raw_data = (raw_data * sensor.acc_scale) >> 16;
    return (int16_t)raw_data;
}

int16_t icm_get_acc_rawx(icm20600 sensor)
{
    icm_bus_readBytes(sensor,ICM20600_ACCEL_XOUT_H,sensor.buffer,2);
    return ((int16_t)sensor.buffer[0] << 8) + sensor.buffer[1];
}

int16_t icm_get_acc_rawy(icm20600 sensor)
{
    icm_bus_readBytes(sensor,ICM20600_ACCEL_YOUT_H,sensor.buffer,2);
    return ((int16_t)sensor.buffer[0] << 8) + sensor.buffer[1];
}

int16_t icm_get_acc_rawz(icm20600 sensor)
{
    icm_bus_readBytes(sensor,ICM20600_ACCEL_ZOUT_H,sensor.buffer,2);
    return ((int16_t)sensor.buffer[0] << 8) + sensor.buffer[1];
}
int16_t icm_get_off_accX(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_XA_OFFSET_H ,sensor.buffer,2)
    rawdata=((uint16_t)sensor.buffer[0]<<8)+sensor.buffer[1];
    rawdata = (rawdata * sensor.acc_scale) >> 16;
    return rawdata;
}
int16_t icm_get_off_accY(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_YA_OFFSET_H ,sensor.buffer,2)
    rawdata=((uint16_t)sensor.buffer[0]<<8)+sensor.buffer[1];
    rawdata = (rawdata * sensor.acc_scale) >> 16;
    return rawdata;
}
int16_t icm_get_off_accZ(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_ZA_OFFSET_H ,sensor.buffer,2)
    rawdata=((uint16_t)sensor.buffer[0]<<8)+sensor.buffer[1];
    rawdata = (rawdata * sensor.acc_scale) >> 16;
    return rawdata;
}

void icm_get_gyro(int16_t* x, int16_t* y, int16_t* z,icm20600 sensor)
{
    *x = icm_get_gyro_x(sensor);
    *y = icm_get_gyro_y(sensor);
    *z = icm_get_gyro_z(sensor);
}
void icm_get_gyro_off(int16_t* x, int16_t* y, int16_t* z,icm20600 sensor)
{
    *x = icm_get_off_gyrX(sensor);
    *y = icm_get_off_gyrY(sensor);
    *z = icm_get_off_gyrZ(sensor);
}
int16_t icm_get_gyro_x(icm20600 sensor)
{
    int32_t raw_data = icm_get_gyro_rawx(sensor);
    raw_data = (raw_data * sensor.gyro_scale) >> 16;
    return (int16_t)raw_data;
}

int16_t icm_get_gyro_y(icm20600 sensor)
{
    int32_t raw_data = icm_get_gyro_rawy(sensor);
```

```
    raw_data = (raw_data * sensor.gyro_scale) >> 16;
    return (int16_t)raw_data;
}

int16_t icm_get_gyro_z(icm20600 sensor)
{
    int32_t raw_data = icm_get_gyro_rawz(sensor);
    raw_data = (raw_data * sensor.gyro_scale) >> 16;
    return (int16_t)raw_data;
}

int16_t icm_get_gyro_rawx(icm20600 sensor)
{
    icm_bus_readBytes(sensor,ICM20600_GYRO_XOUT_H,sensor.buffer,2);
    return ((int16_t)sensor.buffer[0] << 8) + sensor.buffer[1];
}

int16_t icm_get_gyro_rawy(icm20600 sensor)
{
    icm_bus_readBytes(sensor,ICM20600_GYRO_YOUT_H,sensor.buffer,2);
    return ((int16_t)sensor.buffer[0] << 8) + sensor.buffer[1];
}

int16_t icm_get_gyro_rawz(icm20600 sensor)
{
    icm_bus_readBytes(sensor,ICM20600_GYRO_ZOUT_H,sensor.buffer,2);
    return ((int16_t)sensor.buffer[0] << 8) + sensor.buffer[1];
}


int16_t icm_get_off_gyrX(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_XG_OFFS_USRH,sensor.buffer,2)
    rawdata=((uint16_t)sensor.buffer[0]<<8)+sensor.buffer[1];
    rawdata = (rawdata * sensor.gyro_scale) >> 16;
    return rawdata;
}
int16_t icm_get_off_gyrY(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_YG_OFFS_USRH,sensor.buffer,2)
    rawdata=((uint16_t)sensor.buffer[0]<<8)+sensor.buffer[1];
    rawdata = (rawdata * sensor.gyro_scale) >> 16;
    return rawdata;
}
int16_t icm_get_off_gyrZ(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_ZG_OFFS_USRH,sensor.buffer,2)
    rawdata=((uint16_t)sensor.buffer[0]<<8)+sensor.buffer[1];
    rawdata = (rawdata * sensor.gyro_scale) >> 16;
    return rawdata;
}
int16_t icm_get_temp(icm20600 sensor)
{
    uint16_t rawdata;
    icm_bus_readBytes(sensor,ICM20600_TEMP_OUT_H,sensor.buffer,2)
    rawdata = (((uint16_t)sensor.buffer[0]) << 8) + sensor.buffer[1];
    return (int16_t)(rawdata/327 + 25);
}

/*
 * sensor is the information of the sensor
 * regAddr is the register inside the device that is been accessed
 * bytes is the number of bytes that will be read
 */
void icm_bus_readBytes(icm20600 sensor,uint8_t regAddr,uint8_t data, uint8_t bytes)
{
```

```
    twi_package_t packet = {
                                .chip   = sensor.addr,
                                .buffer = (void*) data,
                                .length = bytes,
                                .addr= regAddr,
                                // Wait if bus is busy
                                .no_wait = false};
        i2c_master_transfer(&packet, false);

        i2c_master_transfer(&packet, true);
}

/*
 * sensor is the information of the sensor
 * regAddr is the register inside the device that is been accessed
 * bytes is the number of bytes that will be written
 */
void icm_bus_writeBytes(icm20600 sensor,uint8_t regAddr,uint8_t data, uint8_t bytes)
{
    twi_package_t packet = {
                                .chip   = sensor.addr,
                                .buffer = (void*) data,
                                .length = bytes,
                                .addr= regAddr,
                                // Wait if bus is busy
                                .no_wait = false};
        i2c_master_transfer(&packet, false);

}
```

I²C Protocol:

The I²C Master library used is a modification of a library that already existed, written by Atmel Corporation. Some pieces of the original code were also used to build up part of the I²C library.

I²C Master:

Header File:

```
/**
 * \file
 *
 * \brief TWI driver for AVR.
 *
 * This file defines a useful set of functions for the TWI interface on AVR
 * devices.
 *
 * Copyright (c) 2010-2015 Atmel Corporation. All rights reserved.
 *
 * \asf_license_start
 *
 * \page License
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The name of Atmel may not be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * 4. This software may only be redistributed and used in connection with an
 *    Atmel microcontroller product.
```

```
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * \asf_license_stop
 *
 */
/*
 * Support and FAQ: visit <a href="http://www.atmel.com/design-support/">Atmel Support</a>
 */
#ifndef I2C_MASTER_H
#define I2C_MASTER_H

#ifdef  __cplusplus
extern "C" {
#endif


#ifdef  __cplusplus
}
#endif
#ifdef __GNUC__
#define cpu_irq_enable() sei()
#define cpu_irq_disable() cli()
#define barrier() asm volatile("" ::: "memory")
#else
#define cpu_irq_enable() __enable_interrupt()
#define cpu_irq_disable() __disable_interrupt()
#define barrier() asm("")
#endif

#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdbool.h>

/*************************************************************
 Useful Definitions
 *************************************************************/

#define TWI_BAUD(F_SYS, F_TWI) ((F_SYS / (2 * F_TWI)) - 5)

typedef unsigned char TWI_t;
typedef struct {
        //! TWI chip address to communicate with.
        uint8_t chip;
        //! TWI address/commands to issue to the other chip (node). register
        uint8_t addr/*[3]*/;
        //! Length of the TWI data address segment (1-3 bytes).
        //int addr_length;
        //! Where to find the data to be written.
        void *buffer;
        //! How many bytes do we want to write.
        unsigned int length;
        //! Whether to wait if bus is busy (false) or return immediately (true)
        bool no_wait;
} twi_package_t;
```

```
static inline void i2c_fast_mode_enable()
{
        TWI0_CTRLA |= TWI_FMPEN_bm;
}
static inline void i2c_fast_mode_disable()
{
        TWI0_CTRLA &= (~TWI_FMPEN_bm);
}

/*************************************************************
 STATUS CODES AND CATEGORIES
 *************************************************************/
/** Status code error categories. */
enum status_categories {
        STATUS_CATEGORY_OK     = 0x00,
        STATUS_CATEGORY_COMMON = 0x10,
        STATUS_CATEGORY_ANALOG = 0x30,
        STATUS_CATEGORY_COM    = 0x40,
        STATUS_CATEGORY_IO     = 0x50,
};

/**
 * Status code that may be returned by shell commands and protocol
 * implementations.
 *
 * \note Any change to these status codes and the corresponding
 * message strings is strictly forbidden. New codes can be added,
 * however, but make sure that any message string tables are updated
 * at the same time.
 */
enum status_code {
        STATUS_OK         = STATUS_CATEGORY_OK | 0x00,
        STATUS_VALID_DATA = STATUS_CATEGORY_OK | 0x01,
        STATUS_NO_CHANGE  = STATUS_CATEGORY_OK | 0x02,
        STATUS_ABORTED    = STATUS_CATEGORY_OK | 0x04,
        STATUS_BUSY       = STATUS_CATEGORY_OK | 0x05,
        STATUS_SUSPEND    = STATUS_CATEGORY_OK | 0x06,

        STATUS_ERR_IO                  = STATUS_CATEGORY_COMMON | 0x00,
        STATUS_ERR_REQ_FLUSHED         = STATUS_CATEGORY_COMMON | 0x01,
        STATUS_ERR_TIMEOUT             = STATUS_CATEGORY_COMMON | 0x02,
        STATUS_ERR_BAD_DATA            = STATUS_CATEGORY_COMMON | 0x03,
        STATUS_ERR_NOT_FOUND           = STATUS_CATEGORY_COMMON | 0x04,
        STATUS_ERR_UNSUPPORTED_DEV     = STATUS_CATEGORY_COMMON | 0x05,
        STATUS_ERR_NO_MEMORY           = STATUS_CATEGORY_COMMON | 0x06,
        STATUS_ERR_INVALID_ARG         = STATUS_CATEGORY_COMMON | 0x07,
        STATUS_ERR_BAD_ADDRESS         = STATUS_CATEGORY_COMMON | 0x08,
        STATUS_ERR_BAD_FORMAT          = STATUS_CATEGORY_COMMON | 0x0A,
        STATUS_ERR_BAD_FRQ             = STATUS_CATEGORY_COMMON | 0x0B,
        STATUS_ERR_DENIED              = STATUS_CATEGORY_COMMON | 0x0c,
        STATUS_ERR_ALREADY_INITIALIZED = STATUS_CATEGORY_COMMON | 0x0d,
        STATUS_ERR_OVERFLOW            = STATUS_CATEGORY_COMMON | 0x0e,
        STATUS_ERR_NOT_INITIALIZED     = STATUS_CATEGORY_COMMON | 0x0f,

        STATUS_ERR_SAMPLERATE_UNAVAILABLE = STATUS_CATEGORY_ANALOG | 0x00,
        STATUS_ERR_RESOLUTION_UNAVAILABLE = STATUS_CATEGORY_ANALOG | 0x01,

        STATUS_ERR_BAUDRATE_UNAVAILABLE = STATUS_CATEGORY_COM | 0x00,
        STATUS_ERR_PACKET_COLLISION     = STATUS_CATEGORY_COM | 0x01,
        STATUS_ERR_PROTOCOL             = STATUS_CATEGORY_COM | 0x02,

        STATUS_ERR_PIN_MUX_INVALID = STATUS_CATEGORY_IO | 0x00,
        /**
         * \brief Operation in progress
         *
         * This status code is for driver-internal use when an operation
         * is currently being performed.
         *
         * \note Drivers should never return this status code to any
```

```
         * callers. It is strictly for internal use.
         */
        OPERATION_IN_PROGRESS = -128,
};

typedef enum status_code status_code_t;

/*************************************************************
 FUNCTION DEFINITONS
 *************************************************************/

void i2c_master_init(void);
/*! \brief Perform a TWI master write or read transfer.
 *
 * This function is a TWI Master write or read transaction.
 *
 * \param twi       Base address of the TWI (i.e. &TWI_t).
 * \param package   Package information and data
 *                  (see \ref twi_package_t)
 * \param read      Selects the transfer direction
 *
 * \return  status_code_t
 *      - STATUS_OK if the transfer completes
 *      - STATUS_BUSY to indicate an unavailable bus
 *      - STATUS_ERR_IO to indicate a bus transaction error
 *      - STATUS_ERR_NO_MEMORY to indicate buffer errors
 *      - STATUS_ERR_PROTOCOL to indicate an unexpected bus state
 *      - STATUS_ERR_INVALID_ARG to indicate invalid arguments.
 */
status_code_t i2c_master_transfer(/*TWI_t *twi,*/ const twi_package_t *package, bool read);

//Enable master mode
static inline void i2c_master_enable()
{
        TWI0_MCTRLA |= TWI_ENABLE_bm;
}

//disable master mode
static inline void i2c_master_disable()
{
        TWI0_MCTRLA &= (~TWI_ENABLE_bm);
}
//Checks if the master mode is enabled
static inline uint8_t i2c_master_is_enabled()
{
        return (uint8_t)(TWI0_MCTRLA & TWI_ENABLE_bm);
}
#endif  /* I2C_MASTER_H */
```

Source File:

```
/**
 * \file
 *
 * \brief XMEGA TWI master source file.
 *
 * Copyright (c) 2010-2015 Atmel Corporation. All rights reserved.
 *
 * \asf_license_start
 *
 * \page License
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
```

```
 *    this list of conditions and the following disclaimer in the documentation
 *    and/or other materials provided with the distribution.
 *
 * 3. The name of Atmel may not be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * 4. This software may only be redistributed and used in connection with an
 *    Atmel microcontroller product.
 *
 * THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE
 * EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
 * ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 *
 * \asf_license_stop
 *
 */
/*
 * Support and FAQ: visit <a href="http://www.atmel.com/design-support/">Atmel Support</a>
 */
#include "i2c_master.h"
/* Master Transfer Descriptor */

static struct {
        //TWI_t *                bus;        // Bus register interface
        twi_package_t *        pkg;        // Bus message descriptor
        int                    addr_count; // Bus transfer address data counter
        unsigned int           data_count; // Bus transfer payload data counter
        bool                   read;       // Bus transfer direction
        bool                   locked;     // Bus busy or unavailable
        volatile status_code_t status;     // Transfer status

} transfer;

/**
 * \internal
 *
 * \brief TWI Master Interrupt Vectors
 *
 * The TWI master interrupt request entry points are conditionally compiled
 * for the TWI interfaces supported by the XMEGA MCU variant.  All of these
 * entry points call a common service function, i2cm_interrupt_handler(),
 * to handle bus events.  This handler uses the bus interface and message
 * parameters specified in the global \c transfer structure.
 */
static void i2cm_interrupt_handler(void);

ISR(TWI0_TWIM_vect)
{
        i2cm_interrupt_handler();
}

/**
 * \internal
 *
 * \brief Test for an idle bus state.
 *
 *  Software can determine the TWI master bus state (unknown, idle, owner, or
 *  busy) by reading the bus master status register:
 *
 *          TWI_MASTER_BUSSTATE_UNKNOWN_gc Bus state is unknown.
 *          TWI_MASTER_BUSSTATE_IDLE_gc    Bus state is idle.
```

```
 *          TWI_MASTER_BUSSTATE_OWNER_gc   Bus state is owned by the master.
 *          TWI_MASTER_BUSSTATE_BUSY_gc    Bus state is busy.
 *
 * \param   twi     Base address of the TWI (i.e. &TWI_t).
 *
 * \retval  true    The bus is currently idle.
 * \retval  false   The bus is currently busy.
 */
static inline bool i2cm_idle()
{

        return ((TWI0_MSTATUS & TWI_BUSSTATE_gm) == TWI_BUSSTATE_IDLE_gc);
}

/**
 * \internal
 *
 * \brief Get exclusive access to global TWI resources.
 *
 * Wait to acquire bus hardware interface and ISR variables.
 *
 * \param no_wait  Set \c true to return instead of doing busy-wait (spin-lock).
 *
 * \return STATUS_OK if the bus is acquired, else STATUS_BUSY.
 */
static inline status_code_t i2cm_acquire(bool no_wait)
{
        while (transfer.locked) {

                if (no_wait) {
                        return STATUS_BUSY;
                }
        }

        cpu_irq_disable();

        transfer.locked = true;
        transfer.status = OPERATION_IN_PROGRESS;

        cpu_irq_enable();

        return STATUS_OK;
}

/**
 * \internal
 *
 * \brief Release exclusive access to global TWI resources.
 *
 * Release bus hardware interface and ISR variables previously locked by
 * a call to \ref i2cm_acquire().  This function will busy-wait for
 * pending driver operations to complete.
 *
 * \return  status_code_t
 *      - STATUS_OK if the transfer completes
 *      - STATUS_BUSY to indicate an unavailable bus
 *      - STATUS_ERR_IO to indicate a bus transaction error
 *      - STATUS_ERR_NO_MEMORY to indicate buffer errors
 *      - STATUS_ERR_PROTOCOL to indicate an unexpected bus state
 */
static inline status_code_t i2cm_release(void)
{
        /* First wait for the driver event handler to indicate something
         * other than a transfer in-progress, then test the bus interface
         * for an Idle bus state.
         */
        while (OPERATION_IN_PROGRESS == transfer.status)
                ;
```

```
        while (!i2cm_idle()) {
                barrier();
        }

        status_code_t const status = transfer.status;

        transfer.locked = false;

        return status;
}

/**
 * \internal
 *
 * \brief TWI master write interrupt handler.
 *
 *  Handles TWI transactions (master write) and responses to (N)ACK.
 */
static inline void i2cm_write_handler(void)
{
        //TWI_t *const         bus = transfer.bus;
        twi_package_t *const pkg = transfer.pkg;

        /*if (transfer.addr_count < pkg->addr_length) {

                const uint8_t *const data = pkg->addr;
                TWI0_MDATA                = data[transfer.addr_count++];

        } else*/ if (transfer.data_count < pkg->length) {

                if (transfer.read) {

                        /* Send repeated START condition (Address|R/W=1). */

                        TWI0_MADDR |= 0x01;

                } else {
                        const uint8_t *const data = pkg->buffer;
                        TWI0_MDATA                = data[transfer.data_count++];
                }

        } else {

                /* Send STOP condition to complete the transaction. */

                TWI0_MCTRLB     = TWI_MCMD_STOP_gc;
                transfer.status = STATUS_OK;
        }
}

/**
 * \internal
 *
 * \brief TWI master read interrupt handler.
 *
 *  This is the master read interrupt handler that takes care of
 *  reading bytes from the TWI slave.
 */
static inline void i2cm_read_handler(void)
{
        //TWI_t *const         bus = transfer.bus;
        twi_package_t *const pkg = transfer.pkg;

        if (transfer.data_count < pkg->length) {

                uint8_t *const data         = pkg->buffer;
                data[transfer.data_count++] = TWI0_MDATA;

                /* If there is more to read, issue ACK and start a byte read.
```

```
                 * Otherwise, issue NACK and STOP to complete the transaction.
                 */
                if (transfer.data_count < pkg->length) {

                        TWI0_MCTRLB = TWI_MCMD_RECVTRANS_gc;

                } else {

                        TWI0_MCTRLB     = TWI_ACKACT_bm | TWI_MCMD_STOP_gc;
                        transfer.status = STATUS_OK;
                }

        } else {

                /* Issue STOP and buffer overflow condition. */
                TWI0_MCTRLB     = TWI_MCMD_STOP_gc;
                transfer.status = STATUS_ERR_NO_MEMORY;
        }
}

/**
 * \internal
 *
 * \brief Common TWI master interrupt service routine.
 *
 *  Check current status and calls the appropriate handler.
 */
static void i2cm_interrupt_handler(void)
{
        uint8_t const master_status = TWI0_MSTATUS;

        if (master_status & TWI_ARBLOST_bm) {/*Arbitration lost*/

                TWI0_MSTATUS = master_status | TWI_ARBLOST_bm;
                TWI0_MCTRLB  = TWI_MCMD_STOP_gc;
                transfer.status       = STATUS_BUSY;

        } else if ((master_status & TWI_BUSERR_bm) || (master_status & TWI_RXACK_bm)) {/*Bus error
or Reciever didn't acknowledge*/

                TWI0_MCTRLB = TWI_MCMD_STOP_gc;
                transfer.status      = STATUS_ERR_IO;

        } else if (master_status & TWI_WIF_bm) {/*Write interrupt flag*/

                i2cm_write_handler();

        } else if (master_status & TWI_RIF_bm) {/*Read interrupt flag*/

                i2cm_read_handler();

        } else {

                transfer.status = STATUS_ERR_PROTOCOL;
        }
}

/**
 * \brief Initialize the twi master module
 *
 */
void twi_master_init(void)
{
        i2c_init();
        TWI0.MCTRLA |= (TWI_RIEN_bm | TWI_WIEN_bm);

        transfer.locked = false;
        transfer.status = STATUS_OK;
}
```

```
/**
 * \brief Perform a TWI master write or read transfer.
 *
 * This function is a TWI Master write or read transaction.
 *
 * \param twi       Base address of the TWI (i.e. &TWI_t).
 * \param package   Package information and data
 *                  (see \ref twi_package_t)
 * \param read      Selects the transfer direction
 *
 * \return  status_code_t
 *      - STATUS_OK if the transfer completes
 *      - STATUS_BUSY to indicate an unavailable bus
 *      - STATUS_ERR_IO to indicate a bus transaction error
 *      - STATUS_ERR_NO_MEMORY to indicate buffer errors
 *      - STATUS_ERR_PROTOCOL to indicate an unexpected bus state
 *      - STATUS_ERR_INVALID_ARG to indicate invalid arguments.
 */
status_code_t i2c_master_transfer(/*TWI_t *twi,*/ const twi_package_t *package, bool read)
{
        /* Do a sanity check on the arguments. */

        if (/*(twi == NULL) ||*/ (package == NULL)) {
                return STATUS_ERR_INVALID_ARG;
        }

        /* Initiate a transaction when the bus is ready. */

        status_code_t status = i2cm_acquire(package->no_wait);

        if (STATUS_OK == status) {
                //transfer.bus        = (TWI_t *)twi;
                transfer.pkg        = (twi_package_t *)package;
                transfer.addr_count = 0;
                transfer.data_count = 0;
                transfer.read       = read;

                uint8_t chip = (package->chip) << 1;
                /*
                 * If it reads puts the R/W flag of the TWI0.MADDR up
                 */
                if (/*package->addr_length || (*/false == read/*)*/) {
                        TWI0_MADDR = chip;
                } else if (read) {
                        TWI0_MADDR = chip | 0x01;
                }

                status = i2cm_release();
        }

        return status;
}
```

I^2C

Header File:

```
/*
 * File:   i2c.h
 * Author: DianaM
 *
 * Created on 9 de enero de 2020, 03:50 PM
 */

#ifndef I2C_H
#define I2C_H

#ifdef  __cplusplus
```

```
extern "C" {
#endif

#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#define TWI0_BAUD(F_SCL, T_RISE)
\
        ((((((float)5000000.0 / (float)F_SCL)) - 10 - ((float)5000000.0 * T_RISE / 1000000))) / 2)

void i2c_init(void);

#ifdef  __cplusplus
}
#endif

#endif  /* I2C_H */
```

Source File:

```
#include "i2c.h"
#include <stddef.h>
#include <stdbool.h>
typedef struct {
        //! TWI chip address to communicate with.
        char chip;
        //! TWI address/commands to issue to the other chip (node).
        uint8_t addr[3];
        //! Length of the TWI data address segment (1-3 bytes).
        int addr_length;
        //! Where to find the data to be written.
        void *buffer;
        //! How many bytes do we want to write.
        unsigned int length;
        //! Whether to wait if bus is busy (false) or return immediately (true)
        bool no_wait;
} twi_package_t;

static struct {
        TWI_t *                 bus;        // Bus register interface
        twi_package_t *         pkg;        // Bus message descriptor
        int                     addr_count; // Bus transfer address data counter
        unsigned int            data_count; // Bus transfer payload data counter
        bool                    read;       // Bus transfer direction
        bool                    locked;     // Bus busy or unavailable
        //volatile status_code_t status;     // Transfer status

} transfer;
void i2c_init(void)
{
        TWI0.MBAUD = (uint8_t)TWI0_BAUD(100000, 0); /* set MBAUD register to 100kHz*/

        TWI0.MCTRLA = 1 << TWI_ENABLE_bp        /* Enable TWI Master: enabled */
        | 0 << TWI_QCEN_bp        /* Quick Command Enable: disabled */
        | 0 << TWI_RIEN_bp        /* Read Interrupt Enable: disabled */
        | 0 << TWI_SMEN_bp        /* Smart Mode Enable: disabled */
        | TWI_TIMEOUT_DISABLED_gc /* Bus Timeout Disabled */
        | 0 << TWI_WIEN_bp;       /* Write Interrupt Enable: disabled */

        return 0;
}
```

4.2 Appendix C: Database Source Code

In this appendix you will find the database was built in a MySQL server hosted by wwww.nwksec.com as described in the subsection 2.2.4.1 based on the system design in subsections 2.2.3.4, 2.2.3.5 and 2.2.3.6.

Here you will find the source code that builds the database structure, in the following order:

- Procedures
  - Select Alarm
  - Select Gesture
  - Select Recording
  - Select User
- Tables
  - Alarms
  - Gestures
  - Recording
  - User

```
-- phpMyAdmin SQL Dump
-- version 4.0.10.20
-- https://www.phpmyadmin.net
--
-- Servidor: localhost
-- Tiempo de generación: 19-02-2020 a las 13:01:56
-- Versión del servidor: 5.1.73-community
-- Versión de PHP: 5.6.40

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS
*/;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;


--
-- Data Base: `pf000219_Tesis`
--


DELIMITER $$
--
-- Procedures
--
CREATE DEFINER=`pf000219`@`localhost` PROCEDURE `SeleDatAlarms`()
SELECT * FROM ALARMS$$


CREATE DEFINER=`pf000219`@`localhost` PROCEDURE
`SeleDatGestures`()
SELECT * FROM GESTURES$$
```

```
CREATE DEFINER=`pf000219`@`localhost` PROCEDURE
`SeleDatRecording`()
SELECT * FROM RECORDING$$


CREATE DEFINER=`pf000219`@`localhost` PROCEDURE `SeleDatUser`()
SELECT * FROM USER$$


DELIMITER ;


-- --------------------------------------------------------


--
-- Structure for the table `ALARMS`
--


CREATE TABLE IF NOT EXISTS `ALARMS` (
  `Username` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `alarmNum` int(11) NOT NULL,
  `gesName` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `actSt` tinyint(1) NOT NULL,
  `initTime` time NOT NULL,
  `mon` tinyint(1) NOT NULL,
  `tu` tinyint(1) NOT NULL,
  `wed` tinyint(1) NOT NULL,
  `th` tinyint(1) NOT NULL,
  `fri` tinyint(1) NOT NULL,
  `sat` tinyint(1) NOT NULL,
  `sun` tinyint(1) NOT NULL,
  `Note` varchar(200) CHARACTER SET utf8 COLLATE utf8_spanish_ci
DEFAULT NULL,
  UNIQUE KEY `UserAlarm` (`Username`,`alarmNum`)
```

```
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------


--
-- Structure for the table `GESTURES`
--


CREATE TABLE IF NOT EXISTS `GESTURES` (
  `Username` varchar(100) NOT NULL,
  `gesName` varchar(100) NOT NULL,
  `dur` int(11) NOT NULL,
  UNIQUE KEY `UserGest` (`Username`,`gesName`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;


-- --------------------------------------------------------


--
-- Structure for the table `RECORDING`
--


CREATE TABLE IF NOT EXISTS `RECORDING` (
  `Username` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `gesName` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `recordingID` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `recData` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `Type` tinyint(1) NOT NULL,
  UNIQUE KEY `RecId` (`recordingID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
-- --------------------------------------------------------


--
-- Structure for the table `USER`
--


CREATE TABLE IF NOT EXISTS `USER` (
  `Username` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `Passord` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `Name` varchar(100) CHARACTER SET utf8 COLLATE utf8_spanish_ci
NOT NULL,
  `Surname` varchar(100) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `Gender` enum('F','M','O') CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL,
  `Birthdate` date NOT NULL,
  `Type` tinyint(1) NOT NULL,
  UNIQUE KEY `Username` (`Username`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

4.3 Appendix D: Pre-processing Source Code

In this appendix, you will find the source code pre-processing that would be needed to be done to extract the signal of each of the axis from the IMU. This code was written with MATLAB.

For importing the data to MATLAB it should be in a .txt file, where the data has the format given in the acquisition. As reported in the appendix 4.1 each acquisition stats sending "START_ACQ", and each sensor sample sent form the device is labeled as follows:

- "acalX": for the accelerometer calibration value in the x axis.
- "acalY": for the accelerometer calibration value in the y axis.
- "acalZ": for the accelerometer calibration value in the z axis.
- "aX": for the accelerometer value in the x axis.
- "aY": for the accelerometer value in the y axis.
- "aZ": for the accelerometer value in the z axis.
- "gcalX": for the gyroscope calibration value in the x axis.
- "gcalY": for the gyroscope calibration value in the y axis.
- "gcalZ": for the gyroscope calibration value in the z axis.
- "gX": for the gyroscope value in the x axis.
- "gY": for the gyroscope value in the y axis.
- "gZ": for the gyroscope value in the z axis.

and ended with a ".". This is reflected in the way the data is extracted in this code.

Every axis is plotted taking into account the calibration and the data vector is built considering that the sampling frequency was 100Hz.

```
%Politecnico di Milano
%Thesis work
%Pre-processing
%Author: GONZÁLEZ RAMÍREZ Diana Maritza

close all
clear all
clc
sampling_frequency=100;
timescale=1/sampling_frequency;%100Hz sampling frequency


filename='acquisition.txt';
text = readtable(filename);

%Counts the number of acquisitions in the file
acq_num=count(text.Properties.VariableDescriptions,"START_ACQ
")
textAfter=extractAfter(text.Properties.VariableDescriptions,"
START_ACQ");
%String array that will contain the different acquisitions
string t(acq_num);
%The loop separates the acquisitions in a sting array
for i = 1: acq_num
numAfter=count(textAfter,"START_ACQ");
if(numAfter==0)
t(i)=textAfter   ;
else
t(i)=extractBefore(textAfter,"START_ACQ");
textAfter=extractAfter(textAfter,"START_ACQ");
end
end

maxsamples=0;%Determines the max number of samples of all
acquisitons
for i=1:acq_num
  if(count(t(i),"aX")>maxsamples)
      maxsamples=count(t(i),"aX");
  end
end
time=0:timescale:timescale*maxsamples %time vector
string accXASCII;
string accYASCII;
string accZASCII;
string gyrXASCII;
string gyrYASCII;
string gyrZASCII;
```

```
string accOffXASCII;
string accOffYASCII;
string accOffZASCII;
string gyrOffXASCII;
string gyrOffYASCII;
string gyrOffZASCII;
%Creates the vectors that contain the different signals
for i=1:acq_num
    temptAfter=t(i);
    for k=1:maxsamples

        temptAfter=extractAfter(temptAfter,"acalX");
        accOffXASCII(i,k)=extractBefore(temptAfter,".");
        temptAfter=extractAfter(temptAfter,"aX");
        accXASCII(i,k)=extractBefore(temptAfter,".");

        temptAfter=extractAfter(temptAfter,"acalY");
        accOffYASCII(i,k)=extractBefore(temptAfter,".");
        temptAfter=extractAfter(temptAfter,"aY");
        accYASCII(i,k)=extractBefore(temptAfter,".");

        temptAfter=extractAfter(temptAfter,"acalZ");
        accOffZASCII(i,k)=extractBefore(temptAfter,".");
        temptAfter=extractAfter(temptAfter,"aZ");
        accZASCII(i,k)=extractBefore(temptAfter,".");



        temptAfter=extractAfter(temptAfter,"gcalX");
        gyrOffXASCII(i,k)=extractBefore(temptAfter,".");
        temptAfter=extractAfter(temptAfter,"gX");
        gyrXASCII(i,k)=extractBefore(temptAfter,".");

        temptAfter=extractAfter(temptAfter,"gcalY");
        gyrOffYASCII(i,k)=extractBefore(temptAfter,".");
        temptAfter=extractAfter(temptAfter,"gY");
        gyrYASCII(i,k)=extractBefore(temptAfter,".");

        temptAfter=extractAfter(temptAfter,"gcalZ");
        gyrOffZASCII(i,k)=extractBefore(temptAfter,".");
        temptAfter=extractAfter(temptAfter,"gZ");
        gyrZASCII(i,k)=extractBefore(temptAfter,".");


    end

end
```

```
%convert hexa to decimal and stores the decimal value of the
signals
for i=1:acq_num
    accX(i,:)=hex2dec(accXASCII(i,:));
    accY(i,:)=hex2dec(accYASCII(i,:));
    accZ(i,:)=hex2dec(accZASCII(i,:));
    gyrX(i,:)=hex2dec(gyrXASCII(i,:));
    gyrY(i,:)=hex2dec(gyrYASCII(i,:));
    gyrZ(i,:)=hex2dec(gyrZASCII(i,:));

    accOffX(i,:)=hex2dec(accOffXASCII(i,:));
    accOffY(i,:)=hex2dec(accOffYASCII(i,:));
    accOffZ(i,:)=hex2dec(accOffZASCII(i,:));
    gyrOffX(i,:)=hex2dec(gyrOffXASCII(i,:));
    gyrOffY(i,:)=hex2dec(gyrOffYASCII(i,:));
    gyrOffZ(i,:)=hex2dec(gyrOffZASCII(i,:));

    aX(i,:)=accX(i,:)-accOffX(i,:);
    aY(i,:)=accY(i,:)-accOffY(i,:);
    aZ(i,:)=accZ(i,:)-accOffZ(i,:);
    gX(i,:)=gyrX(i,:)-gyrOffX(i,:);
    gY(i,:)=gyrY(i,:)-gyrOffY(i,:);
    gZ(i,:)=gyrZ(i,:)-gyrOffZ(i,:);
end
%plots the different signals in time
for i=1:acq_num
    figure
    plot(time(1:length(aX(i,:))),aX(i,:));
    title(['accelerometer X, acquisition:',num2str(i)]);
    figure
    plot(time(1:length(aY(i,:))),aY(i,:));
    title(['accelerometer Y, acquisition:',num2str(i)]);
    figure
    plot(time(1:length(aZ(i,:))),aZ(i,:));
    title(['accelerometer Z, acquisition:',num2str(i)]);
    figure
    plot(time(1:length(gX(i,:))),gX(i,:));
    title(['gyroscope X, acquisition:',num2str(i)]);
    figure
    plot(time(1:length(gY(i,:))),gY(i,:));
    title(['gyroscope Y, acquisition:',num2str(i)]);
    figure
    plot(time(1:length(gZ(i,:))),gZ(i,:));
    title(['gyroscope Z, acquisition:',num2str(i)]);
end
```

5. References

Amft, O. (2018). How wearable computing is shaping digital health. *IEEE Pervasive Computing*.

Anliker, U., Ward, J. A., Lukowicz, P., Tröster, G., Dolveck, F., Baer, M., ... Vuskovic, M. (2004). Amon: A wearable multiparameter medical monitoring and alert system. *IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 8, NO. 4*.

Bonato, P. (2010). Wearable sensors and systems. *IEEE ENGINEERING IN MEDICINE AND BIOLOGY MAGAZINE*.

Costin, H., Rotariu, C., Morancea, O., Aandruseac, G., Cehan, V., Felea, V., ... Costin, C. (2008). Complex telemonitoring of patients and elderly people for telemedical and homecare services. *1st WSEAS International Conference on Biomedical Electronics and Biomedical Informatics (BEBI '08)*.

Espina, J., Falck, T., Panousopoulou, A., Schmitt, L., Mülhens, O., & Yang, G.-Z. (2014). *Body sensor networks, chapter 5, network topologies, communication protocols and standards* (G.-Z. Yang, Ed.). Springer.

Feldmann, L. M. (2017). *Highway to health.* Retrieved from `https://www.volkswagenag.com/en/news/stories/2017/01/highway-to-health.html`

Gridling, G., & Weiss, B. (2007). *Introduction to microcontrollers, chapter 2, microcontroller components.* Vienna University of Technology.

Heydon, R. (2013). *Bluetooth Low Energy: The Developer's Handbook, Chapter 1, What Is Bluetooth Low Energy?* (B. Goodwin, J. Fuller, E. Ryan, & B. Russell, Eds.). One Lake Street, Upper Saddle River, New Jersey 07458: Pearson Education, Inc.

InvenSense (Ed.). (2016). *ICM-20600* [Datasheet].

Jiménez, M., Palomera, R., & Couvertier, I. (2014). *Introduction to embedded systems, chapter 1.* New York: Springer.

Kok, M., Hol, J. D., & Schön, T. B. (2017). *Using inertial sensors for position and orientation estimation, chapter 2, interial sensors.* Netherlands: Delft Center for Systems and Control, Delft University of Technology.

Lai, X., Liu, Q., Wei, X., Wang, W., Zhou, G., & Han, G. (2013). A survey of body sensor networks. *Sensors (Basel, Switzerland).*

Lamkin, P. (2018). *Smart wearables market to double by 2022: $27 billion industry forecast.* Retrieved from `https://www.forbes.com/sites/paullamkin/2018/10/23/smart-wearables-market-to-double-by-2022-27-billion-industry-forecast/#576b19592656`

LeCroy (Ed.). (2011). *Computation of Effective Number of Bits, Signal to Noise Ratio, & Signal to Noise & Distortion Ratio Using FFT* [Application Note].

Luinge, H. J. (2002). *Inertial sensing of human movement* (Unpublished doctoral dissertation). University of Twente.

Microchip (Ed.). (2016-2017a). *RN4870/71 Bluetooth® 4.2 Low Energy Module* [Datasheet].

Microchip (Ed.). (2016-2017b). *RN4870/71 Bluetooth® Low Energy Module User's Guide* [User's guide].

Microchip (Ed.). (2017). *AVR315:Using the TWI Module as I2C Master* [Application Note].

Microchip (Ed.). (2018). *Bootloader for tinyAVR® 0- and 1-series, and megaAVR® 0-series* [Application Note].

Microchip (Ed.). (2019a). *ATtiny1617/3217 tiny AVR® 1-series* [Datasheet].

Microchip (Ed.). (2019b). *Getting Started with USART* [Application Note].

Movassaghi, S., Abolhasan, M., Lipman, J., Smith, D., & Jamalipour, A. (2014). Wireless body area networks: A survey. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS.*

Negra, R., Jemili, I., & Belghith, A. (2016). Wireless body area networks: Applications and technologies. *Procedia Computer Science.*

Normansell, R., Kew, K. M., & Stovold, E. (2017). Interventions to improve adherence to inhaled steroids for asthma (review). *Cochrane Database of Systematic Reviews.*

Pantelopoulos, A., & Bourbakis, N. G. (2010). A survey on wearable sensor-based systems for health monitoring and prognosis. *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*.

Phillips (Ed.). (2000). *THE I2C-BUS SPECIFICATION version 2.1.*

Phillips (Ed.). (2003). *AN10216-01 I2C Manual* [Application Note].

Sabaté, E. (2003). *WHO Adherence to Long-Term Therapies,Section I, Chapters 1-3* . Geneva,Switzerland: World Health Organization.

Shnayder, V., rong Chen, B., Lorincz, K., Fulford-Jones, T. R. F., & Welsh, M. (2005). Sensor networks for medical care. *Harvard Computer Science Group Technical Report TR-08-05.*.

Townsend, K., Cufí, C., Akiba, & Davidson, R. (2014). *Getting Started with Bluetooth Low Energy, Chapters 1-2.* 1005 Gravenstein Highway North Sebastopol, CA 95472: O'Reilly Media, Inc.

X, M., & C.Menon, J. (2018). *Wearable technology in medicine and health care, chapter 7, wearable technologies and force myography for healtcare.* Metro Vancouver, BC, Canada: Menrva Research Group, Schools of Mechatronic Systems and Engineering Science, Simon Fraser University.