

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING
Master Degree in Automation Engineering

**BLE EMBEDDED SOLUTION
FOR GAIT ANALYSIS
ON SUBJECTS AFFECTED BY
MULTIPLE SCLEROSIS**

Relatore:
Chiar.mo Prof.
LUIGI DI STEFANO

Candidato:
ALESSANDRO SFRAPPINI

Correlatore:
Ing.
NIDHAL LOUHICHI

Session I
Academic Year 2021/2022

Contents

I	Low Power Communication	8
1	Bluetooth Low Energy	9
1.1	Overview	9
1.2	BLE Stack	10
1.2.1	Physical Layer	11
1.2.2	Link Layer	12
1.2.3	Host Controller Interface	22
1.2.4	Logical Link Control and Adaptation Protocol	23
1.2.5	Security Manager Protocol	24
1.2.6	Attribute Protocol	24
1.2.7	Generic Attribute Profile	24
1.2.8	Generic Access Profile	27
2	Software Implementation	30
2.1	Tools	30
2.2	Implemented Solution	32
2.2.1	Project Zero	32
2.2.2	Battery Service	34
2.2.3	Imu Service	34
2.3	Current Consumption Optimization	35
2.3.1	TX Power Level	35
2.3.2	Advertisement Interval	35
2.3.3	Advertising Packet	37
2.4	Sensor Controller	38

2.4.1	Battery Task	39
2.4.2	Imu Task	40
2.5	Experimental Results	45
II	Gait Analysis	46
3	Human Activity Recognition	49
3.1	Machine Learning	49
3.2	Neural Networks	51
3.3	Recurrent Neural Networks	54
3.3.1	Long Short-Term Memory	55
3.4	Experimental Results	57
3.4.1	UMAFall	57
3.4.2	Preprocessing	58
3.4.3	Training	59
3.4.4	Prediction	63
4	Walking Detection	65
4.1	Overview	65
4.2	Background	65
4.3	Gait Measures	66
4.4	Experimental Results	68
4.4.1	Turns and Pauses Detection	69
4.4.2	Gait Analysis	69

List of Figures

1.1	BLE protocol stack	11
1.2	BLE channel arrangement	12
1.3	Frequency Shift Keying	13
1.4	Link Layer State Machine	14
1.5	Link Layer Packet Format	14
1.6	Advertising Channel PDU	16
1.7	Advertising Channel PDU Header	16
1.8	Data Channel PDU	16
1.9	Data Channel PDU Header	17
1.10	Frequency Hopping in three different BLE connections	18
1.11	Adaptive Frequency Hopping in presence of three interfering Wi-Fi networks	19
1.12	Connection Event and Interval	20
1.13	Slave Latency	21
1.14	L2CAP Architectural Blocks	23
1.15	Logical Attribute Representation	25
1.16	GATT Client and Server	25
1.17	Profiles and Services	26
1.18	Connection Procedures and Peripheral Connectability	28
1.19	BLE Heart Monitor Application	29
2.1	CC2650 Block Diagram	31
2.2	Tx Power and Current Consumption	36
2.3	I/O Mapping for Sensor Controller	39
2.4	SPI Single Master to Single Slave	40
2.5	SPI Read Protocol	41

2.6	SPI Write Protocol	41
2.7	Activity/Inactivity Detection	44
2.8	Tilt Detection	45
3.1	Instance-based learning vs. Model-based learning	51
3.2	Three Layer Neural Network	52
3.3	General Neuron Model	53
3.4	Activation Functions	53
3.5	LSTM Unit structure	55
3.6	Activity Recognition Network Model	60
3.7	Loss and Accuracy plots	62
3.8	Confusion Matrix	62
3.9	Test Scores on UMAFall dataset	63
4.1	Gait Measures: Rhythm Domain	67
4.2	Gait Analysis	70

Introduction

Motivations

Multiple sclerosis (MS), also known as encephalomyelitis disseminata, is a demyelinating disease in which the insulating covers of nerve cells in the brain and spinal cord are damaged. The damage disrupts the ability of parts of the nervous system to transmit signals. The initial clinical course is variable, but the majority of patients either present with or transition into a progressive course, characterised by the gradual accumulation of disability independent of clinical relapses, which usually significantly affect their ability to walk [35]. It is estimated that more than 1.3 million people have progressive MS worldwide. Walking related disability has a significant impact on the quality of life and is rated by people with MS as one of their worst symptoms. Studies with inertial measurement units (IMUs) have identified a disease related reduction of gait quality, such as decreased stride length or gait speed and have helped to identify characteristic motor patterns occurring prior to freezing of gait. In recent years, the development of small and light-weight IMUs has facilitated long-term studies of gait outside of the clinical environment. Gait measurements over several days and weeks have allowed to capture a more representative disease status of patients that can for example serve the estimation of their fall risk [33]. The fundamental advantage of free-living gait analysis conditions is the increase of the acquired data volume. Larger amounts of data provide a more representative impression of a patient gait to base better treatment decisions upon.

Main Objective

This work can be divided into two main components.

First, the development of an embedded low power medical device for data collection and Bluetooth Low Energy communication. The main feature that the device must have is to be low power. There are two reasons for the power consumption constraint. The final board will mount a dedicated energy harvester, that will recharge the battery during physical movements. So the first reason for low power is that we want to create an energy autonomous device, because the energy harvester will be able to completely compensate for the power losses. The second lies in the user experience problem. Since we are talking about a device which has to be used by people affected by the MS disease, the device must not be a constant reminder of the fact that they are undergoing medical tests. For this to be possible, apart from the fact that the subjects will interact directly only with their smartphone and with customized plantars to put inside the shoes, the embedded device must work for the longest time possible before a recharge is needed. Having to charge the board every other day can significantly affect the user experience of the customer.

The following part takes into account the problem of gait assessment. The device is designed to assist the doctors in the assessment of the quality of the patient's mobility. Therefore algorithms are studied to extract gait information from the sensor data, according to the environment defined by some standard clinical tests, that will be supported by the app interface. For the purpose of daily monitoring, the software must be also able to detect walking patterns without knowing if the user is taking any test. Therefore machine learning approaches are considered.

This two parts project is described in four main chapters.

In the first chapter, an overview of BLE communication protocol is considered.

In the second chapter, an embedded solution for data collection and Bluetooth communication is shown, with a power consumption optimization procedure.

In the third chapter, a machine learning approach is considered and a neural network is designed to identify walking patterns from sensor data arranged as time sequences. The network is trained with a public dataset, then tested on data collected by the sensors.

In the fourth chapter, walking detection methods are considered to extract gait-related information from collected data. The devised algorithm is tested on sensor data collected by the device.

Contributions

This study has been carried out under the supervision of Nidhal Louichi, CEO of eSteps srls. eSteps was founded in 2020 in order to contrast the increase of motor impairment of the lower limbs. It offers telemonitoring programmes before, during and after hospitalization. These solutions also apply to sports, with state-of-the-art technology, personalised and sustainable solutions centered around the patient/athlete.

eSteps has designed a 3D-printed plantar for clinical tests, with a customized housing for the sensory device. This thesis is the result of the collaboration with eSteps' RD department during a 600 hours internship. During the first month, the Bluetooth Low Energy protocol and the implementation of custom services for the transmission of sensory data to a smartphone app were studied. In the remaining hours, the software implementation was perfected and a gait assessment method was developed.

The work has been balanced between smartworking sessions, during the study phase, and in-presence interactions during the testing phase.

Part I

Low Power Communication

Chapter 1

Bluetooth Low Energy

1.1 Overview

The development of 'short-link' radio technology was initiated in 1989 by Nils Rydbeck, CTO at Ericsson Mobile in Sweden. In the following years, Bluetooth technology was formalized by the Bluetooth Special Interest Group (SIG) in the IEEE 802.15.1 standard. Bluetooth was designed for short range, low power radio communications as an alternative to wire connections.

In 2001, Nokia started the development of a wireless technology compatible with the Bluetooth standard, but with lower costs and lower energy consumption [1]. The new technology was inserted with the name Bluetooth Smart in the 4.0 Bluetooth specification in 2010 [2]. After that, the new standard came to be known with the name of *Bluetooth Low Energy* (BLE).

Bluetooth Low Energy defines a *Wireless Personal Area Network* (WPAN) suitable for applications that require low power consumption, low latency, low cost, and support for high connections numbers of devices on the same network. With the implementation of IPV6 defined by the IETF 6LoWPAN Working Group, the BLE has become a promising wireless candidate for applications in the Internet of Things (IoT) [3].

1.2 BLE Stack

A communication protocol is a set of rules that allows two or more entities to transmit and receive information with one another. The BLE standard is a collection of protocols that describes different aspects of the communication. When implemented in software, a group of protocols is defined as a protocol stack. Bluetooth SIG defines the BLE protocols in an architecture defined as *BLE stack*.

The basic structure of the BLE stack is subdivided into three sections: Controller, Host and Application [2]. Figure 1.1 shows the architecture of the protocol stack. The **Controller** encodes the packet and transmits it as a radio signal; on reception, it decodes the radio signal and reconstructs the packet. The **Host** consists of various protocols and profiles that manage the communication between two or more devices. The **Application** is the highest block of the stack, it represents the direct interface with the user. The **Controller** includes the following layers:

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

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 Protocol (SMP)
- Host Controller Interface (HCI), Host side

In the following sections, all the layers will be described in detail.

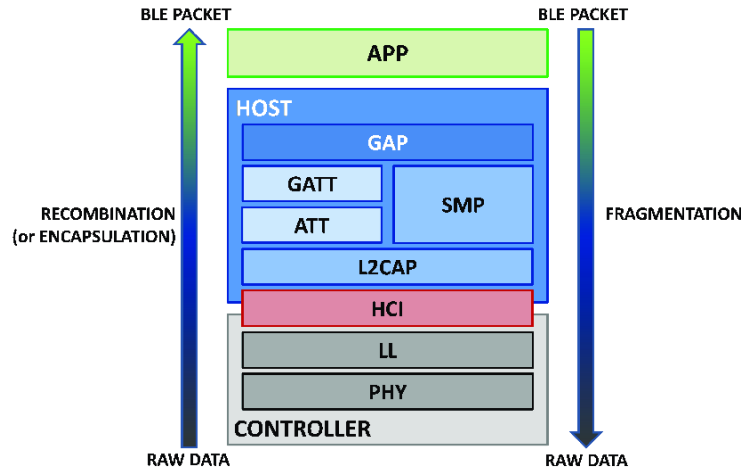


Figure 1.1: BLE protocol stack

1.2.1 Physical Layer

PHY is the bottom layer of the BLE stack and is responsible for transmitting and receiving information over the air via radio waves. It provides services to the Link Layer.

Frequency Bands

The BLE technology is designed to operate in the Industrial, Scientific and Medical (ISM) band included in 2.4-2.5 GHz. In particular, the BLE radio band goes from 2.4000 GHz to 2.4835 GHz. The number of frequency channels has been reduced from 79, as used in the classical Bluetooth protocol, to 40 channels. The channels' frequencies are 2 MHz spaced. The frequency of each channel is defined as follows:

$$f = 2402 + 2kMHz \quad (1.1)$$

Where k is the channel number that ranges from 0 to 39, as shown in Figure 1.2.

Modulation and Data Rate

The BLE radio transmits 1 million symbols per second, which translates to 1 Mbps assuming an encoding of 1 bit per symbol. Information is transmitted using a scheme

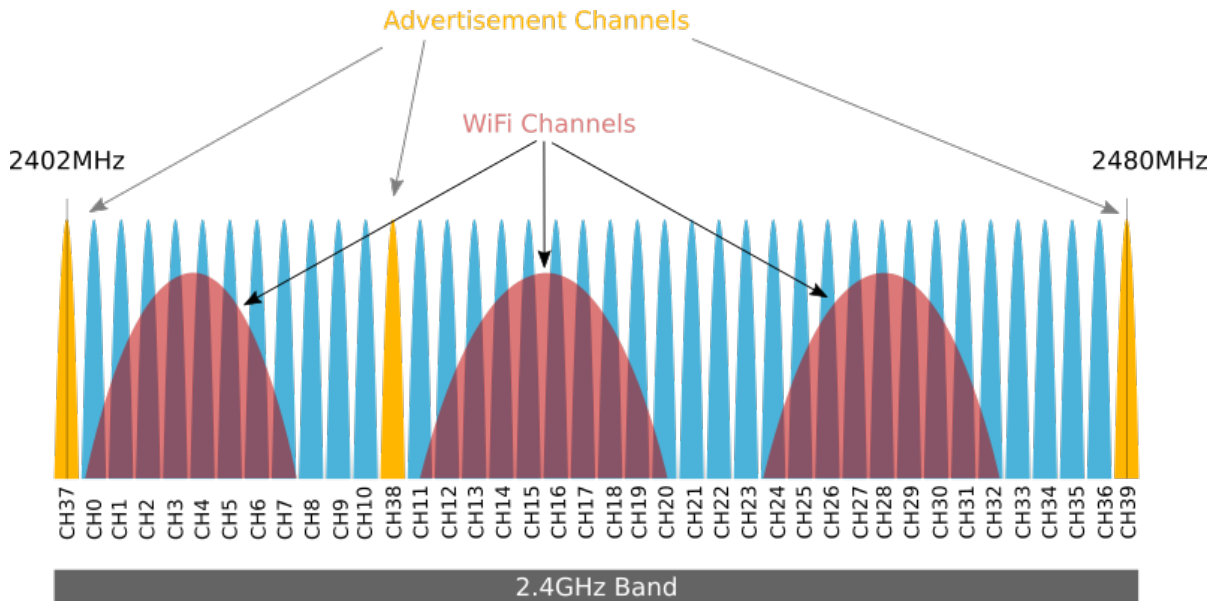


Figure 1.2: BLE channel arrangement

called Gaussian Frequency-Shift Keying (GFSK), which is a refinement of Frequency-Shift Keying (FSK). FSK conveys information by decreasing the carrier frequency for the duration of a 0 symbol and increasing the frequency for the duration of a 1 symbol (Figure 1.3).

In GFSK data pulses are filtered with a Gaussian filter before being applied to alter the carrier frequency. This filter has the advantage of reducing sideband power, reducing interference with neighbouring channels, at the cost of increasing intersymbol interference. BLE modulation is GFSK with a bandwidth bit period product equal to 0.5, with a modulation index between 0.45 and 0.55 [2]. A zero is coded to a negative frequency deviation and a 1 is coded to a positive frequency deviation. The minimum frequency deviation shall be no smaller than $\pm 80\%$ of the frequency deviation with respect to the transmit frequency. In addition, the minimum frequency deviation shall never be less than 185 kHz.

1.2.2 Link Layer

The Link Layer is the part that directly interfaces to the Physical Layer. It defines the type of communications that can be created between BLE devices through the managing

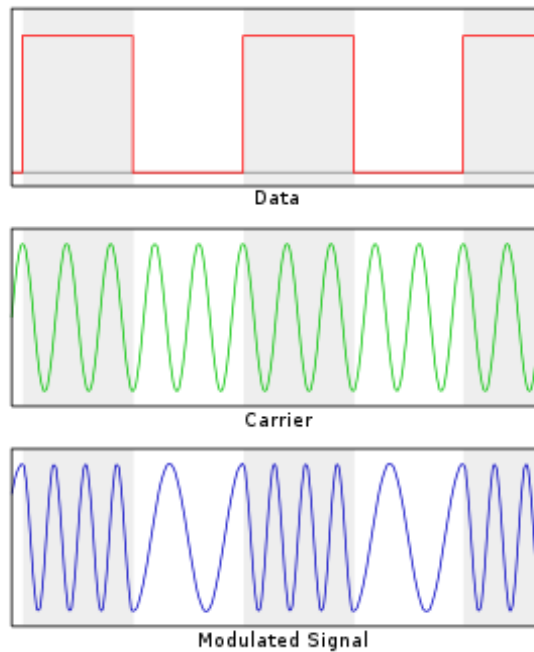


Figure 1.3: Frequency Shift Keying

of the link state of the radio. Link Layer is responsible for advertising, scanning, and creating or maintaining connections. The Link Layer state machine is represented in Figure 1.4.

According to the LL state, devices can be identified by different roles. An **advertiser** transmit data without connecting, while a **scanner** scan for advertisers. An **initiator** is a device that responds to an advertiser with a request to connect. If the advertiser accepts the connection request, both the advertiser and the initiator enter a connected state. The device initiating the connection becomes the **master** and the device accepting the request becomes the **slave**. LL defines two types of channels: **Advertising Channels** and **Data Channels**. Among the 40 BLE channels, channels 1-36 are data channels, while channels 37-38-39 are advertising channels.

Packet Format

LL has only one packet format used both for advertising channel packets and data channel packets. The packet is shown in Figure 1.5 and consists of four fields.

The **Preamble** is used in the receiver to perform frequency synchronization, symbol

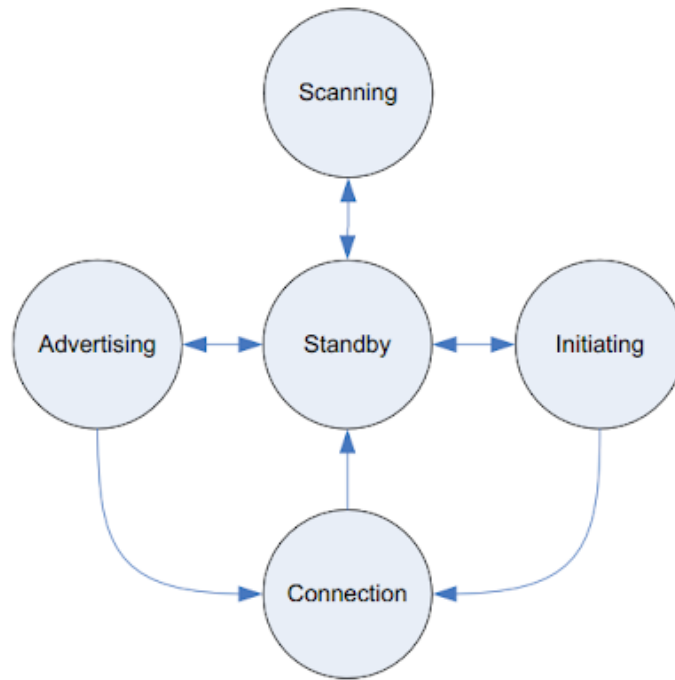


Figure 1.4: Link Layer State Machine

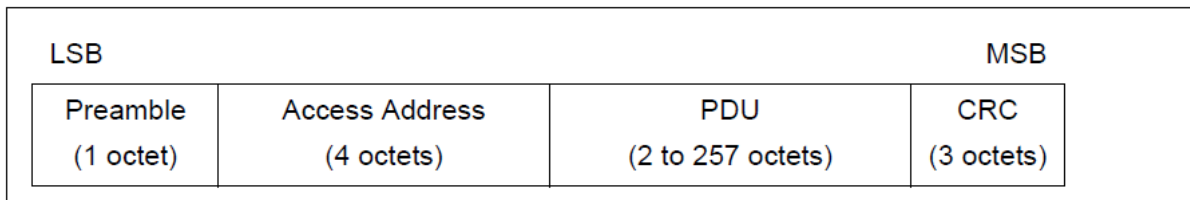


Figure 1.5: Link Layer Packet Format

timing estimation, and Automatic Gain Control (AGC) training.

The **Access Address** contains information about the connection between any two devices. The Access Address is the same for all advertising channel packets (0x8E89BED6). In data channel packets the Access Address is different for each connection. The 32-bit value representing this field is a random value, generated by the device in the initiating state and sent in a connection request. The initiator ensures that the Access Address meets specific requirements [2].

The **Packet Data Unit** contains the data to send to the connected device. When a packet is transmitted in an advertising physical channel, the PDU shall be the Advertising Channel PDU. When a packet is transmitted in a data physical channel, the PDU shall be the Data Channel PDU.

The **Cyclic Redundancy Check** is used to detect accidental changes in raw data. This value is calculated on the PDU field as the remainder of a polynomial division. The polynomial has the form $x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$. At the receiver side, the calculation is repeated and, if the two values don't match, a corrective action can be taken against data corruption.

The **Advertising Channel PDU** has a 16-bit header and a variable size payload, as shown in Figure 1.6. Figure 1.7 shows the header structure. The header fields can be described as follows:

- **PDU Type**: it defines the type of advertising event (connectable/non connectable etc.)
- **TxAdd/RxAdd**: it specifies whether the advertiser/initiator address is public or random. According to the PDU type, they can either be defined or Reserved for Future Use (RFU).
- **Length**: it indicates the payload field length in octets. The valid range is 6 to 37 octets.

The **Data Channel PDU** has a 16-bit header and a variable size payload, and may include a Message Integrity Check (MIC) field, as shown in Figure 1.8. Figure 1.9 shows the header structure. The header fields can be described as follows:

- **LLID**: it indicates whether the packet is a Data PDU or a Control PDU

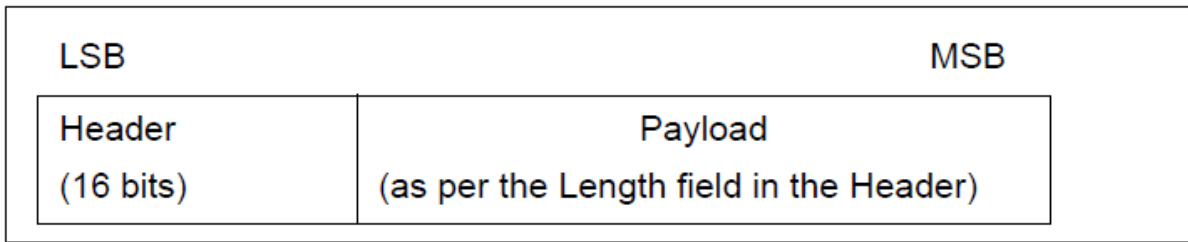


Figure 1.6: Advertising Channel PDU

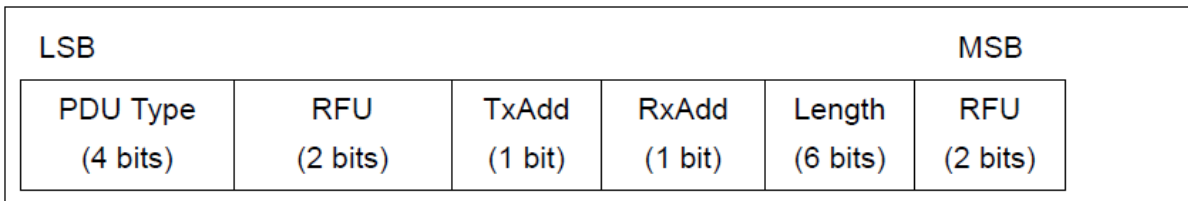


Figure 1.7: Advertising Channel PDU Header

- **NESN**: next expected sequence number
- **SN**: sequence number
- **MD**: more data
- **Length**: it represents the size, in octets, of the Payload and MIC, if included. Payload field is less than or equal to 251 octets in length, while the MIC is 4 octets long.

Advertising Channels

Advertising channels are reserved for sending and receiving short messages broadcast and announcing the presence of the device. They allow the advertiser to connect to

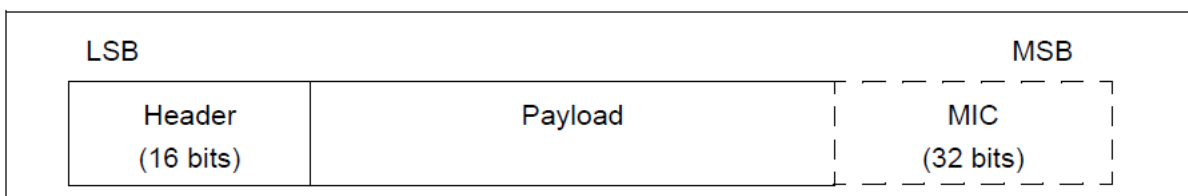


Figure 1.8: Data Channel PDU

Header					
LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	RFU (3 bits)	Length (8 bits)

Figure 1.9: Data Channel PDU Header

another device or to simply be discoverable, meaning that other device in the area can acknowledge their presence. These channels are strategically located at frequencies 2402 MHz, 2426 MHz and 2480 MHz, resulting in the reduction of the coexistence interference with other wireless networks operating in the ISM band, like Wi-Fi, Classic Bluetooth, Microwaves and others [2]. Also, if any single advertising channel is blocked, the other channels are likely to be free since they are separated by quite a few MHz of bandwidth. During advertisement, a BLE device transmits the same packet on the 3 advertising channels, one after the other. A device scanning for advertisers will listen to those channels for the advertising packets.

Data Channels

Data channels are used to transmit application data during a connection. The 37 available data channels are not strategically located, only 9 are free from interferences from other wireless networks. The main problems with interfering communications resides in collisions. A collision occur when two or more devices transmit data on the same radio channel in overlapping time periods. To address the collision problem in data channels, Bluetooth technology applies a form of frequency-hopping spread spectrum (FHSS) called Adaptive Frequency Hopping (AFH). A standard frequency-hopping algorithm is used to cycle through the 37 data channels:

$$f_{n+1} = (f_n + hop) \bmod 37 \quad (1.2)$$

Where f_{n+1} is the frequency to use on the next connection event and hop is a value that can range from 5 to 16 and is set when the connection is created. Figure 1.10 depicts three active BLE connections, showing the frequency hopping sequence.

Adaptive Frequency Hopping is used by the Link Layer to remap a given packet from a known bad channel to a known good channel so that interference is reduced. Over time,

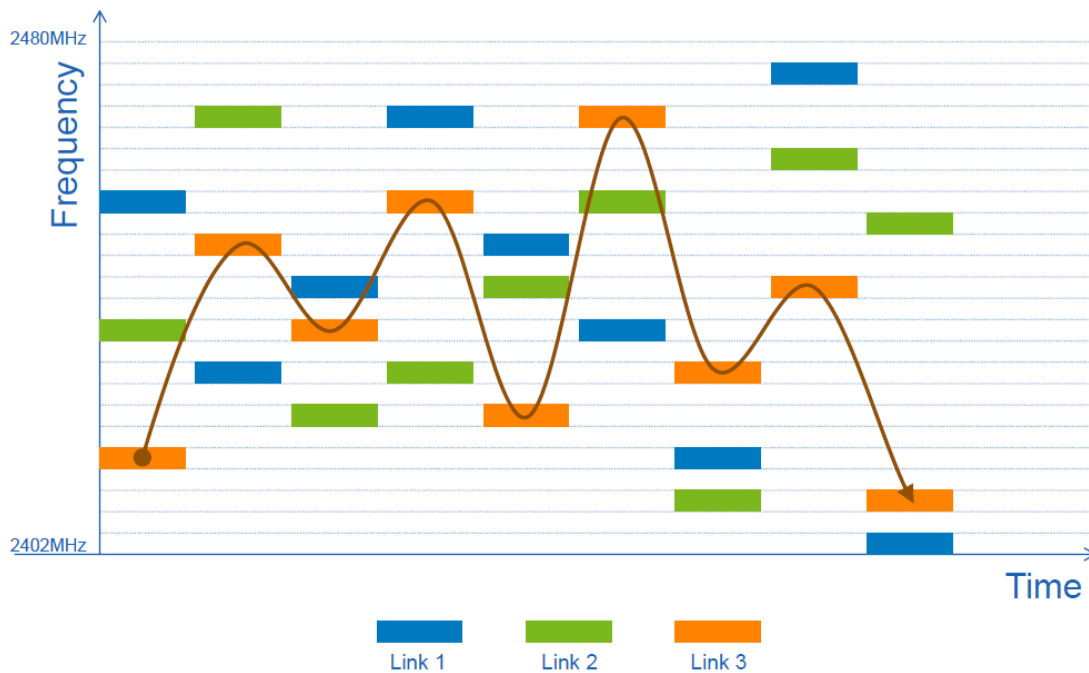


Figure 1.10: Frequency Hopping in three different BLE connections

the list of reliable channels may change, as other wireless communication devices in the environment come and go. The primary device in a connection maintains a *channel map* which classifies each channel that is working well as *used* or *unused*. The channel map is shared with the second device using a Link Layer procedure so that they each have the same information about which channels will be used and which will be avoided. Figure 1.11, for example, that a BLE device is in the same area as several Wi-Fi networks on channels 1, 6, and 11. The BLE device would mark channels 0-8, 11-20, and 24-32 as bad channels. This means that when the two devices are communicating, they would cycle through the channels and remap these to a set of good channels.

Control Procedures

LL defines seven control procedures:

- **Connection Update Procedure:** update the connection intervals
- **Channel Map Update Procedure:** update the adaptive frequency hopping map
- **Encryption Start Procedure:** start encryption using a Long Term Key

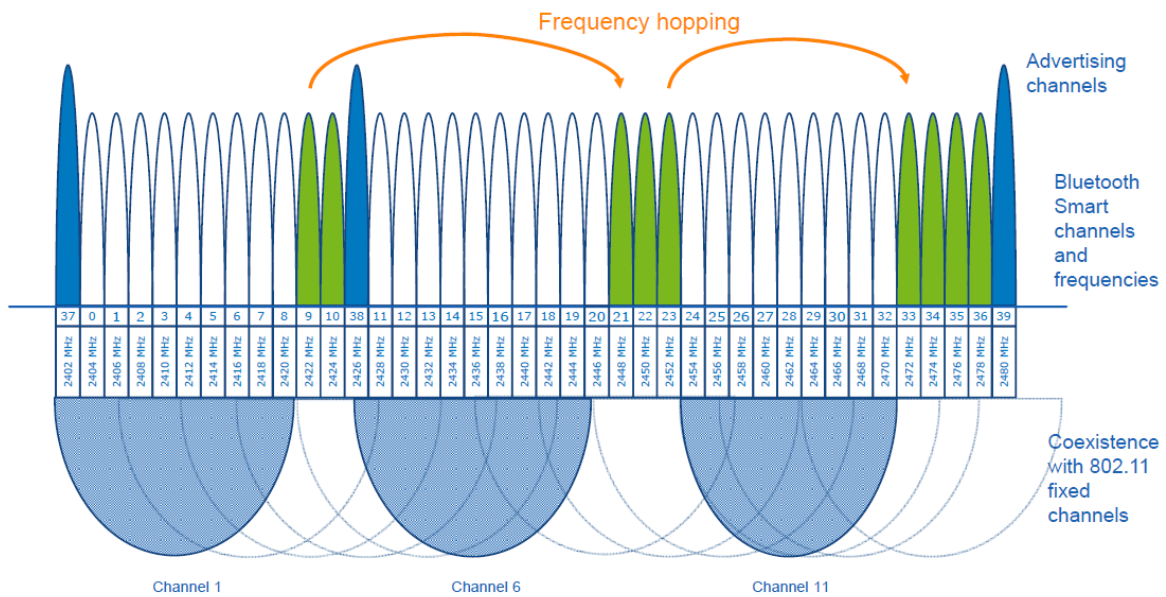


Figure 1.11: Adaptive Frequency Hopping in presence of three interfering Wi-Fi networks

- **Encryption Pause Procedure:** pause encryption to change Long Term Key
- **Feature Exchange Procedure:** exchange the current supported feature set
- **Version Exchange Procedure:** exchange the current version information
- **Termination Procedure:** terminate the connection

Connection Parameters

Upon sending the connection request, the initiator sends the connection parameters. Once the connection is established, these parameters can be modified by either device.

- **Connection Interval**
- **Slave Latency**
- **Supervision Time-out**

In BLE connections, two devices send and receive data from one another only on a specific channel at a specific time. They will then meet a specific amount of time later at a new channel, according to the frequency hopping algorithm handled by the Link

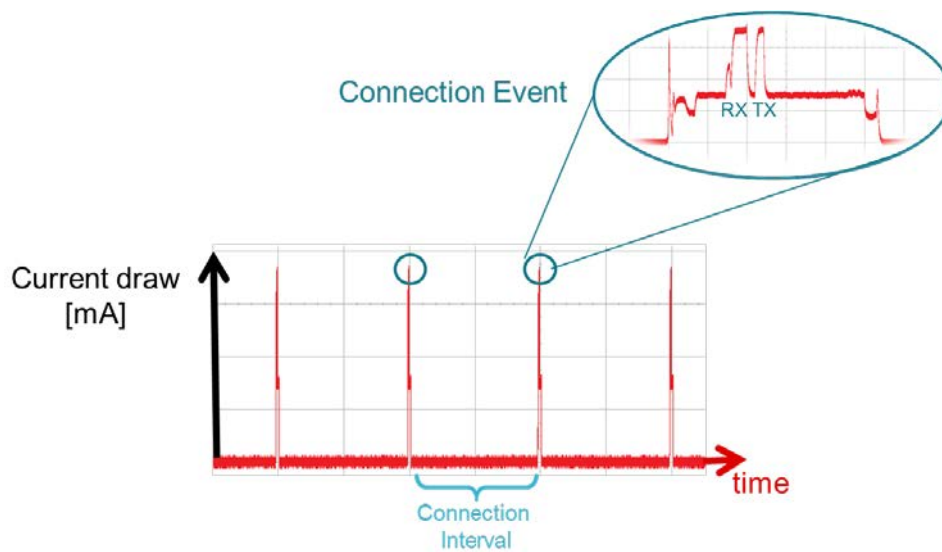


Figure 1.12: Connection Event and Interval

Layer. This meeting where the two devices share data is defined as a *connection event*. If there is no application data to be shared, the devices exchange Link Layer data to maintain the connection.

The **connection interval** is the amount of time between two connection events. Its value is defined in units of 1.25 ms and it can range from a minimum value of 6 (7.5 ms) to a maximum of 3200 (4.0 s). The connection is established with the connection interval parameter value set by the master. The peripheral may request different values, but the master has the final say and may choose different values. This feature allows for dynamic changes in the interval parameter during a connection, for example when bigger data transfer are required. The BLE stack allows setting a minimum and maximum value for connection interval. This allows the central device to choose within a range which is acceptable both in terms of power consumption and throughput. Figure 1.12 shows the current consumption during the connection events.

Reducing the connection interval does as follows:

- Increases the power consumption for both devices
- Increases the throughput in both directions
- Reduces the time for sending data in either direction

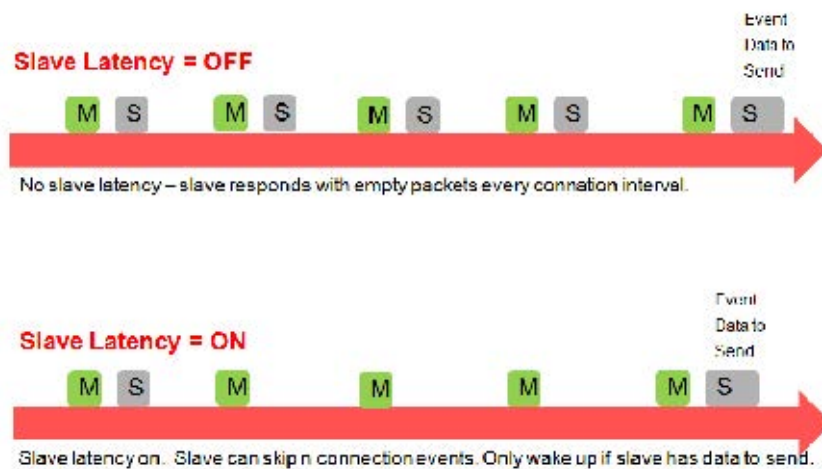


Figure 1.13: Slave Latency

Increasing the connection interval does as follows:

- Reduces the power consumption for both devices
- Reduces the throughput in both directions
- Increases the time for sending data in either direction

The **slave latency** parameter gives the slave device the option of skipping a number of connection events if it does not have any data to send (Figure 1.13). This ability gives the peripheral device some flexibility, since it allows it to stay asleep and save power. The peripheral can skip connection events but must not skip more than allowed by the slave latency parameter or the connection fails. The slave latency value represents the maximum number of events that can be skipped. This number can range from a minimum value of 0 (no connection events can be skipped) to a maximum of 499. The maximum value must not make the effective connection interval greater than 16 s [15].

Reducing the slave latency (or setting it to zero) does as follows:

- Increases the power consumption for the peripheral device
- Reduces the time for the peripheral device to receive the data sent from a central device

Increasing the slave latency does as follows:

- Reduces power consumption for the peripheral during periods when the peripheral has no data to send to the central device
- Increases the time for the peripheral device to receive the data sent from the central device

The **supervision time-out** is the maximum amount of time between two successful connection events. If this time passes without a successful connection event, the device terminates the connection and returns to an unconnected state. This parameter value is represented in units of 10 ms and it can range from a minimum of 10 (100 ms) to a maximum of 3200 (32.0 s). The timeout must be larger than the effective connection interval.

The **effective connection interval** is equal to the amount of time between two connection events, assuming that the slave skips the maximum number of possible events, if slave latency is allowed. The effective connection interval can be computed as follows:

$$EffectiveConnectionInterval = (ConnectionInterval) * (1 + SlaveLatency) \quad (1.3)$$

1.2.3 Host Controller Interface

HCI is a standard protocol that takes care of the communication between the Controller and the Host, which is the core of the BLE stack. Its role is to define a set of commands and events in order to translate raw data into data packets and viceversa. In a pure network processor application the HCI layer is implemented through a transport protocol such as SPI or UART. In embedded wireless MCU projects, the HCI layer is implemented through function calls and callbacks within the wireless MCU. All of the commands and events discussed, such as ATT, GAP, and so forth, pass from the upper layers of the protocol stack through the HCI layer to the controller. Likewise, the controller sends received data and events to the host and upper layers through HCI. As well as standard BLE HCI commands, a number of HCI extension vendor-specific commands are available which extend some of the functionality of the controller for use by the application.

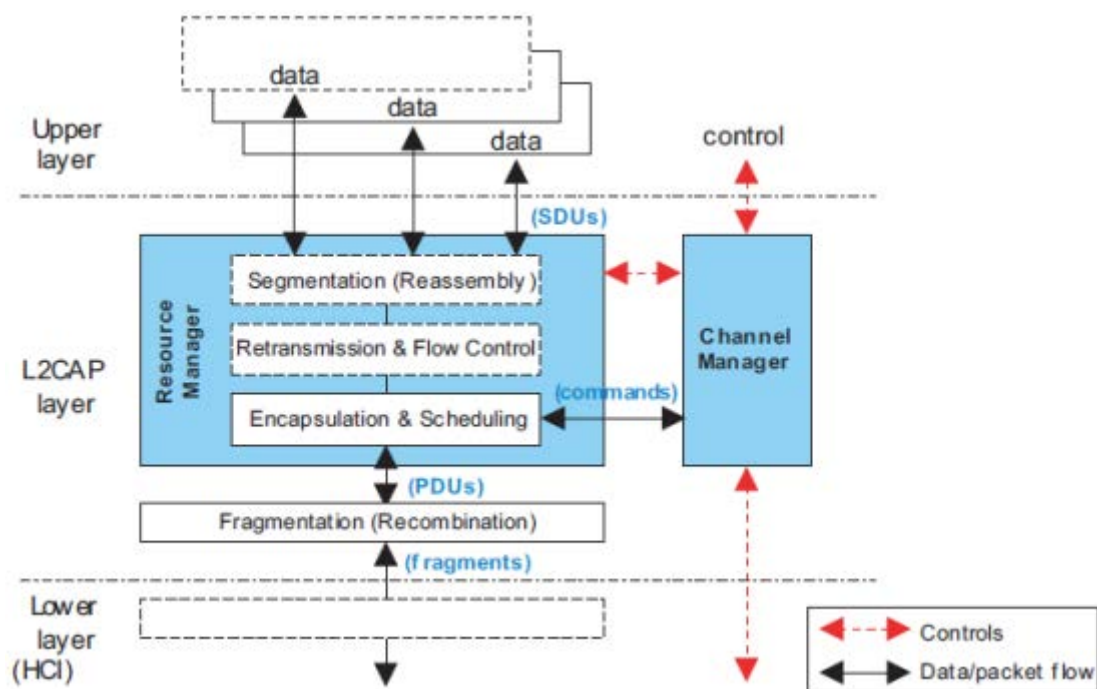


Figure 1.14: L2CAP Architectural Blocks

1.2.4 Logical Link Control and Adaptation Protocol

The L2CAP layer sits on top of the HCI layer on the host side and transfers data between the upper layers of the host (GAP, GATT, application) and the lower layer protocol stack. This layer is responsible for protocol multiplexing capability, segmentation, and reassembly operation for data exchanged between the host and the protocol stack. Figure 1.14 shows the L2CAP layer architecture. L2CAP permits higher-level protocols and applications to transmit and receive upper layer data packets (L2CAP service data units, SDU) up to 64KB long.

L2CAP is based around channels. Each endpoint of an L2CAP channel is referred to by a channel identifier (CID). Channels can be divided into fixed and dynamic channels. For example, data exchanged over the GATT protocol uses channel 0x0004. A dynamically allocated CID is allocated to identify the logical link and the local endpoint. The local endpoint must be in the range from 0x0040 to 0xFFFF.

1.2.5 Security Manager Protocol

SMP is composed of several security algorithms in order to encrypt and decrypt data packets.

1.2.6 Attribute Protocol

ATT defines the roles of a client-server architecture. It also performs data organization into attributes, to which is assigned a handle, a Universally Unique Identifier (UUID), a set of permissions and a value. Figure 1.15 shows the general attribute structure:

- Handle: 16-bit identifier, it determines the order of sequence of attributes that a client can access.
- UUID: 128-bit number that is globally unique.
- Properties: access permissions to determine what data can be read and/or written by the client. Each attribute can have the following access permissions:
 - Read: the attribute can be read by the client.
 - Write: the attribute can be written by the client.
 - Notify: the client can request a notification for a particular attribute from the server. Once the client enables the notification, the server will send the value to the client whenever it becomes available or it is changed.
 - Indicate: it's similar to notification, but it has the additional feature of receiving a confirmation message back from the server. This way the server knows that the message has reached the client.
- Value: value of the attribute, described by the attribute type (UUID).

1.2.7 Generic Attribute Profile

GATT encapsulates the ATT layer, its main role is to group attributes according to the attribute type. From a GATT standpoint, when two devices are connected they are each in one of two roles:

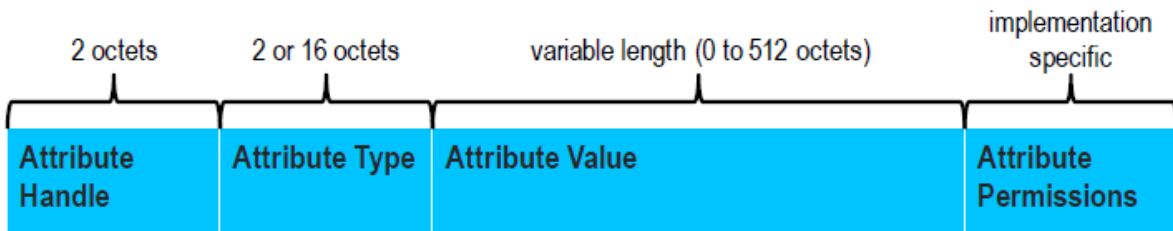


Figure 1.15: Logical Attribute Representation

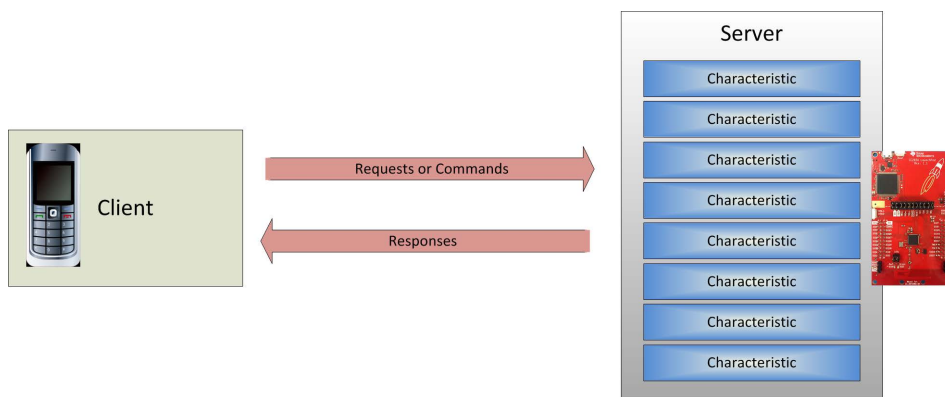


Figure 1.16: GATT Client and Server

- **GATT server** is the device containing the characteristic database that is being read or written by a GATT client.
- **GATT client** is the device that is reading or writing data from or to the GATT server.

GATT has the same client-server architecture as ATT, except that data are collected into services and characteristics. Figure 1.16 shows an example of client-server architecture between a smartphone and an embedded device.

Characteristics represent the lowest level concept in GATT transactions. Each characteristic has several attributes:

- **Declaration** (UUID: 0x2803): it always comes before the value, it describes whether the value attribute can be read or written, it contains the UUID of the characteristic and the handle of the value attribute.
- **Value**

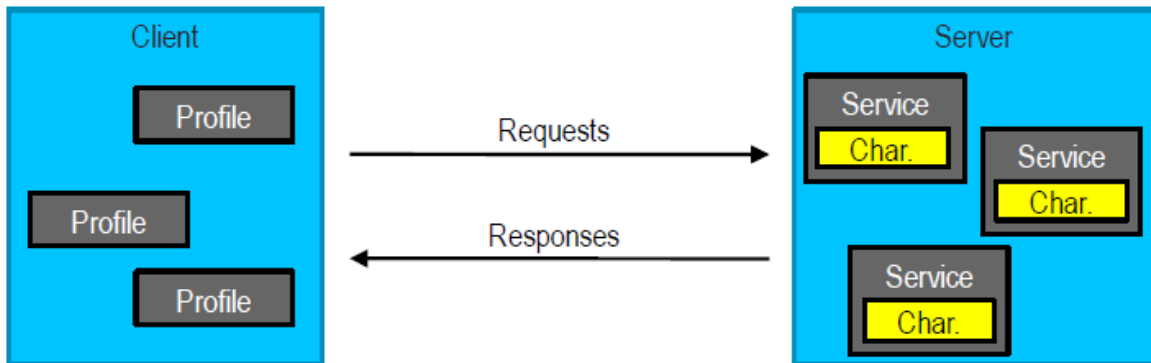


Figure 1.17: Profiles and Services

- **Descriptors:** they can give a description of the characteristic in string format, or describe how the value should be interpreted, or configure whether the GATT server may send notifications about value changes.

In particular, the **Client Characteristic Configuration Descriptor (CCCD)** is an attribute with UUID 0x2902 and is readable and writable. The value a GATT client writes to this attribute will determine whether the GATT server is allowed to send notifications or indications.

Services are collection of characteristics. Similarly to characteristics, services starts with a declaration. Every attribute from the declaration of a service up to the declaration of another service is a member of the service. There are two types of services:

- **Primary Service:** it exposes primary usable functionality of a device.
- **Secondary Service:** it provides the auxiliary functionality of a device and is referenced by at least one other primary service on the device.

Profiles defines a collection of one or more services and how services can be used. A profile can interpret UUIDs and values of the characteristics contained in the services. Figure 1.17 shows the relationship between profiles and services in a client-server architecture.

1.2.8 Generic Access Profile

GAP is collocated at the highest level, directly interfaced to the Application Layer. The GAP layer handles the access modes and procedures of the device including device discovery, link establishment, link termination, initiation of security features, and device configuration.

GAP defines the following roles when operating over the LE physical channel:

- **Broadcaster:** the device can only send advertising events.
- **Observer:** the device can only receive advertising events.
- **Peripheral:** the device can accept the establishment of a LE physical link using any connection procedure. It will be in the slave role in the Link Layer Connection State.
- **Central:** the device can initiate the establishment of a physical connection. It will be in the master role in the Link Layer Connection State.

With the advertising, the peripheral can specify its connectable mode as follows:

- Not connectable - default
- Directed connectable - can connect to a specific device
- Undirected connectable - can connect to any device

The central can define one of the following connection establishment procedures:

- Auto Connection - automatically connect to a set of devices (whitelist)
- General Connection - connect to any device, allowing also private connections
- Selective Connection - connect to a set of devices, separate configuration per device
- Direct Connection - connect to a specific device

The combination of connection procedures and connectability gives the diagram depicted in figure 1.18.

	Non-Connectable	Directed Connectable	Undirected Connectable
Auto Connection	No	Yes if in list	Yes if in list
General Connection	No	Yes if in list	Yes if in list
Selective Connection	No	Yes if in list	Yes if in list
Direct Connection	No	Yes	Yes

Figure 1.18: Connection Procedures and Peripheral Connectability

Figure 1.19 shows the relationships between roles defined in LL, GATT and GAP layers. The example is set on a standard Heart Monitor application with a Heart Rate Profile. The Heart Rate Profile contains a Heart Rate Service and a Device Information Service. Within the Heart Rate Service, there are three Characteristics, each containing different information. The device in the diagram is configured as a Sensor role, meaning that in the context of the Heart Rate Profile, the device is a GAP Peripheral and a GATT Server.

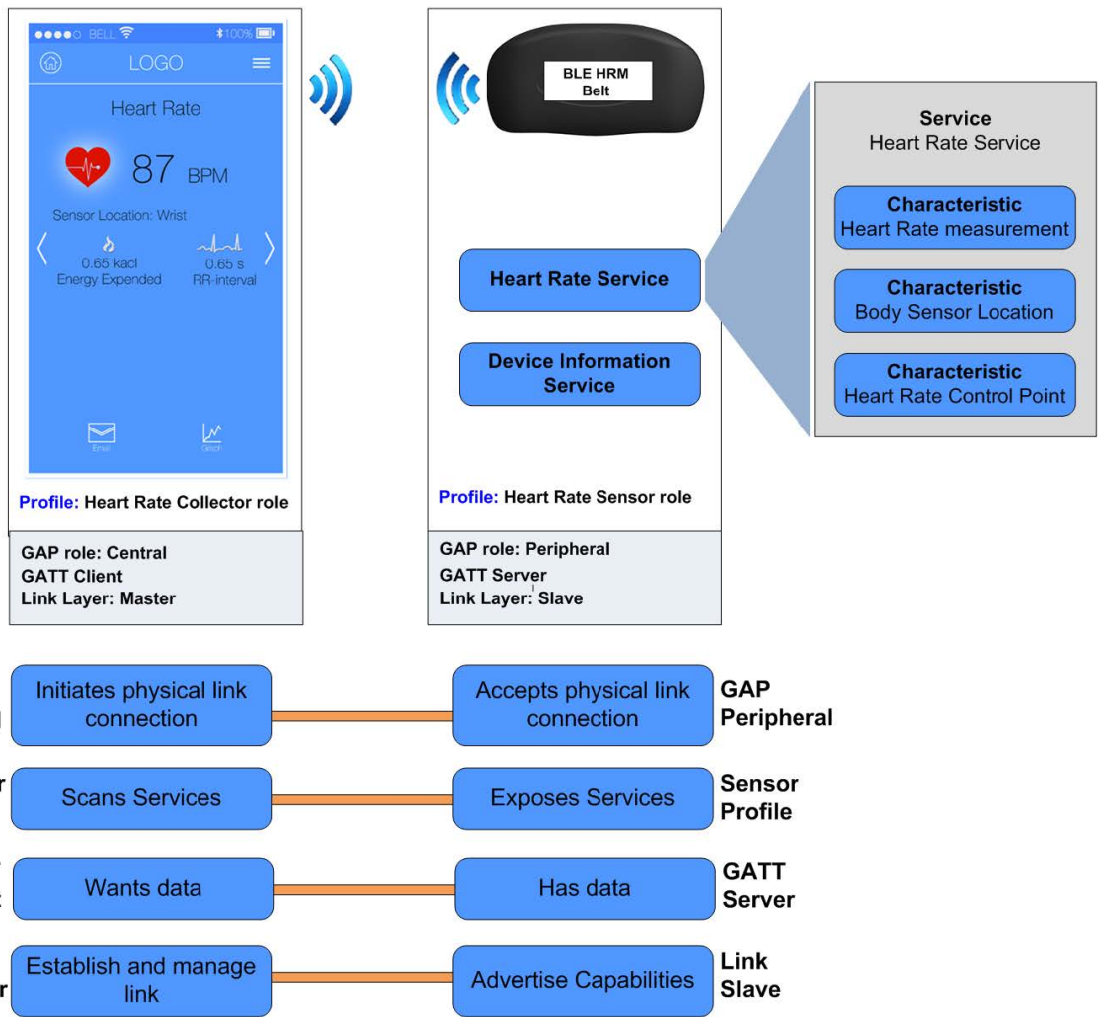


Figure 1.19: BLE Heart Monitor Application

Chapter 2

Software Implementation

2.1 Tools

The RD department decided for an embedded device to be put inside the shoes, under an orthopedic plantar. Following from the starting point of the design, the solution has been a custom board in which the BLE module is interfaced with an IMU sensor which collects motion data for telemonitoring. The purpose of this thesis is the design of an application to collect sensory data from an inertial module and send them to a smart-phone app through BLE protocol. The CC2650 Microcontroller Unit (MCU) from Texas Instruments (TI) has been chosen by the RD department, therefore all software implementations have been developed using Texas Instruments tools. The CC2650 device is a wireless MCU targeting Bluetooth, Zigbee and 6LoWPAN remote control applications [13]. The device is a member of the CC26xx family of 2.4-GHz RF devices. Before printing the custom board, the software has been developed on TI certified Bluetooth Low Energy boards.

The hardware used in this project comes from two Development Kits from Texas Instruments, composed of:

- CC2650 Launchpad
- CC2650 Boosterpack

The Boosterpack has been selected because it mounts the CC2650 MCU on a compact module that can be easily mounted on custom boards [11]. According to [11], the

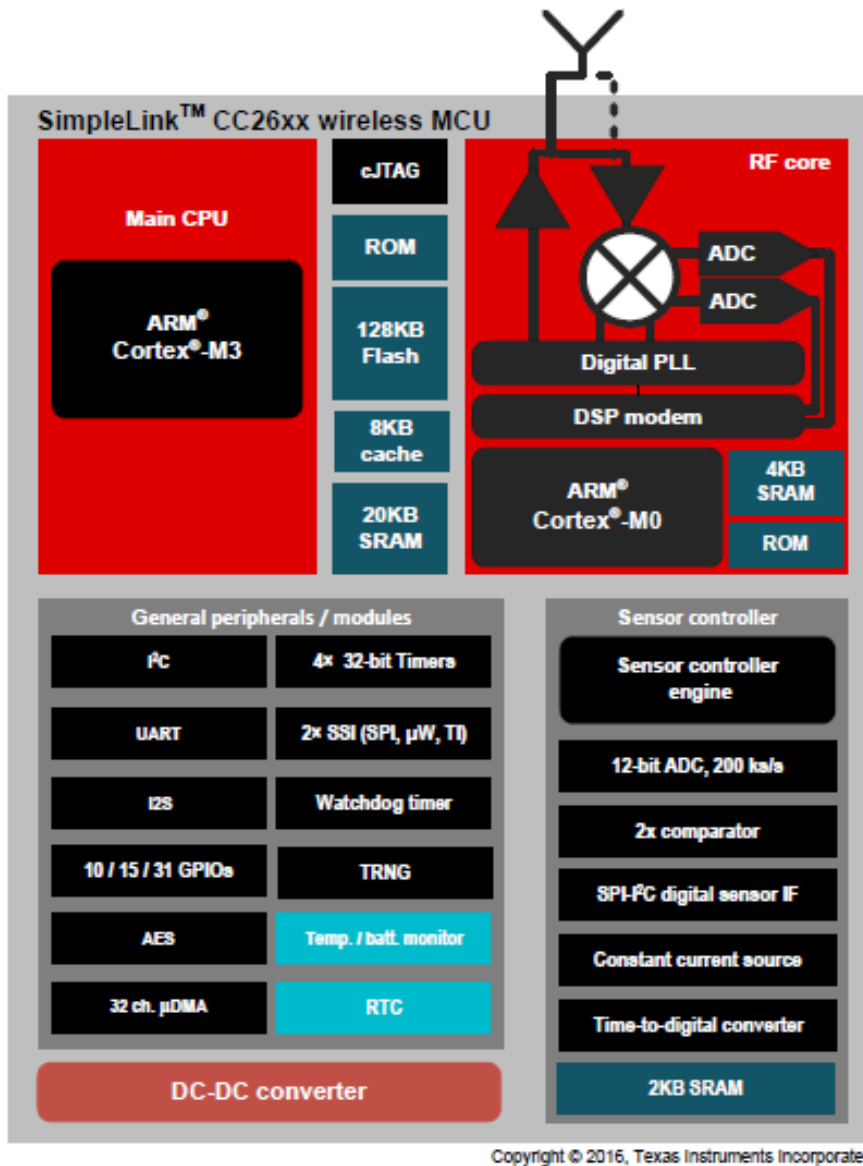


Figure 2.1: CC2650 Block Diagram

recommended procedure was to develop the software on the Launchpad, then port it on the Boosterpack. Figure 2.1 shows a block diagram for the CC2650 MCU. It contains an integrated antenna, an ARM Cortex-M3 32-bit MCU, in-system flash memory, 15 I/Os and it's precertified for FCC/IC, CE and ARIB radio standards [11]. When used with TI's BLE-Stack, the module also has a Bluetooth RF-PHY qualified component.

The software tools used in this project are from Texas Instruments as well, in particular:

- Code Composer Studio (CCS) - Integrated Development Environment for TI microcontrollers and embedded processors
- TI BLE Software Development Kit (SDK) - software kit for BLE applications
- TI Simplelink Academy - workshop with lab assignments and practical examples
- TI Sensor Controller - high level IDE for programming the Sensor Controller, a separate CPU core that can be used to read and process sensor data independently from the main application

2.2 Implemented Solution

2.2.1 Project Zero

The software development started with the implementation of the *Project Zero*, an example of code for a BLE device in peripheral role given in the TI Simplelink Academy. As defined in the Bluetooth Low Energy protocols, a device in peripheral role advertises its availability for connection and provides an interface for communication. The standard *Project Zero* offers custom services. These services allow the user to connect with a smartphone to remotely toggle on-board LEDs, send text strings to the LaunchPad serial port, and receive button press notifications. All the custom service identifiers (UUIDs) are on the form F000XXXX-0451-4000-B000-000000000000 where each service has a different 16-bit XXXX-part identifier, and the rest of the UUID is the Texas Instruments 128-bit UUID namespace used for demo purposes. The services can be identified from the 16-bit part of the UUID as follows:

- 0x1110 LED Service - Service Declaration

- 0x1111 LED0 State - Read state or write 01 or 00
- 0x1112 LED1 State - Read state or write 01 or 00
- 0x1120 Button Service - Service declaration
- 0x1121 BUTTON0 State - Read state or subscribe to notifications
- 0x1122 BUTTON1 State - Read state or subscribe to notifications
- 0x1130 Data Service - Service Declaration
- 0x1131 String char - Read/Write a long string
- 0x1132 Stream char - Send or receive WriteNoRsp/Notification

Once asserting the performances of the device and the communication with an Android smartphone, the BLE services were modified to collect and send data from the sensors. After discussion with the rest of the team, the services have been defined as follows:

- 0x1170 Battery Service - Service Declaration
- 0x1171 Left Battery - Read state or subscribe to notification
- 0x1172 Right Battery - Read state or subscribe to notification
- 0x1140 Imu Service - Service Declaration
- 0x1141 Left IMU - Read state or subscribe to notification
- 0x1142 Right IMU - Read state or subscribe to notification

Both services contain a single characteristic with Read/Notify permissions. However, to simplify data collection on client side, the UUID of the same characteristics is different to distinguish between the boards mounted on the left and right plantar. Also, to allow such distinction before the connection, the name of the device in the advertising packet is distinguished between left and right, namely we have *PLTSX* and *PLTDX*. The implemented services will now be described in detail.

2.2.2 Battery Service

As the device runs on a portable battery, the application is required to communicate periodically the amount of electrical charge left for the device to properly function. The voltage of the battery pin is read through a 12-bit ADC and stored in a 2 bytes variable. The value is measured in mV and converted in a percentage value over the total voltage set in the battery datasheet. This conversion is required to give the app user a simpler status on the amount of battery left before recharge. The periodicity of the data communication is set by an internal clock with a period of one second. The Battery Service characteristic has a 4 bytes value. The first two bytes represent the battery level, the others indicate the number of steps computed by the inertial module. Initially this step counter was collected and stored in the IMU Service, but the RD department decided to have it updated only once per second. Therefore it has been inserted into the battery characteristic, since that structure is already being updated once every second. In the testing environment, the one second update period represented a midpoint between power consumption and user-friendly experience.

2.2.3 Imu Service

The IMU sensor sends the collected data through SPI peripheral. For monitoring purposes, data from accelerometer and gyroscope sensors are collected. The student also implemented a sequence number to synchronize data on the client side. All these data are separately stored in 8-bit variables by the Sensor Controller. The IMU Service is not updated periodically through an internal clock, but it relies on a Data Ready signal sent by the inertial module when the accelerometer has new data available for collection. Once the signal is received, the main CPU proceeds to collect the sensory data with a specific order and stores it in an array. Then the array is set as a 14 byte value for the IMU characteristic. The 14 bytes are assigned in the following way: 6 bytes for accelerometer xyz output data, 6 bytes for gyroscope xyz output data, and 2 bytes for the sequence number.

2.3 Current Consumption Optimization

The main goal of the embedded design is to satisfy the need for a device for collection and transmission of inertial data for medical purposes. However, the design must also be consistent with a low power consumption. Since the final device will have to be as much energy autonomous as possible with its energy harvester, the basic protocol must be as less energy consuming as possible.

The main factors affecting current consumption in a BLE device are the transmission power and the amount of time that the radio is active [7]. Power transmission has to take into account the physical distance between the transmitting antenna and the receiving one. In real applications, the actual range of the transmission is also greatly influenced by obstacles in the environment and traffic in the chosen frequency band [5]. The amount of time a radio is active depends on how often it must transmit or receive and the time required to complete that operation. Frequency of communication is determined by the specific services that the device have to offer, while time of transmission is mainly influenced by the length of data to be sent or received [7].

The power consumption of a BLE device can be optimized by fine-tuning some specific parameters according to the working constraints and on the resulting user experience, since the devices are expected to interact with a smartphone app. In the following sections, the parameter optimization for power consumption will be considered.

2.3.1 TX Power Level

The BLE protocol defines a transmission power that ranges from 5 dBm to -21 dBm [10]. A higher power offers a wider range of action for the device, at expense of an higher power consumption. Figure 2.2 shows how the current consumption changes in relationship with the selected Tx power, the connection interval and the duty cycle of the radio. For the specifications of the project, a default transmission power of 0 dBm has been chosen, since lower values bring problems with the reception of data packets.

2.3.2 Advertisement Interval

When a BLE peripheral device is in advertising mode, advertising packets are sent periodically on each advertising channel. The time interval between packet sets has both a

interval (ms)	RF duty cycle	average current		
		0 dBm	3 dBm	8 dBm
7,5	20,0 %	970	1070	1170
20	7,5 %	329	362	402
40	3,8 %	168	185	202
100	1,5 %	75	82	89
200	0,8 %	41	45	49
500	0,3 %	20	22	23
1000	0,2 %	13	13	14
2000	0,1 %	9	9	10

Figure 2.2: Tx Power and Current Consumption

fixed interval and a random delay. The fixed interval can be set between 20 ms and 10.24 s, in steps of 0.625 ms. The random delay is a pseudo-random value from 0 ms to 10 ms that is automatically added. The random component helps reducing the possibility of collisions between advertisements of different devices [7, 9, 10].

The wide range of choice of advertisement interval can be reduced to three main guidelines [9]:

- [< 100] ms - for very aggressive connections and usually for shorts periods of time.
- [100, 500] ms - normal fast advertising for most devices.
- [1000, 2000] ms - for devices that connect to gateways and where latency is not critical.

Though increasing advertising interval reduces power consumption exponentially [9], it is important to consider that a longer waiting time for the connection of the device can be bad for user experience, especially for devices that require to be connected only for small periods of time.

For the purpose of this project an advertisement interval of 1000 ms was chosen. Longer intervals proved inefficient for the user experience, since most of the times the actual waiting time to connect amounts to twice the advertising interval [9, 10].

2.3.3 Advertising Packet

The length of the packet to be transmitted during advertising greatly influences the power consumption [2, 9]. According to the Bluetooth Specification, the packet data unit for the advertising channel is defined as Advertising Channel PDU [2]. It contains a 2-byte header and a variable payload from 6 to 37 bytes. The actual length of the payload is defined in the 6-bit Length field in the header. For peripheral devices, the PDU type is set as ADV_IND [2]. ADV_IND PDU is used for connectable undirected advertising events [2].

For connectable devices, the payload has an Advertisement Address of 6 bytes and a variable number of advertisement data structures. This means that the advertisement packet may contain at most $37 - 6 = 31$ bytes for actual advertisement data. Each advertisement data structure is identified by a Data Type. The most commonly used Advertising Data Type are:

- 0x06 Incomplete List of 128-bit Services: UUIDs of the services provided by the peripheral can be advertised in 128-bit format.
- 0x02 Incomplete List of 16-bit Service Class UUID.
- 0x08 Shortened Local name
- 0x09 Complete Local Name
- 0xFF Manufacturer Specific: for custom payloads.

The choice of advertising services provided by the peripheral is decisive for the central device to immediately filter out unwanted devices, since you do not have to connect to know what services the peripheral has to offer. For the purpose of this project, the advertisement data packet contains information about discovery mode and local name of the device. Specifically, the device supports General Discovery Mode, meaning that the device advertises indefinitely, until a connection request is received. In section 2.4.2 wake up functions programmable in the inertial module are considered. Implementing a wake up routine for the CC2650 module may allow to define a Limited Discovery Mode, meaning that advertising is kept for a limited amount of time before the device goes into sleep. Then, the Sensor Controller may wake up the main module after detecting a significant motion, which is defined according to the selected detection algorithm.

2.4 Sensor Controller

Sensor Controller is an ultra-power, 16-bit CPU core. It is programmable and allows users to read and process data to make low level decisions while the rest of the system sleeps. The Sensor Controller can then wake up the system to perform more computationally-intensive tasks or transmit a message with the radio. TI offers a custom IDE to create Sensor Controller tasks **Sensor Controller Studio** is used to write, test and debug code for the CC26xx Sensor Controller. The tool generates a Sensor Controller Interface driver, which is a set of C source files to be compiled into the System CPU (ARM Cortex-M3) application. These source files contain the Sensor Controller firmware image and associated definitions, and generic functions that allow the System CPU application to control the Sensor Controller and exchange data. Sensor Controller tasks are implemented using a programming language with syntax similar to C, but with limited features. The task code is divided into blocks that represents function bodies that are called through the Sensor Controller's firmware framework.

- The system CPU application starts Sensor Controller tasks by triggering the **Initialization Code** block
- The Initialization Code must schedule the first execution and/or setup the initial event trigger(s)
- The **Execution Code** and **Event Handler Code** blocks must keep the task alive by scheduling the next execution and/or setup new event triggers
- The system CPU application stops Sensor Controller tasks by triggering the **Termination Code** block
- The Termination Code must cancel any currently enabled event triggers (scheduling is deactivated automatically)

The Event Handler Code can be triggered by events from timers, I/O input pins and other signals on the AUX event bus. The Sensor Controller firmware framework enters standby mode only when not running task code blocks. All code blocks run to completion (no preemption), while context switching can only occur when the Sensor Controller is ready to enter or has already entered standby mode. Upon task creation, the user

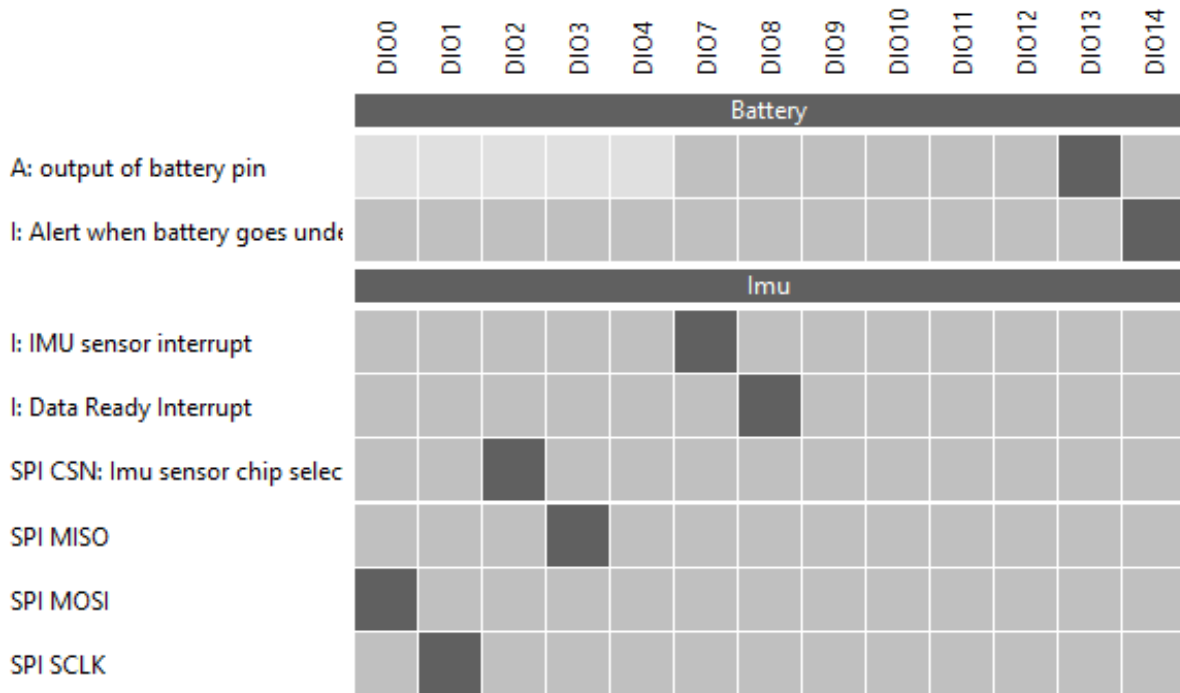


Figure 2.3: I/O Mapping for Sensor Controller

must specify the resources that the Sensor Controller will have access to, such as pins, comparators, timers, alerts, triggers, real-time scheduling, peripheral etc. Figure 2.3 shows the initialized pins for each task.

For the purpose of this study, two tasks have been programmed within the Sensor Controller: the *Battery Task* and the *IMU Task*. The tasks will be described in the following sections.

2.4.1 Battery Task

As the device runs on a portable battery, it is mandatory for the user to be able to check the battery status through the smartphone app. The battery task implements a 12-bit ADC that is used to convert the analog voltage measured at the indexed pin into a digital data, measured in mV.

The task also implements an interrupt that activates when the battery voltage goes under 3.3 V, for safety measures. As soon as the voltage reaches the threshold, the battery is disconnected from the board.

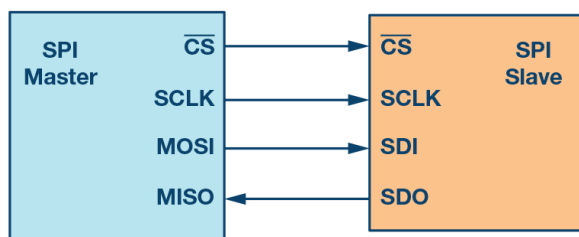


Figure 2.4: SPI Single Master to Single Slave

2.4.2 Imu Task

The IMU task works with the SPI peripheral to interact with the IMU mounted on the board. During the project development, two inertial modules have been tested and implemented for the final product.

LSM6DSL from STMicroelectronics is an always-on 3D accelerometer and 3D gyroscope that met the required project specifications in term of sensors performance and low power features [16].

LSM6DSOX is an upgraded version of LSM6DSL, with additional features that will not be considered for the purpose of this thesis. All the software choices that have been made are compatible with both the inertial devices.

SPI Communication

SPI (Serial Peripheral Interface) is a synchronous serial communication interface. SPI devices supports full-duplex mode with a master-slave architecture with a single master. Figure 2.4 shows the four wire structure that characterises this peripheral. The four logic signals shown in figure 2.4 are defined as follows:

- **CS**: Chip Select, also known as Slave Select (SS), used to activate communication with the desired slave.
- **SCLK**: Serial CLock, to synchronize the communication.
- **MOSI**: Master Out Slave In (Slave Data In on slave side), data transmitted from master to slave.
- **MISO**: Master In Slave Out (Slave Data Out on slave side), data transmitted from slave to master.

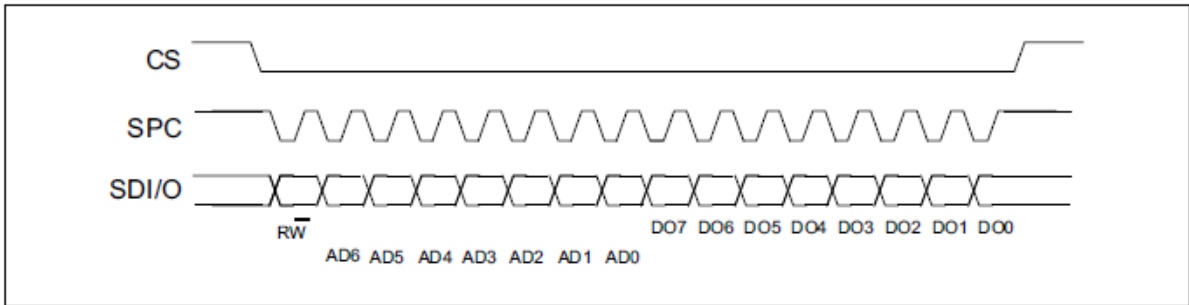


Figure 2.5: SPI Read Protocol

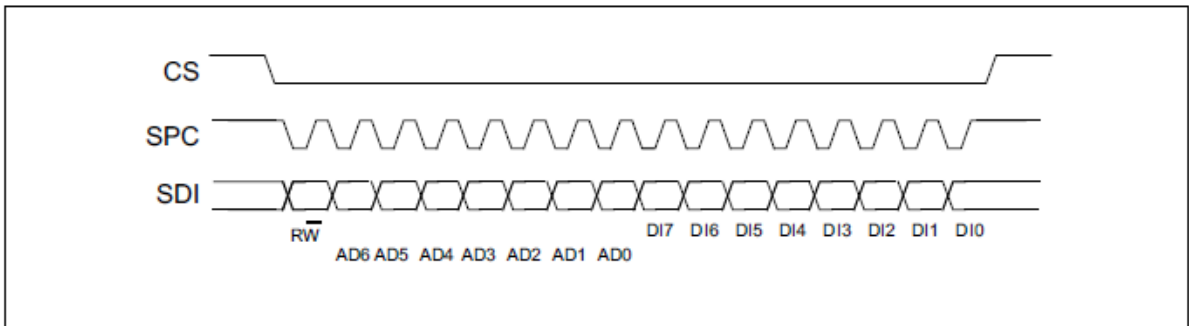


Figure 2.6: SPI Write Protocol

Figure 2.3 shows the mapping for SPI pins. LSM6DSL defines SPI read and write protocol as follows.

SPI read is performed with 16 clock pulses. A multiple byte read command is performed by adding blocks of 8 clock pulses to the previous one. Figure 2.5 shows the procedure in detail. The master sends a byte of data and the slave answers with one or multiple bytes. The master byte contains the READ bit (value 1) and a 7-bit address of the desired register. The slave byte contains the value read from the indexed register, or multiple values in case of multiple read command.

SPI write is similar to SPI read, except that the first bit written by the master is the WRITE bit (value 0) and the master sends both the address of the desired register and the value that must be written on that same register. Figure 2.6 shows the procedure in detail.

Sensor Controller Studio supports multiple byte read/write operations up to two bytes. It also allows to read from multiple contiguous registers just by specifying the first address, each additional read procedure will be executed on the next address in the

list.

Accessed Registers

The LSM6DSL inertial module has to be programmed by setting specific bits into the indexed registers required. Register description is not examined in detail here, only the main registers will be mentioned. All the necessary information is contained in the inertial module datasheet [16, 18]. Therefore all the registers will be distinguished by address for simplicity. The main registers to be programmed are the following:

- **(0x15)-(0x16)**: set up low power mode for accelerometer and gyroscope.
- **(0x19)**: enable step counter. The module has a built-in pedometer algorithm that can use ± 2 g or ± 4 g as full scale.
- **(0x10)-(0x11)**: set up full-scale and output data rate for accelerometer and gyroscope. For the final product, the RD department asked for an output data rate of 104 Hz. The full scales have been set up to ± 8 g for the accelerometer, ± 1000 dps for the gyroscope.
- **(0x0E)**: enable interrupt with data ready signal from accelerometer. The interrupt will activate every time the accelerometer data is updated.

The output registers to be read are:

- **(0x22) to (0x27)**: gyroscope xyz output data.
- **(0x28) to (0x2D)**: accelerometer xyz output data.
- **(0x4B)-(0x4C)**: step counter output data.

This procedure provides the basic requirements of sensory data collection. The main CPU will then collect the data and set them as a characteristic value inside the IMU service. Other than the internal sensors, the inertial module provides embedded low power extra features [16]. The low power features that the student has been tasked to study and evaluate are the following:

- Timestamp storage

- Pedometer/Step counter
- Activity/Inactivity recognition
- Tilt detection

The **Timestamp** is a number that indicates the amount of time that has passed from a specific moment. The RD department tasked the student with the collection of timestamps to allow the smartphone app to associate a time variable to the evolution of inertial variables. Evaluations with the app programmer deemed preferable to replace the timestamp with a sequence number that is incremented with each packet.

The **Pedometer** is an embedded algorithm that uses data from the accelerometer to detect and count steps. As already mentioned, the algorithm works at 26 Hz, so the accelerometer output data rate have to be set at 26 Hz or higher. The programmable values for the full scale are limited to ± 2 g and ± 4 g. For the purpose of this project, the pedometer algorithm is implemented with a full scale of ± 4 g. Tests conducted with the board fixed on the plane and with the board installed in the plantar for walking routines outlined that the algorithm overestimates the number of steps. Therefore the step counter is kept as a reference, while the actual value is directly computed through data analysis and will be addressed in the following chapters.

The **Activity/Inactivity Recognition** function allows the device to automatically decrease the accelerometer sampling rate to 12.5 Hz when a certain number of consecutive output data is smaller than a configurable threshold. This feature can be extended to the gyroscope with three possible options:

- Gyroscope configurations do not change.
- Gyroscope enters in Sleep mode.
- Gyroscope enters in Power-Down mode.

When a single sample of accelerometer data becomes bigger than the threshold, the accelerometer and gyroscope settings are immediately restored. This detection procedure allows to drastically reduce the current consumption, assuming that the device spends continuous time windows being unused. Figure 2.7 shows how the status of the device changes according to the slope of the accelerometer data.

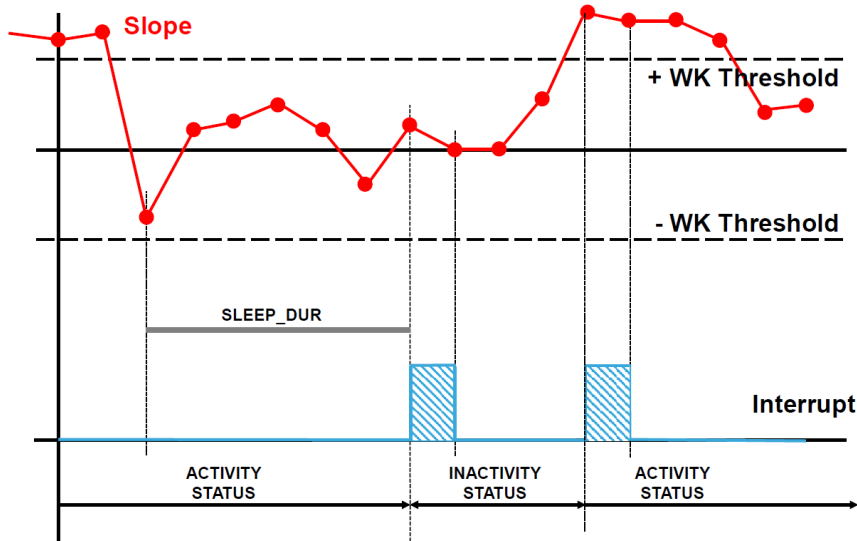


Figure 2.7: Activity/Inactivity Detection

The **Tilt Detection** is implemented using only the accelerometer. An interrupt is generated when the device is tilted by an angle greater than 35 degrees from the start position. The start position is defined as the position of the device when the tilt detection is enabled or the position of the device when the last tilt interrupt was generated. For the generation of the first tilt interrupt the device should be continuously tilted by an angle greater than 35 degrees from the start position for a period of time of 2 seconds. Figure 2.8 shows the tilt detection procedure for a generic starting position. This feature has been evaluated to give the device a gesture recognition procedure. The idea is that the device can switch on upon a continuous rotation of over 35 degrees, otherwise it stays in deep sleep, reducing power consumption during the day.

It is important to underline that both activity/inactivity and tilt detection can be implemented to provide different layers of power consumption, for example in case of long, continuous time windows in which the device is switched off. Then it can be turned on through tilt detection to be used for more than one movement activity that need constant monitoring. Activity/inactivity detection allows the device to reduce the consumption between the activities, assuming that the device is detached from the body at the end of each activity.

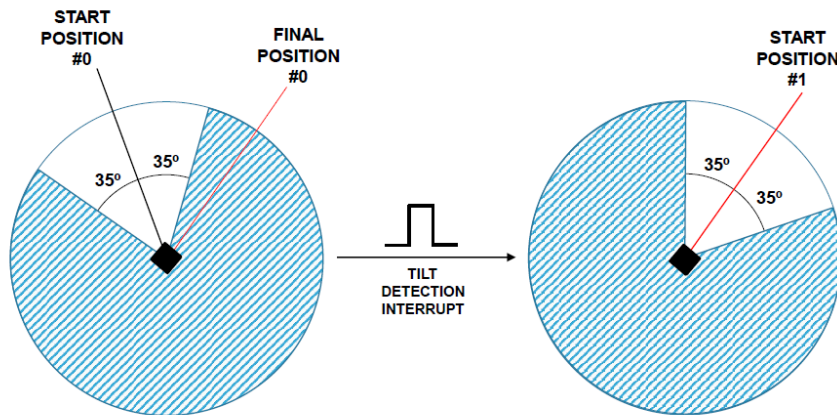


Figure 2.8: Tilt Detection

2.5 Experimental Results

The device consumption has been tested since the beginning of the software implementation. The first implementation of the project Zero example with customized services had a measured power consumption of 3680 μA .

The activity/inactivity recognition function reduced the power consumption of more than 60%, going from 3680 μA in the active phase to 1460 μA in the inactive phase.

Reducing the Tx power level to 0 dBm and increasing the advertising interval to 1000 ms further reduced the active power consumption to 1200 μA .

The following values depicts the power consumption of the final implementation of the BLE device:

- Advertising: 30-110 μA
- Active communication: 1200 μA - all the sensor controller tasks are active, battery updated at 1 Hz, sensor data at 104 Hz
- Inactive communication: 700-800 μA - gyroscope in power-down, accelerometer samples at 12.5 Hz, battery updated at 1 Hz

Part II

Gait Analysis

Introduction

Mobility describes a person's ability to change positions and move in the environment. Walking represents an important aspect of mobility that is commonly affected by ageing and chronic diseases. Mobility is particularly important for diseases involving a significant burden of mobility disability [32]. In recent years, mobile sensor-based gait analysis has become an applicable and objective tool for the assessment of motor impairments and disease progression in several neurological and musculoskeletal diseases [33]. This is possible because the IMUs have been upgraded from being an inaccurate device to motivate physically active adults to a reliable measurement tool to monitor mobility performances [32].

In Multiple Sclerosis (MS), studies with IMUs have identified a disease related reduction of gait quality, such as decreased stride length or gait speed [34, 35]. Gait is a complex sensorimotor activity that involves not only the spatial and temporal coordination of the lower limbs but also the coordination of the trunk and arms as well as the dynamic balance. Standard approaches to evaluate mobility for MS rely either upon scores in the Expanded Disability Status Scale (EDSS) [36] or upon recommended clinical tests, like timed or distance tests (6-Minute Walk Test, Timed 25-Foot Walk and others) [35].

The purpose of this project is to extend the standard clinical tests reproduction to long-term monitoring in real-time environments. This is done mainly to increase the volume of the acquired data [33]. However, a full-time monitoring with inertial sensors becomes challenging in presence of unsupervised activities, because it is not always possible to know a-priori if the subject is walking in a specific time window. Therefore, a segmentation procedure is required, to detect gait before the actual computation of further parameters. This thesis proposes a neural network approach to categorize the data patterns and isolate the walking sessions. Then, gait parameters are extracted from the

highlighted time frames to assess over-time changes in the mobility performance. The estimation of MS disease stages is left for future developments within clinical trials.

Chapter 3

Human Activity Recognition

3.1 Machine Learning

Machine Learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data [25]. ML is closely related to statistics, biology and informatics. It has multiple applications but it often deals with large dataset to create models and automatic systems that are able to solve complex problems. A few examples of application may be computer vision, classification and autonomous guidance. ML approaches differs from traditional programming because it is not based on strict rules and can often be used in problems that are yet to be solved [26, 28, 29].

Machine Learning approaches split the used data into three categories:

- Training Set: data used during training
- Validation Set: data used for tuning of algorithm parameters, usually called *hyperparameters*
- Test Set: data used to evaluate the sytem performances

This categorization has to respect some conditions. First, every set has to represent the starting set, without redundancies. Also, in case of time related data, they cannot be randomly mixed. Two of the most used techniques to satisfy these requirements are hold-out validation and k-fold validation.

Hold-out validation partitions data into two sets, namely the train and test set. Usually 80% of data will be used for training, while the rest will be left for testing.

K-fold validation consists in splitting the data into k groups. One of the groups is used as the test set and the others become training set. The model is trained on the training set and scored on the test set. Then the process is repeated until each unique group has been used as the test set.

K-fold is usually preferable because it gives the model the opportunity to train on multiple train-test splits. Hold-out, on the other hand, it's dependent on just one train-test split. K-fold takes more computational power and time to run, while Hold-out is better to use for large datasets and short amounts of time.

Going back to ML systems, another classification is based upon the learning approach:

- **Supervised Learning:** each training set data has a label attached to it, representing the class of the data.
- **Unsupervised Learning:** training set is not labeled, so the system has to classify the data on its own.
- **Semi-supervised Learning:** training set contains both labeled and unlabeled data.
- **Reinforcement Learning:** the system, defined as the agent, observes the environment and executes actions on it. Each correct action is rewarded, while a wrong action is labeled as punishment. The system will proceed to maximize rewards and minimize the punishments.

A ML system that is trained using the entire training set is called as batch system. Otherwise it is known as incremental system. Another classification for ML systems lies in the generalization of data:

- **Instance-based learning:** the system uses a measure of similarity to identify the class of the new data (Figure 3.1a).
- **Model-based learning:** the system creates a model from training set samples and uses it to predict the class of the new data (Figure 3.1b).

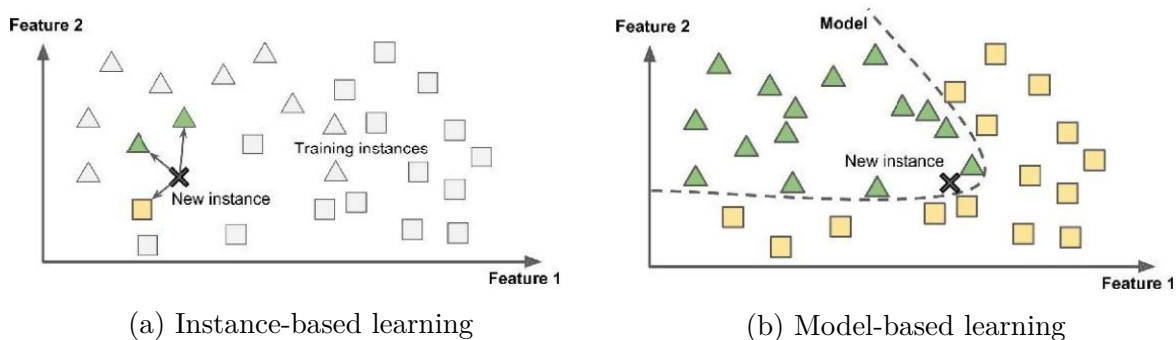


Figure 3.1: Instance-based learning vs. Model-based learning

3.2 Neural Networks

Neural Networks (NNs) are a series of algorithms that mimic the operations of a human brain to recognize relationships between vast amounts of data. A neural network works similarly to the human brain's neural network. NN is based on a collection of connected nodes called neurons. Each connection can transmit a signal to other neurons. A neuron that receives a signal then processes it and can signal neurons connected to it. The output of each neuron is computed by some non-linear function of the sum of its inputs. Neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Figure 3.2 shows three types of layers: input layers that take raw input from the domain, hidden layers that take input from another layer and pass output to another layer, and output layers that make a prediction [24].

A NN is based on neurons connected in a structure. The first approach to NN was a MISO binary architecture, in which the output was equal to 1 when at least a certain number of inputs was 1 [19]. One of the simplest neural network is the Perceptron [27]. Here, inputs and outputs are not binary and every input connection has a weight associated to it. Figure 3.3 shows a general model for neurons.

To model a neuron we need to define:

- Number of input channels
- Type of input signals
- Connection weights
- Activation function

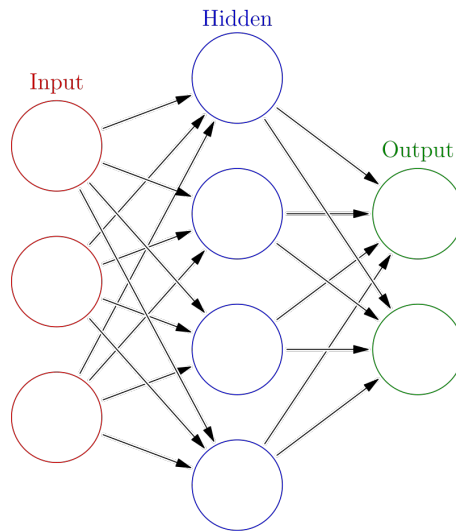


Figure 3.2: Three Layer Neural Network

The activation function defines how the weighted sum of the inputs is transformed into an output from a node.

The perceptron is the simplest feed-forward NN, meaning that the inputs directly influence the output through weighted connections. However, this structure was limited in the number of possible operations [27]. This limit can be solved by adding different perceptrons and using non-linear activation functions at every level creating a multi-level or multi-layer network. In layered networks the weights of each layer can be represented through a connection matrix. The external levels are the input layer and the output layer. The levels in between are called hidden layers. There can be zero to many hidden layers, each fully-connected to the following layer. A network with two or more hidden layers is called Deep Neural Network (DNN). Each layer has its own activation function [24].

The algorithm to train a multi-level neural network is called *backpropagation algorithm* (Rumelhart et al., 1986a). At first, the algorithm feeds input instances to the network and computes the output of each neuron at each level. This process is called *feed-forward step*. Then an error is computed as the difference between the desired output and the actual output. The algorithm, then, proceeds backwards to calculate how much each neuron has contributed to the total error. This process is called *back-propagation step*. The goal is to modify the weights of the connections to reduce the error. The

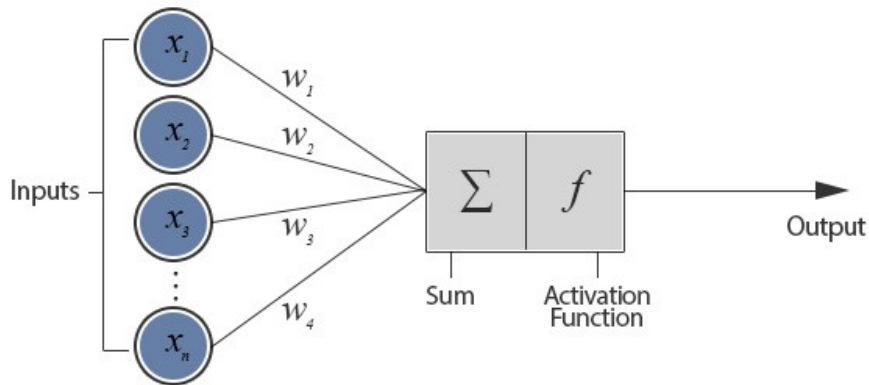


Figure 3.3: General Neuron Model

update of the weights usually follows the *Gradient Descent method*.

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. Each weight is changed in proportion to the derivative of the error with respect to that weight, provided that the non-linear activation functions are differentiable.

Figure 3.4 shows the most common activation functions:

- Rectified Linear Unit (ReLU): $R(z) = \max(0, z)$
- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$

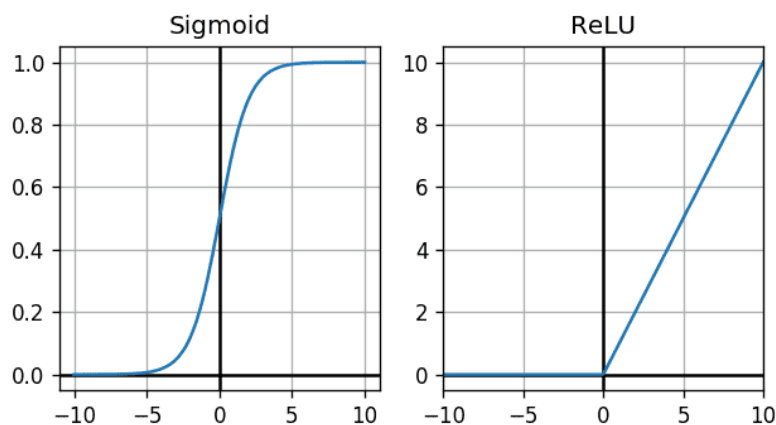


Figure 3.4: Activation Functions

The parameters that defines a neural network are also called *hyperparameters*. For example, the number of layers, the number of neurons per layer, the type of activation function, the initialization rule for the weights etc.

The number of hidden layers defines the classification ability of the network. For example, in feature recognition applications, the lower layers can identify low level structures like segments, intermediate layers combine these features into intermediate ones, like basic shapes. Then, higher layers and output layer can model high level structures, like faces or complex objects.

The choice of the number of neurons in hidden layers can lead to *overfitting*, meaning that the network corresponds too much to the training set and can fail to fit additional data or predict observations reliably. To prevent this condition training could be stopped before degenerating or techniques could be applied. For example, the *dropout* allows to shut down some neurons during training, to temporarily remove their contribution to weights updating.

3.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a class of NN where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour [23]. RNN derives from feedforward neural networks, adding an internal state (memory) to process variable length sequences of inputs. [22] describes the working principle of a standard RNN. The main distinction from standard NN is that the size of input and output vectors is fixed and the number of layers is also fixed. RNN accepts sequences of vectors, both in the input and in the output. The output vector will depend not only on the last input, but also on the entire history of inputs. This means that the RNN has one or more internal states that get updated every time a new input arrives. The RNN accepts x as input vector and gives y as output vector. The internal states are represented by an hidden vector h . The RNN parameters are W_{hh} , W_{xh} , W_{hy} and they represent the connection matrices between the layers. The state is initialized with the zero vector. The system update is computed as follows:

$$h_t = F(W_{hh}h_{t-1} + W_{xh}x_t) \quad (3.1)$$

$$y_t = F(W_{hy}h_t) \quad (3.2)$$

Where F is the activation function. Note that F is applied termwise and it may represent a different function in each equation. The RNN training consists in finding the matrices that give rise to the desirable behaviour, as measured with some loss function that expresses the preference to what kind of outputs y is expected in response to a specific input sequence x .

3.3.1 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a recurrent neural network well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTM were developed to deal with the *vanishing gradient problem* that can be encountered in traditional RNNs.

The vanishing gradient problem happens with gradient-based learning methods and backpropagation. Sometimes, during the weight update procedure, the gradient will be vanishingly small, preventing the weight from changing value. In the worst case, this may completely stop the NN from further training.

A standard LSTM unit is composed of a *cell*, an *input gate*, an *output gate* and a *forget gate*. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. Figure 3.5 shows the structure

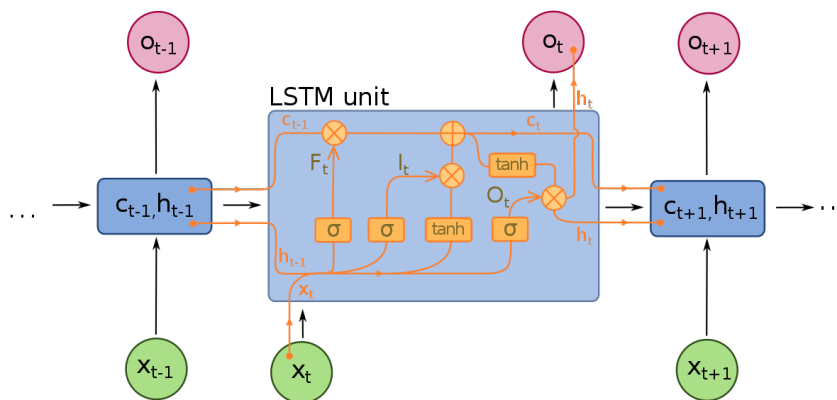


Figure 3.5: LSTM Unit structure

of an LSTM unit with forget gate. The equations that describe the LSTM behaviour are

the following:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (3.3)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (3.4)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (3.5)$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (3.6)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \quad (3.7)$$

$$h_t = o_t \circ \tanh(c_t) \quad (3.8)$$

The initial values are $c_0 = 0$ and $h_0 = 0$, the operator \circ denotes the element-wise product.

The variables are vectors and they represent:

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in \mathbb{R}^h$: forget gates's activation vector
- $i_t \in \mathbb{R}^h$: input/update gate's activation vector
- $o_t \in \mathbb{R}^h$: output gate's activation vector
- $h_t \in \mathbb{R}^h$: hidden state vector, also known as output vector of the LSTM unit
- $\tilde{c}_t \in \mathbb{R}^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h$: weight matrices and bias vector parameters, which need to be learned during training

An RNN using LSTM units can be trained in a supervised fashion on a set of training sequences. Weights are changed according to gradient descent combined with backpropagation through time. The vanishing gradient can happen in standard RNN because $\lim_{n \rightarrow \infty} W^n = 0$ if the spectral radius of W is smaller than 1 [31]. In presence of LSTM units, however, when error values are backpropagated from the output layer, the error remains in the LSTM unit's cell. This continuously feeds error back to each of the LSTM unit's gates, until they learn to cut off the value.

3.4 Experimental Results

For this project, a RNN with LSTM units has been developed, following the work by Chevalier [20]. The network has been trained using the UMAFall public dataset.

We used TensorFlow 2.5 for Python to design and train the network. TensorFlow is an open-source software library for machine learning.

3.4.1 UMAFall

UMAFall contains mobility data extracted from 17 subjects that emulated a set of pre-determined activities of daily life (ADL). The aim of this dataset is the evaluation of fall detection algorithms [21].

For the creation of the dataset, an architecture has been developed in which a smartphone interacts with four wearable sensors located in four different positions of the body of the experimental users.

The sensing units were implemented on four SimpleLink SensorTag units from Texas Instruments. These Sensortags mount the same CC2650 MCU that was chosen for the eSteps final device.

The smartphone acts as data sink for the other devices. A BLE star topology of 5 nodes is deployed, in which the smartphone acts as Master.

The sensor data collected by each unit consist of a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer. For each set of these 9 measurements, the smartphone adds a timestamp and the Bluetooth MAC address of the device that sent the sequence.

The data are stored in a CSV (Comma Separated Values) formatted file. For each set of measures, the smartphone will add its own accelerometer measurement to enrich the mobility information.

The SensorTags were attached to the ankle, waist, right wrist and chest. The smartphone was inserted in the right trouser pocket.

17 users, 10 males and 7 females, were asked to perform daily activities. The 8 ADLs that were selected are the following:

- Squatting
- Climbing stairs up

- Climbing stairs down
- Hopping
- Light jogging
- Lying down on a bed and getting up
- Sitting down on a chair and getting up
- Walking at a normal pace

Additionally, three kinds of falls were emulated by the subjects, namely:

- Backward fall
- Forward fall
- Lateral fall

The initial position of the body before each test was standing straight up. Each movement has been replicated at least three times.

The accelerometer embedded in the smartphone was sampled at 200 Hz, while the SensorTags sampled at 20 Hz, to avoid saturation of BLE communications. The selected full-scales for the sensor nodes were: ± 8 g for the accelerometer, ± 256 dps for the gyroscope and ± 4800 μT for the magnetometer.

The final dataset contains 531 files in CSV format. Among these results, 322 are ADLs and 209 represent falls. Every file contains a 15-second set of measurements.

3.4.2 Preprocessing

First of all, the dataset from UMAFall is preprocessed. The dataset is loaded in memory and the time window is cut down to 12 seconds.

For the purpose of this thesis, only data from the ankle sensor are kept. The measurements from the magnetometer are ignored, since our implementation does not concern with magnetometer informations.

3.4.3 Training

The network has a multi-layered architecture. In particular, the network is composed of 6 layers, meaning that we have 4 hidden layers between the input and output layers.

The number of nodes in each layer is defined as follows:

- Input: 6 nodes
- Hidden 1: 512 nodes
- Hidden 2: 256 nodes
- Hidden 3: 128 nodes
- Hidden 4: 64 nodes
- Output: 8 nodes

The input nodes represent the six sensor axes, three for the accelerometer and three for the gyroscope. Each data is represented by a 12-seconds sequence sampled at 20 Hz. This means that each input sequence contains more than 240 data samples. Figure 3.6 represents the transformation of the data along the different layers. Notice that the architecture of the network can be distinguished in two main parts. The first two hidden layers are RNN layers with LSTM units (refer to section 3.5 for further details). LSTM units are implemented to deal with the temporal aspect of the data.

The other two hidden layers are standard fully-connected layers, also known as dense layers. In the second part of the architecture a dropout procedure is conducted at each layer. Recalling the notion explained at the end of section 3.2, dropout shuts down random neurons in the layer to reduce the problem of overfitting, at the advantage of the network's ability to generalize.

The activation function selected for the second part of the hidden network is ReLU. In the end, the output layer gets Softmax as activation function. Softmax, also known as normalized exponential function, is defined as follows.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, \dots, K \quad \text{and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K. \quad (3.9)$$

This means that for each output node, an exponential function is applied on the value. The result is then normalized by dividing by the sum of all the exponentials.

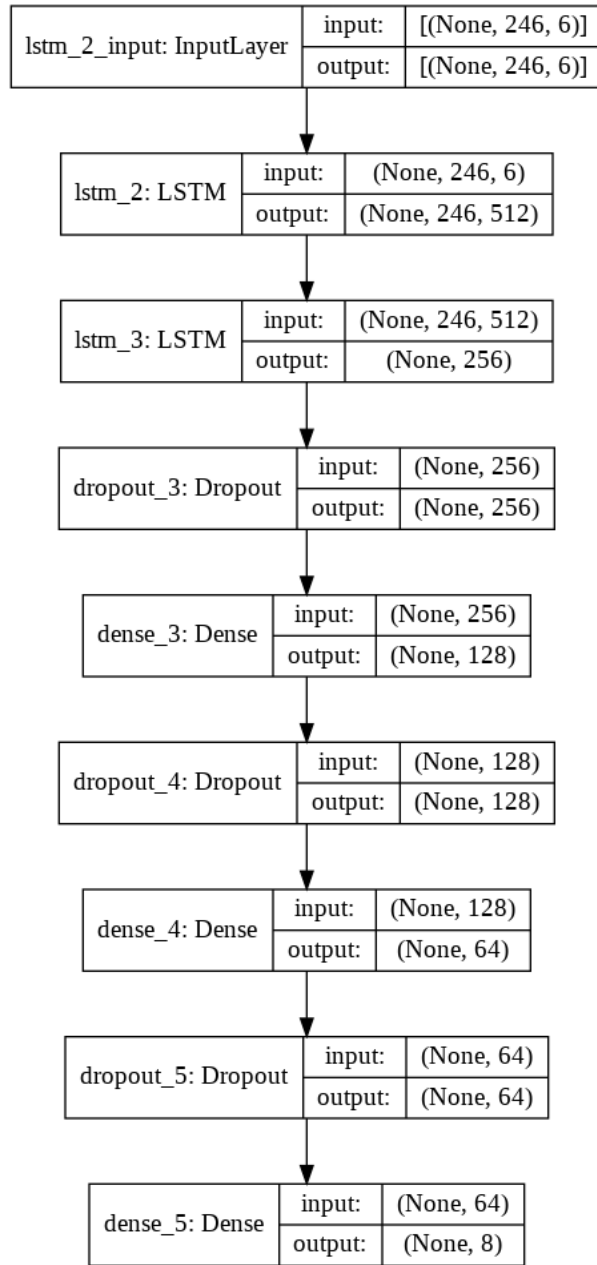


Figure 3.6: Activity Recognition Network Model

The choice for the error function lied in the *Categorical Cross-Entropy* loss function. This function can be written as:

$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i \quad (3.10)$$

where \hat{y}_i is the i -th scalar value in the model output, y_i is the corresponding target value, and output size is the number of scalar values in the model output.

The data used for the training of the network was split into three groups:

- Training Set: 63%
- Validation Set: 7%
- Test Set: 30%

The training set is additionally divided into 10 batches that are sequentially trained and validated. The training is cycled for 20 epochs.

Figure 3.7 shows the evolution of the loss and the accuracy of the network during the training iterations. *Accuracy* gives a direct percentage on the total number of correctly predicted labels.

To further prevent overfitting, the *early stopping* method is implemented: training is stopped if accuravy doesn't improve after 3 epochs.

The results of the training can be represented as a confusion matrix (figure 3.8, in which the labels predicted in the test phase are plotted with respect to the actual labels provided by the dataset. Looking at a single label, additional metrics to evaluate the goodness of the network can be defined. In this study the considered metrics are the following:

- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $f1 = 2 \times \frac{Precision \times Recall}{Precision+Recall}$

With:

- TP: True Positive

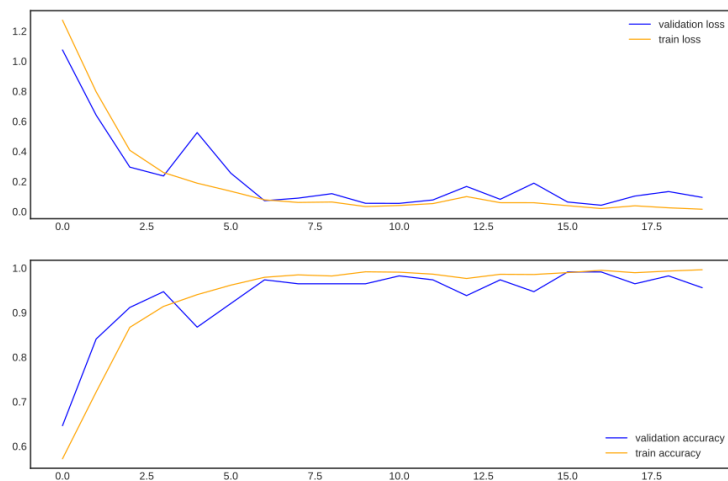


Figure 3.7: Loss and Accuracy plots

Predicted

		Fall	GoDownstairs	GoUpstairs	Hopping	Jogging	LyingDown	sitting	Walking
Actual	Fall	185	0	0	0	0	4	0	0
	GoDownstairs	0	21	0	0	0	0	1	0
	GoUpstairs	2	0	20	0	0	0	0	0
	Hopping	0	0	0	50	0	0	0	0
	Jogging	0	0	0	0	24	0	0	0
	LyingDown	2	0	1	0	0	51	0	0
	sitting	0	0	0	0	0	2	59	0
	Walking	0	0	0	0	0	0	0	62

Figure 3.8: Confusion Matrix

	Precision	Recall	f1_score
Fall	0,97884	0,97884	0,97884
GoDownstairs	0,75862	1	0,86275
GoUpstairs	1	1	1
Hopping	1	1	1
Jogging	1	1	1
LyingDown	0,96	0,88889	0,92308
Sitting	0,96825	1	0,98387
Walking	1	0,91935	0,95798

Figure 3.9: Test Scores on UMAFall dataset

- FP: False Positive
- FN: False Negative

Precision represent the percentage of correct identifications among the total set of positive identifications. It is usually taken into account when the cost of False Positive is high.

Recall, on the other hand, represent the percentage of correct identifications among the total set of actual positives. It is selected when there is a high cost associated with False Negative.

F1 is a function of Precision and Recall. It is a better measure to use if we need to seek a balance between the other two metrics and there is an uneven class distribution.

Figure 3.9 shows the metrics obtained for each ADL.

3.4.4 Prediction

Once the network is trained, it can predict ADLs on the sensor data collected by the eSteps devices according to the procedures depicted in Part 1 of this thesis.

For the purpose of this analysis, results from 6-Minute Walk Tests are considered.

The inertial data, initially sampled at 104 Hz, are downsampled to 20 Hz to be consistent with the UMAFall dataset. Then, the 6 minutes windows are split into batches of 12 seconds.

Prediction has been evaluated with two batches of walk patterns, namely the control set, which contains patterns from healthy subjects walking at a normal pace, and the wobbly set, which contains patterns from healthy subjects walking in a slow and wobbly manner.

The results showed a perfect prediction of normal walks (100% accuracy), while the wobbly set prediction was unsuccessful, scoring a 40% accuracy on the right foot device and 0% on the left one.

This results was expected, since there was not the opportunity to train the network on actual data extracted from MS patients.

Chapter 4

Walking Detection

4.1 Overview

Gait is a complex sensorimotor activity that involves not only the spatial and temporal coordination of the lower limbs but also the coordination of the trunk and arms as well as the dynamic balance. With the development of MEMS (Micro-Electro-Mechanical System) techniques and wireless communications, motion related applications in smart-phones and embedded devices have been increasingly studied as an indoor alternative to GPS (Global Positioning System) tracking and positioning service. In the activity recognition domain, additionally, walking detection and step counting are helpful to identify the physical conditions of patients [34, 35, 37].

4.2 Background

The existing studies can be categorized into time domain approaches, frequency domain approaches and feature clustering approaches [37].

Time domain approaches include thresholding, peak detection, zero-crossing counting, autocorrelation, etc.

The thresholding approach counts steps by judging whether sensory data satisfy some predefined thresholds that differ according to various devices held by different users at different positions. Though this is the simplest time method, it is very difficult to select the optimal threshold for all the cases.

The peak detection approach estimates steps based on the number of peaks given a sequence of sensory data. Though it doesn't rely on thresholds, it suffers from interference peaks due to environmental noises and occasional disturbances.

Similarly, the zero-crossing counting approach counts steps by detecting the number of zero points in sensory data, which is susceptible to disturbances.

The autocorrelation approach detects cyclic periods directly in the time domain by evaluating the autocorrelation. This method shows good accuracy, but it suffers from high computational costs for transforming reference systems.

The frequency domain approaches focus on the frequency content of successive windows of measurements based on short-term Fourier transform (STFT), Fast Fourier Transform (FFT), and continuous/discrete wavelet transforms (CWT/DWT). They can generally achieve high accuracy, but suffer from either resolution issues or computational overheads.

The feature clustering approaches employ machine learning algorithms, e.g., Hidden Markov models (HMMs), KMeans clustering, etc., in order to classify activities based on both time domain and frequency domain feature extracted from sensory data.

4.3 Gait Measures

Sensor data processing allows the segmentation of walking windows. In particular, looking for local maxima in the angular velocity for a specific foot allows the detection of the initial floor contacts, defined as Heel-Strikes (HSs), and of the final floor contacts, defined Toe-Offs (TOs).

According to [35], the sensor data can be processed into 15 gait measures, grouped in three domains: *rhythm*, *variability*, and *balance and coordination*.

The rhythm measures are the following:

- **Stride duration:** time between two consecutive HSs of the same foot
- **Step duration:** time between HS of one foot and HS of the opposite foot
- **Stance duration:** time in which the reference limb is in contact with the floor
- **Swing duration:** time in which the reference limb is not in contact with the floor

- **Single support duration:** time in which only one foot is in contact with the floor
- **Double support duration:** time in which both feet are in contact with the floor

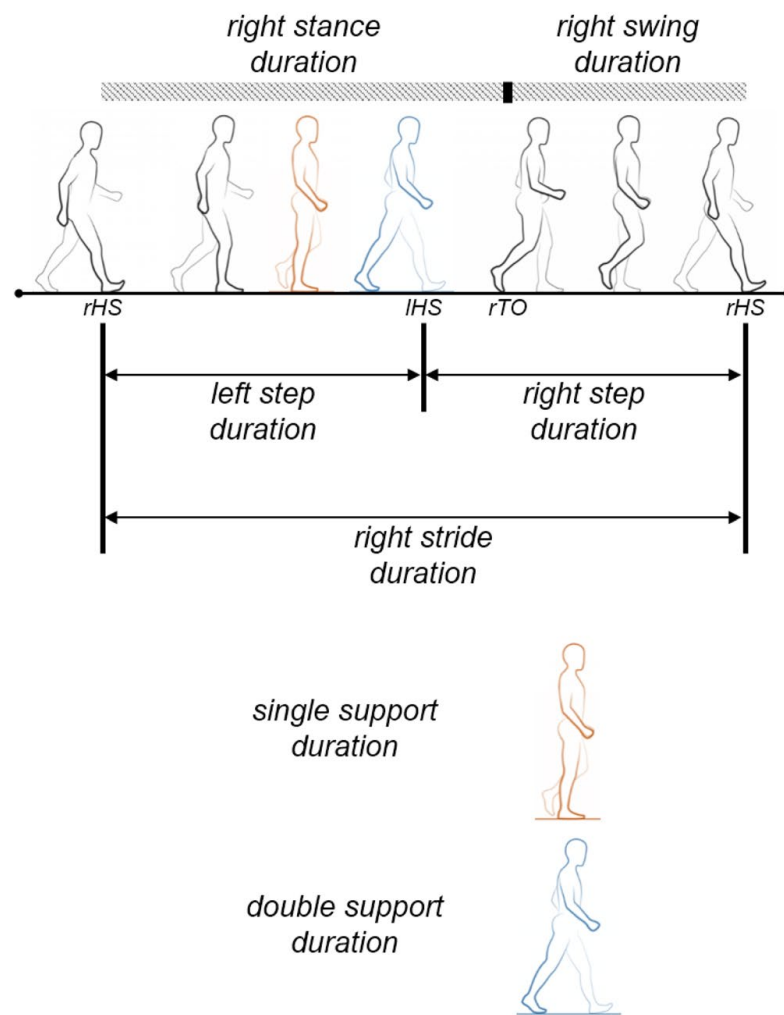


Figure 4.1: Gait Measures: Rhythm Domain

Figure 4.1 gives a graphical representation to the rhythm measures. Variability is computed on stride, step, stance and swing durations.

The balance and coordination measure are the following:

- **Intensity:** it can be computed as the root mean square (RMS) of the acceleration modulus. It can be interpreted as a measure of dynamic balance.
- **Jerk:** it can be computed as the RMS of the first derivative of the acceleration modulus. Lower values reflect a higher ability to effectively pre-plan motor strategies, resulting in a reduced likelihood to fall.
- **Step regularity:** it is the first peak value in the normalized autocorrelation function computed from the acceleration modulus. It quantifies the correlation between consecutive steps.
- **Stride regularity:** it is the second peak value in the normalized autocorrelation function computed from the acceleration modulus. It quantifies the correlation between consecutive strides.
- **Symmetry:** it is a measure of the correlation between left and right steps and it is computed as

$$Symmetry = 1 - \frac{|Strideregularity - Stepregularity|}{mean(Strideregularity, Stepregularity)} \quad (4.1)$$

When compared to healthy controls, gait measurements on subjects with MS reports a less rhythmic gait pattern [35], resulting in longer times to take any of the gait related actions. Intensity and jerk are usually lower in values with respect to healthy controls, meaning that the subjects with MS are able to stabilize their balance with a smoother walking pattern even with an impairment of lower limbs.

4.4 Experimental Results

Gait analysis has been conducted on walking tests taken by the eSteps devices.

The subject is required to walkback and forth at a normal pace for 6-minutes on a straight 10 m track, changing direction by turning always on the same side.

The test allows for pauses during the 6 minutes track, so the walking pattern is expected to be discontinuous.

The actual analysis is done on the patterns between turns and pauses.

4.4.1 Turns and Pauses Detection

Data from the walking test are filtered with a 10 Hz low-pass filter.

Turns are detected using the accelerometer module computed from the three axes. The algorithm implements a 2-seconds sliding window. If the max value of the module inside the window is less than 10% of the max of the total test, that windows is identified as a pause and removed from the set.

Turns are detected using the gyroscope value on the z axis. First the value is integrated to obtain degrees information. Once again, the 2-seconds sliding window is applied. Inside the window, the difference between the last value and the first is computed. If the difference is bigger 160 degrees, a potential turning point is detected, therefore the algorithm iteratively decreases the window size until the difference between the first and last values reaches 120 degrees. The data between the last identified points is identified as a turn and removed from the set.

4.4.2 Gait Analysis

After the removals, the 6-minutes walking data is reduced to a batch of short walking patterns.

In order to compute the different parameters identified in [35], HSs and TOs peaks have to be detected. Figure 4.2 shows the peak detection procedure. Local maxima detection is applied to locate the main contact points of the foot during the walking activity. In these experiments, peaks are detected in the accelerometer medio-lateral axis.

Experimental tests from literature shows that HS and TO always have different peak values, HS being the lowest and TO being the highest. For peak detection, a simple thresholding is implemented. The threshold is initially set as 40% of the maximum value. Then it is iteratively updated until the following conditions are verified:

- HSs peaks are under the threshold
- TOs peaks are over the threshold
- HS and TO periodically alternate during the walking activity, meaning that there cannot be two consecutive HSs or TOs

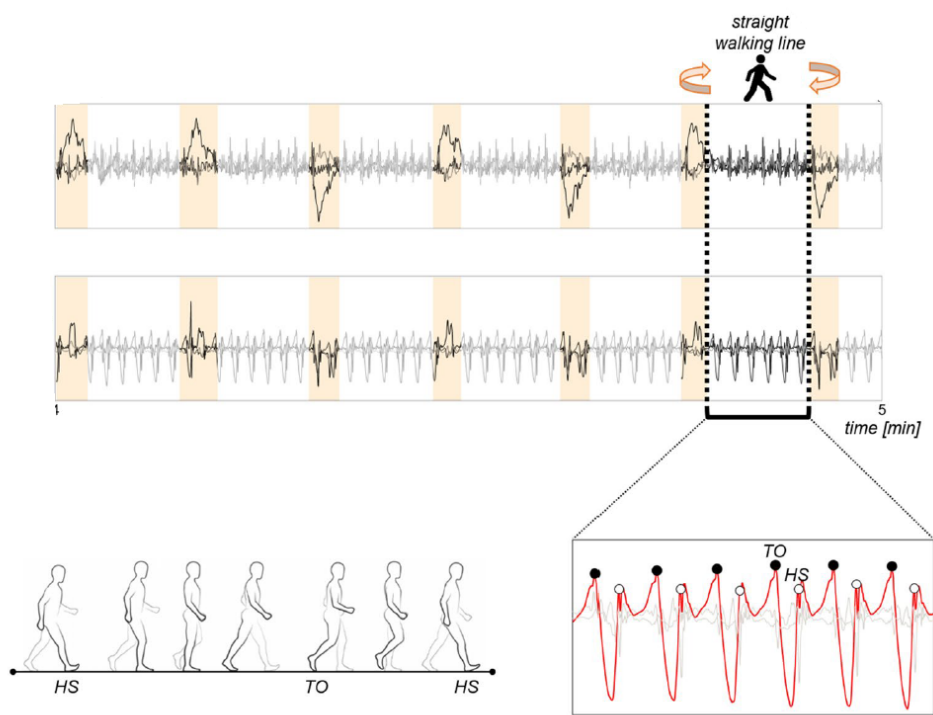


Figure 4.2: Gait Analysis

This detection is done for both feet data, then they are put together on a single timeline. Here, the we check that the sequence of contact points is consistent with a standard walking pattern.

Once TOs and HSs are detected, all the rhythm and balance metrics are computed according to section 4.3.

All the experimental computations show consistency with the control set measurements taken in the control set in [35].

Future developments within clinical trials will be used to evaluate the difference in the gait measurements among subjects affected by MS.

Conclusions

This thesis followed the development of a low power medical device for MS monitoring. Standard clinical tests usually take place in an environment which is unfamiliar to the patient. Creating a device that allows the patient to undergo those same tests in a friendly environment, without the need of leaving the house, has been a necessary task in the field of health monitoring, especially after the recent pandemic outbreak.

The project then evolved from a clinical test emulator to a continuous monitoring of the physical activities of the patient, to extract a bigger dataset for MS related studies. This required an additional study in the optimization of power consumption, to ease the user experience of the patient that would otherwise be constantly reminded of the device's presence by having to frequently charge it back. The new and promising world of free-living gait analysis pushed the student to evaluate new approaches to this optimization problem, implementing sleep routines and gestures to make the device as much energy efficient as possible.

Standard data analysis approaches have been enriched with the aid of RNNs. The experiments showed the possibility for an appropriately trained NN to extract walking patterns from a constant flow of sensory data regarding various and a priori unknown activities. The idea was to free the patient from the idea of having to take daily tests. Once again, the device has been developed to be an invisible monitor for the patient mobility.

Experiments showed promising results in activity recognition and gait analysis on healthy subjects. However, the lack of data from actual MS subjects deeply affected the overall analysis, in particular in the activity recognition component.

Future development will be the re-training of the network with tests directly taken with the eSteps devices, without downsampling the signals. The network should be able to correctly identify walking patterns even within disease-related walks.

The gait analysis will also be improved to confirm the higher variability in mobility measures taken from MS patients.

Bibliography

- [1] *Nokia's Wibree and the Wireless Zoo*, The Future of Things, 2006.
- [2] S. Bluetooth, *Bluetooth specifications version 4.2*, Bluetooth SIG, 2014.
- [3] K. Mikhalov, *Simulation of network-level performance for bluetooth low energy*, in *Personal, Indoor, and Mobile Radio Communication (PIMRC), 2014 IEEE 25th Annual International Symposium on*. IEEE, 2014, pp. 1259-1263.
- [4] M. Marawaha, P. Jha, R. Razdan, J. Dukes, *Performance Evaluation of Bluetooth Low Energy Communication*, in *Journal of Information Sciences and Computing Technologies, ISSN 2394-9066*. JISCT, Volume 7, Issue 2, 2018, pp. 718-725.
- [5] F. S. De Sousa, C. EE. Capovilla, I. R. S. Casella, *Analysis of Bluetooth Low Energy Technology in Indoor Environments*, in *Consumer Electronics, 2016 IEEE International Symposium on*. IEEE, 2016.
- [6] S. H. Gerez, *Implementation of Digital Signal Processing: Some Background on GFSK Modulation*, University of Twente, Department of Electrical Engineering, 2016.
- [7] Silicon Labs, *Optimizing Current Consumption in Bluetooth Low Energy Devices*, docs.silabs.com.
- [8] R. Heydon, *An Introduction to Bluetooth Low Energy*, Qualcomm Technologies International, Ltd.
- [9] Argenox Technologies, *BLE Advertising Primer*, argenox.com/library/bluetooth-low-energy.

- [10] Texas Instruments, *BLE-Stack User's Guide*.
- [11] N.Siegel, *Using TI Certified Bluetooth Low Energy Module (CC2650MODA) as Single-Chip Wireless MCU*, in *Application Report SWRA534A*, Texas Instruments, 2017.
- [12] J. Lindh, C. Lee, M. Hernes, S. Johnsrud, *Measuring CC13xx and CC26xx Current Consumption*, in *Application Report SWRA478D*, Texas Instruments, 2019.
- [13] Texas Instruments, *CC2650 SimpleLink Multistandard Wireless MCU, Literature Number: SWRS158B*, Texas Instruments, 2016.
- [14] J. Lindh, *Bluetooth Low Energy Beacons*, in *Application Report SWRA475A*, Texas Instruments, 2016.
- [15] Texas Instruments, *CC26x0 SimpleLink Bluetooth low energy Software Stack 2.2.x - Developer's Guide, Literature Number: SWRU393E*, Texas Instruments, 2018.
- [16] STMicroelectronics, *iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope*, LSM6DSL Datasheet.
- [17] STMicroelectronics, *LSM6DSL: always-on 3D accelerometer and 3D gyroscope*, LSM6DSL Application Note.
- [18] STMicroelectronics, *iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope*, LSM6DSOX Datasheet.
- [19] W. Pitts, W. McCulloch, *A logical calculus of the ideas immanent in nervous activity* 1943.
- [20] G. Chevalier, *LSTMs for Human Activity Recognition*, 2016.
- [21] E. Casilari et al., *UMAFall: A Multisensor Dataset for the Research on Automatic Fall Detection*, in *Procedia Computer Science 110(2017):32-39*.
- [22] A. Karpathy, *The Unreasonable Effectiveness of Recurrent Neural Networks*, 2015, <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>.

- [23] S. Dupond, *A Thorough Review on the Current Advance of Neural Network Structures*, in *Annual Reviews in Control*, 2019, 14:200-230).
- [24] J. Brownlee, *How to Choose an Activation Function for Deep Learning*, 2021, <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>.
- [25] T. Mitchell, *Machine Learning*, New York: McGraw Hill, 1997, ISBN 0-07-042807-7.
- [26] F. Chollet, *Deep Learning with Python*, Manning, 2018.
- [27] F. Rosenblatt, *The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain*, 1958.
- [28] S. J. Russel, P. Norvig, *Artificial intelligence. A modern approach volume 1*, Prentice Hall, 2010.
- [29] S. J. Russel, P. Norvig, *Artificial intelligence. A modern approach volume 2*, Prentice Hall, 2005.
- [30] R. J. Williams, G. E. Hinton, D. E. Rumelhart, *Learning Representations by Back-propagating Errors*, in *Nature*, 323: 533-536, October 1986.
- [31] S. Hochreiter et al., *Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies*, in *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press, 2001.
- [32] M. Viceconti et al., *Toward a Regulatory Qualification of Real-World Mobility Performance Biomarkers in Parkinson's Patients Using Digital Mobility Outcomes*, in *Sensors*, 2020, 20, 5920. www.mdpi.com/journal/sensors.
- [33] M. Ulrich et al., *Detection of Gait from Continuous Inertial Sensor Data Using Harmonic Frequencies*, in *IEEE Journal of Biomedical and Health Informatics*, vol.24, no.7, July 2020.
- [34] R. Kaur, Z. Chen, R. Motl, M. E. Hernandez, R. Sowers, *Predicting Multiple Sclerosis from Gait Dynamics Using an Instrumented Treadmill - A Machine Learning Approach*, in DOI 10.1109/TBME.2020.3048142, *IEEE Transactions on Biomedical Engineering*.

- [35] L. Angelini, W. Hodgkinson, C. Smith, J. Moorman Dodd, B. Sharrack, C. Mazz, D. Paling, *Wearable sensors can reliably quantify gait alterations associated with disability in people with progressive multiple sclerosis in a clinical setting*, *Journal of Neurology* (2020) 267:2897-2909.
- [36] J. F. Kurtzke, *Rating Neurologic Impairment in Multiple Sclerosis: an Expanded Disability Status Scale (EDSS)*, in *Neurology*, 1983, 33.1444-1452.
- [37] X. Kang, B. Huang, G. Qi, *A Novel Walking Detection and Step Counting Algorithm Using Unconstrained Smartphones*, in *Sensors*, 2018, 18, 297. www.mdpi.com/journal/sensors.