Instituto Superior de Engenharia
Politécnico de Coimbra

DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA

Development of an IoT-based Health Monitoring System

Trabalho de Projeto para a obtenção do grau de Mestre em Engenharia Eletrotécnica

Especialização em Automação e Comunicações em Sistemas Industriais

Autora

Giada Profiti

Orientadores

Professor Doutor Fernando José Pimentel Lopes

Professor Coordenador

Professor Doutor Victor Daniel Neto dos Santos

Professor Coordenador

INSTITUTO POLITÉCNICO DE COIMBRA

INSTITUTO SUPERIOR DE ENGENHARIA DE COIMBRA

Coimbra, Dezembro de 2024

ACKNOWLEDGEMENTS

In the first place, I would like to express my deepest appreciation to my advisors, Professor Doctor Fernando José Pimentel Lopes and Professor Doctor Victor Daniel Neto dos Santos for all their competence, total availability, concern and for all the fundamental teachings during all the years of my academic journey and during the development of this final project.

I am extremely grateful to my family for always believing in me throughout my life and for the unconditional support shown despite the distance.

I am deeply grateful to my boyfriend Eugenio, my companion on adventures, for his presence and tenderness.

Special thanks to all those who have always rejoiced in my achievements.

Portugal and, in particular, Coimbra will forever remain in my heart.

RESUMO

Nas próximas décadas prevê-se um aumento significativo da idade média da população dos países desenvolvidos. Assim sendo, o número de pessoas idosas que necessitam de assistência médica e de monitorização remota da saúde também aumentará. Ao proporcionar aos idosos uma monitorização remota da saúde baseada na Healthcare Internet of Things (H-IoT), estes podem ser sempre seguidos enquanto mantêm um estilo de vida independente.

O principal objetivo deste projecto é desenvolver e implementar um sistema eletrónico de monitorização de saúde, baseado numa rede de sensores corporais, para utentes idosos, que não se encontram em estado crítico, mas que necessitem de uma monitorização contínua ou periódica. Este sistema monitoriza a temperatura corporal, a frequência cardíaca e a saturação periférica de oxigénio dos utentes, permitindo também a deteção de ocorrência de quedas.

O sistema desenvolvido utiliza três sensores não invasivos para recolher os dados: um termístor para medir a temperatura corporal, um acelerómetro/giroscópio para detetar os movimentos do paciente e perceber se o idoso caiu e um oxímetro de pulso para medir a frequência cardíaca e analisar o nível de saturação de oxigénio no sangue do paciente. O sistema implementado possui também um botão de emergência, que sinaliza eventos de emergência, o qual pode torna-se em um botão de confirmação, que pode interagir com mensagens apresentadas no ecrã do sistema implementado. Os dados recolhidos são armazenados numa base de dados local e num servidor externo, para que o médico possa analisar remotamente o estado de saúde do paciente.

O sistema implementado incluí um Raspberry Pi 4 Modelo B; um Arduino Uno R4 Minima que contribui para a recolha de dados e um transceiver TTGO LoRa32 Modelo T3 baseado na tecnologia LoRaWAN. Os dados recolhidos são enviados para a plataforma Akenza, utilizando um de dois caminhos alternativos. Se o dispositivo estiver dentro do alcance da rede Wi-Fi local, os dados serão enviados via MQTT para o servidor Akenza. Por outro lado, se o Wi-Fi estiver fora de alcance, os dados serão enviados através da rede LoRaWAN para o servidor The Things Network (TTN). Neste segundo caso, através de uma integração, os dados serão redirecionados para o servidor Akenza. O Akenza Dashboard visualiza os dados recolhidos e recebidos e, se os valores ultrapassarem determinados limiares, o Akenza envia mensagens de correio eletrónico de aviso ao médico e/ou aos familiares do paciente.

Palavras-Chave: Pessoas idosas; Monitorização da saúde; Sensores corporais; IoT; Deteção de quedas.

ABSTRACT

In the coming decades, the average age of the population in the developed countries is expected to rise significantly. Therefore, the number of elderly people requiring medical assistance and remote health monitoring will also increase. By offering remote health monitoring based on the Healthcare Internet of Things (H-IoT) to the elderly, they can always be observed while maintaining an independent lifestyle.

The main objective of this project is to develop and implement an electronic health monitoring system, based on a body sensor network, for elderly users who are not in critical condition, but need continuous or periodic monitoring. This system monitors the body temperature, the heart rate and the peripheral oxygen saturation, also allowing the detection of falls.

The developed system uses three non-invasive sensors to collect the data: a thermistor to measure the body temperature, an accelerometer/gyroscope to detect the patient's movements and to accordingly understand if the elderly person has fallen, and a pulse oximeter to measure the heart rate and analyse the level of oxygen saturation in the patient's blood. The implemented system also owns an emergency button, able to report emergency events, which can become a confirmation button to interact with the messages displayed on the screen of the implemented system. The collected data is stored in a local database and in an external server, so that the doctor can remotely analyse the patient's health condition.

The implemented system includes a Raspberry Pi 4 Model B, an Arduino Uno R4 Minima, which contributes to the data collection, and a TTGO LoRa32 Model T3 transceiver based on the LoRaWAN technology. The collected data is sent to the Akenza platform using one out of two alternative paths. If the device is within the range of the local Wi-Fi network, the data is sent via MQTT to the Akenza server. On the other hand, if the Wi-Fi is out of range, the data will be sent via the LoRaWAN network to The Things Network (TTN) server. In this second case, through an integration, the data will be redirected to the Akenza server. The Akenza Dashboard visualises the collected and received data and, if the values exceed certain thresholds, Akenza sends warning emails to the doctor and/or to the patient's relatives.

Keywords: Elderly people; Health monitoring; Body Sensors; IoT; Fall detection.

TABLE OF CONTENTS

Acknowledgements i
Resumo iii
Abstract v
Table of contents vii
Table of figures xi
Tables xix
List of acronyms xxi
List of symbols xxiii
1 Introduction 1
1.1 Older People Living Alone (OPLA) 1
1.1.1 Reasons behind the different household sizes 2
1.1.2 Trend 3
1.1.3 Consequences of COVID-19 4
1.2 Scope and motivation 4
1.2.1 Body Temperature 5
1.2.2 Fall Detection 5
1.2.3 Heart Rate 5
1.2.4 SpO2 6
1.3 Objectives 6
1.4 Document structure 8
2 State of the Art 11
2.1 Wearable devices 11
2.1.1 Historical perspective on wearables 11
2.1.2 Medical and Close-to-Medical area 12
2.1.3 Apple Watch Series 10 15
2.2 Internet of things (IoT) 16
2.2.1 Architecture of the IoT 17
3 Architecture of the Planned System 19
3.1 The things or devices layer 19
3.1.1 Sensors 19

3.1.2 Dual-purpose Button 20
3.1.3 Edge computing 20
3.2 The connectivity layer 20
3.3 The application layer 21
4 Things or Devices Layer 23
4.1 NTC Thermistor for Body Temperature Measurement 23
4.1.1 Body Temperature 23
4.1.2 NTC Thermistor 26
4.1.3 MA300 29
4.2 Analog-to-Digital Converter ADC 30
4.2.1 ADC Resolution 30
4.2.2 Successive-approximation ADC 30
4.2.3 MCP3008 32
4.3 Accelerometer and Gyroscope 34
4.3.1 Accelerometer 35
4.3.2 Gyroscope 40
4.3.3 MPU-6050 44
4.3.4 Registers of the MPU-6050 50
4.4 Pulse Oximeter and Heart Rate Detector 54
4.4.1 Heart and blood circulation 54
4.4.2 Analysis of the absorbed light 57
4.4.3 Heart Rate 58
4.4.4 Oxygen Saturation 59
4.4.5 MAX30100 61
4.4.6 Registers of the MAX30100 63
4.5 Dual-Purpose Button 64
4.6 Edge Computing 66
4.6.1 Raspberry Pi 66
4.6.2 LILYGO TTGO LoRa32 68
4.6.3 Arduino Uno R4 Minima 70
5 Serial Communication Protocols 75
5.1 Inter-Integrated Circuit serial protocol (I^2C) 75
5.1.1 How the I^2C communication works 77

5.1.2 I²C Communication between the Arduino and the MAX30100 78
5.2 Serial Peripheral Interface (SPI) 80
5.2.1 How the SPI communication works 81
5.2.2 SPI Communication between the Raspberry and the ADC 83
5.3 Universal Asynchronous Receiver-Transmitter (UART) over Universal Serial Bus (USB) 84
5.3.1 USB Cable and Connectors 84
5.3.2 Universal Asynchronous Receiver-Transmitter (UART) 86
6 Connectivity and Application Layers 89
6.1 ISO OSI Model and TCP/IP Model 89
6.2 Wi-Fi 91
6.2.1 Wi-Fi Architecture 92
6.3 MQTT 93
6.4 LoRa 95
6.4.1 Chirp Spread Spectrum 96
6.4.2 LoRa Message Format 98
6.5 LoRaWAN 99
6.5.1 LoRaWAN Message Format 103
6.5.2 OTA Activation of the End-Device 104
6.6 The Things Network 105
6.6.1 DevEUI, DevAddr 106
6.6.2 Transport Layer Security (TLS) Protocol 106
6.6.3 Webhook Integration between TTN and Akenza 107
6.7 Akenza 107
7 Implemented System 109
7.1 Sensors and dual-purpose button connection 109
7.1.1 Connection of the MA300 to the Raspberry using the MCP3008 109
7.1.2 Connection of the MPU-6050 to the Raspberry 111
7.1.3 Connection of the MAX30100 to the Arduino 112
7.1.4 Connection of the Button to the Raspberry 112
7.2 Initial configuration of the Raspberry 113
7.3 Getting started with TTN version 3 116
7.4 Cayenne LPP Formatter Type 123
7.4.1 The Cayenne LPP in TTN 124

7.5 Getting started with Akenza 125
7.5.1 Akenza Dashboard 131
7.6 Integration between TTN and Akenza 133
7.7 Akenza Rules 141
7.8 Description of the Software running on the Arduino 144
7.9 Description of the Software running on the Raspberry 145
7.9.1 Software Module for the Data Collection 145
7.9.2 Software Module for the Fall Detection 149
7.10 Description of the Software running on the LoRa32 154
8 Experimental Results from the Developed Prototype 159
8.1 Test of the Software running on the Arduino 159
8.2 Test of the Software running on the LoRa32 161
8.3 Test of the Software running on the Raspberry 164
8.3.1 Test of the Software Module for the Fall Detection 164
8.3.2 Test of the Software Module for the Data Collection 171
9 Field Test Results Analysis from Platform Dashboard 173
10 Conclusions and Future Work 183
Bibliography 185
Appendix A – Wearable devices History 193
Appendix B – NTC datasheet 205

TABLE OF FIGURES

Figure 1 – Percentage of people aged 60 and older in each household type [2]. 1
Figure 2 – Average number of residents per household [2]..... 2
Figure 3 – Single-person households per ranges of ages in years 2018, 2028 and 2038 [3]..... 3
Figure 4 – Elderly loneliness during COVID-19 [4]..... 4
Figure 5 – Essential Architecture of the System. 7
Figure 6 – Evolution of wearables from the 13th century to nowadays [9]..... 12
Figure 7 – Chronolife smart vest..... 13
Figure 8 – SYNCHRONY 2 cochlear implants. 13
Figure 9 – Neofect's powered glove. 14
Figure 10 – Asynchronous Coded Electronic Skin. 14
Figure 11 – D-mine Pump. 14
Figure 12 – Apple Watch Series 10. 15
Figure 13 – Detection of a hard fall in Apple Watch Series 10. 16
Figure 14 – Three-layered IoT Architecture. Adapted from [12]..... 17
Figure 15 – Architecture of the Implemented System..... 19
Figure 16 – Temperatures in the human body [5]..... 24
Figure 17 – Iron (III) oxide Fe_2O_3. 27
Figure 18 – Iron (III) oxide Fe_2O_3 after the replacement. 27
Figure 19 – Example of 4-bit ADC conversion. 30
Figure 20 – Successive-Approximation ADC [16]. 31
Figure 21 – ADC conversion algorithm. Adapted from [16]..... 32
Figure 22 – MCP3008 [17]. 32
Figure 23 – MCP3008 Functional Block Diagram [17]. 33
Figure 24 – MCP3008 Pins [17]..... 33
Figure 25 – Structure of the Accelerometer. Adapted from [19]. 36
Figure 26 – Acceleration on the accelerometer along its sensitive axis. Adapted from [19]..... 36
Figure 27 – Analysis of the capacitors in the accelerometer (part 1). 37

Figure 28 – Analysis of the capacitors in the accelerometer (part 2). 38
Figure 29 – Direction of the Coriolis force. Adapted from [22]. 40
Figure 30 – Coriolis force on one moving mass [21]. 41
Figure 31 – Coriolis force on two oscillating masses [21]. 41
Figure 32 – The four proof masses of the MEMS gyroscope [21]. 42
Figure 33 – Roll mode [21]. 42
Figure 34 – Pitch mode [21]. 43
Figure 35 – Yaw mode [21]. 43
Figure 36 – MPU-6050 Sensor Module [23]. 44
Figure 37 – MPU-6050 Functional Block Diagram [23]. 44
Figure 38 – MPU-6050 3-Axis Accelerometer [23]. 45
Figure 39 – MPU-6050 3-Axis Gyroscope [23]. 46
Figure 40 – MPU-6050 Pins [24]. 49
Figure 41 – Register 27 – Gyroscope Configuration GYRO_CONFIG [25]. 50
Figure 42 – Register 28 – Accelerometer Configuration ACCEL_CONFIG [25]. 51
Figure 43 – Register 56 – Interrupt Enable INT_ENABLE [25]. 51
Figure 44 – Register 58 – Interrupt Status INT_STATUS (Read-Only) [25]. 52
Figure 45 – Registers 59 to 64 – Accelerometer Measurements [25]. 52
Figure 46 – Registers 65 and 66 – Temperature Measurement [25]. 53
Figure 47 – Registers 67 to 72 – Gyroscope Measurements [25]. 53
Figure 48 – Heart structure [28]. 54
Figure 49 – Systemic and pulmonary circulations [29]. 55
Figure 50 – Cardiac cycle. Adapted from [30]. 57
Figure 51 – Finger on the sensor. Adapted from [31]. 57
Figure 52 – AC and DC components of the absorbed light in a finger [32]. 58
Figure 53 – Heart Rate measurement. Adapted from [33]. 59
Figure 54 – Absorption of Haemoglobin as a function of Wavelength. Adapted from [34]. 60
Figure 55 – Relation of R and SpO2 [35]. 61
Figure 56 – Graphical Representation of the FIFO Data Register [31]. 61
Figure 57 – MAX30100 Functional Block Diagram [31]. 62
Figure 58 – MAX30100 Pins. 63

Figure 59 – Interrupt Status Register [31]. 63
Figure 60 – Interrupt Enable Register [31]. 64
Figure 61 – Connection of the Button. 65
Figure 62 – Email due to Akenza Rule. 65
Figure 63 – Raspberry Pi 4 Model B [37]. 66
Figure 64 – Raspberry Pins [40] 67
Figure 65 – LILYGO TTGO LoRa32. 68
Figure 66 – Rear and frontal view of the LoRa32 T3 model. 69
Figure 67 – Arduino Uno R4 Minima [44]. 71
Figure 68 – Arduino Uno R4 Minima Pins [44]. 71
Figure 69 – Arduino IDE 2.0. Adapted from [46]. 73
Figure 70 – I^2C connection of one Master and one Slave devices [48]. 75
Figure 71 – N-Channel MOSFET symbol and structure. Adapted from [49]. 76
Figure 72 – I^2C communication procedure [51]. 77
Figure 73 – SPI single Master and single Slave interface. Adapted from [52]. 80
Figure 74 – Shift Registers in the SPI communication [52]. 81
Figure 75 – Example of SPI communication with CPOL=0 and CPHA=0 [53]. .. 82
Figure 76 – SPI Communication with the MCP3008 (Mode 0,0) [17]. 83
Figure 77 – USB connectors. Adapted from [54] 85
Figure 78 – USB 2.0 cable. Adapted from [55] 85
Figure 79 – UART frame format. Adapted from [56]. 87
Figure 80 – *ls /dev/tty** while neither the Arduino nor the LoRa32 were connected to the Raspberry. 87
Figure 81 – *ls /dev/tty** while only the Arduino was connected to the Raspberry. .. 88
Figure 82 – *ls /dev/tty** while both the Arduino and the LoRa32 were connected to the Raspberry. 88
Figure 83 – OSI and TCP/IP Models. Adapted from [57]. 89
Figure 84 – MQTT on TCP/IP 93
Figure 85 – MQTT Topic [62] 94
Figure 86 – MQTT Broker. Adapted from [62]. 94
Figure 87 – MQTT bidirectional communication. Adapted from [62]. 95
Figure 88 – Linear up-chirp [67] 96

Figure 89 – Example of LoRa signal with SF=2. 97
Figure 90 – Comparison of LoRa Spreading Factors: SF=7 to SF=12 in the symbol 0 [67] 98
Figure 91 – LoRa Physical layer Message [69]. 98
Figure 92 – LoRa Physical layer Message (Frequency versus Time). Adapted from [70]. 99
Figure 93 – LoRaWAN Star-of-Stars Topology. Adapted from [72]. 100
Figure 94 – LoRaWAN Architecture. Adapted from [73]. 100
Figure 95 – LoRaWAN Gateways in Coimbra. 102
Figure 96 – LoRaWAN Frame [74] 103
Figure 97 – Over-the-Air (OTA) Activation. Adapted from [75]. 104
Figure 98 – The Things Stack. Adapted from [76] 105
Figure 99 – The Voltage Divider circuit for Rntc. 109
Figure 100 – How the Thermistor is connected to the ADC. 110
Figure 101 – Connection of the MCP3008 to the Raspberry. 110
Figure 102 – Connection of the MPU-6050 to the Raspberry. 111
Figure 103 – Connection of the MAX30100 to the Arduino. 112
Figure 104 – Connection of the Button to the Raspberry. 113
Figure 105 – Choosing the Raspberry Pi OS on the Raspberry Pi Imager. 114
Figure 106 – a) PuTTY Configuration. b) Raspberry Command Prompt seen from PuTTY. 115
Figure 107 – The VNC Viewer Software. 115
Figure 108 – a) Arduino IDE, File → Preferences. b) Arduino IDE, Boards Manager. 116
Figure 109 – Arduino IDE, Select board → Select Other Board and Port. 117
Figure 110 – Arduino IDE, Library Manager. 117
Figure 111 – File *lmic_project_config.h*. 118
Figure 112 – TTN, Applications. 118
Figure 113 – TTN, Create application. 119
Figure 114 – TTN, Application overview. 119
Figure 115 – TTN, Register end device (part 1). 120
Figure 116 – TTN, Register end device (part 2). 120
Figure 117 – TTN, Register end device (part 3). 121

Figure 118 – TTN, LoRa32 Device overview. 121
Figure 119 – Arduino IDE, inserting AppEUI, DevEUI and AppKey into the code. 122
Figure 120 – Arduino IDE, writing the correct pins of the specific TTGO LoRa32. 122
Figure 121 – TTN, setting the Cayenne LPP Formatter type. 125
Figure 122 – Akenza, welcome page. 126
Figure 123 – a) Akenza, creating an Organization. b) Akenza, creating a Workspace. 126
Figure 124 – a) Akenza, getting started. b) Akenza, Data Flows. 127
Figure 125 – Akenza, creating a Data Flow. 127
Figure 126 – Akenza, creating the MQTT Data Flow. 128
Figure 127 – Akenza, saving the MQTT Data Flow. 128
Figure 128 – Akenza, creating the Raspberry Device (part 1). 129
Figure 129 – Akenza, creating the Raspberry Device (part 2). 129
Figure 130 – Akenza, generating the Raspberry Device ID. 130
Figure 131 – Akenza, the Raspberry Device created. 130
Figure 132 – Inserting the MQTT details into the Raspberry code. 131
Figure 133 – Akenza, Dashboard homepage. 131
Figure 134 – Akenza, creating the Dashboard (General information). 132
Figure 135 – Akenza, creating the Dashboard (Scope selection). 132
Figure 136 – Akenza, creating the Dashboard (Settings). 133
Figure 137 – TTN, Application ID. 133
Figure 138 – TTN, creating an API key (part 1). 134
Figure 139 – TTN, creating an API key (part 2). 134
Figure 140 – Akenza, creating the Integration with TTN. 135
Figure 141 – Akenza, inserting the details from TTN. 135
Figure 142 – Akenza, naming the Integration with TTN. 136
Figure 143 – Akenza, the created Integration. 136
Figure 144 – TTN, the resulting Integration. 137
Figure 145 – Akenza, Assets. 137
Figure 146 – Akenza, list of Data Flows. 138
Figure 147 – Akenza, creating the LoRaWAN Data Flow. 138

Figure 148 – Akenza, choosing the Device Connector. 139
Figure 149 – Akenza, choosing the Device Type. 139
Figure 150 – Akenza, choosing the Output Connector. 140
Figure 151 – Akenza, updated list of Devices in Assets. 140
Figure 152 – Akenza, assigning the LoRaWAN Data Flow to the LoRa32 device. 141
Figure 153 – Akenza, New Rule. 141
Figure 154 – Akenza, Input Block of the Rule. 142
Figure 155 – Akenza, Logic Block of the Rule. 142
Figure 156 – Akenza, Action Block of the Rule. 143
Figure 157 – Akenza, list of the Rules. 143
Figure 158 – Akenza's email after the condition is met. 143
Figure 159 – Flowchart of the Software that runs on the Arduino. 144
Figure 160 – Flowchart of the Raspberry Software Module for Sensors Data Collection. 148
Figure 161 – Flowchart of the *button_pressed_callback()* function of the Raspberry Sensors Data Collection Software Module. 149
Figure 162 – Modulo of the Acceleration as a function of Time during a Fall [87]. 150
Figure 163 – Behaviour of the Modulo of the Acceleration in different situations. 151
Figure 164 – Flowchart of the Raspberry Software Module for Fall Detection. ..153
Figure 165 – Flowchart of the *setup()* function of the LoRa32 Software. 154
Figure 166 – Flowchart of the *loop()* function of the LoRa32 Software. 154
Figure 167 – Flowchart of the *do_send()* function of the LoRa32 Software. 158
Figure 168 – Various Readings of HR and SpO2 seen on the Serial Monitor of the Arduino IDE. 159
Figure 169 – Experimental Test to plot the Heart Rate versus Time. 160
Figure 170 – Experimental Test to plot the SpO2 versus Time. 161
Figure 171 –LoRa32 waiting for data from the Raspberry. 161
Figure 172 – Joining and Joined events. 162
Figure 173 – Priority to Fall Detection. 162
Figure 174 – Priority to Emergency. 163
Figure 175 – Priority to Message. 163

Figure 176 – Sensors Data processing on the LoRa32. 164
Figure 177 – Device placed in a belt bag. 165
Figure 178 – Device powered by a power bank. 165
Figure 179 – Experimental Plot of the Modulo of the Acceleration in a Fall. 166
Figure 180 – Experimental Plot of the Modulo of the Acceleration in a Walking activity. 167
Figure 181 – Experimental Plot of the Modulo of the Acceleration in a Running activity. 168
Figure 182 – Experimental Plot of the Modulo of the Acceleration in a Jump activity. 169
Figure 183 – Experimental Plot of the Modulo of the Acceleration in a Sitting and Standing up activity. 170
Figure 184 – Experimental Plot of the Modulo of the Acceleration in a Lying down and Standing up activity. 171
Figure 185 – Experimental Test to plot the Temperature versus Time. 172
Figure 186 – Arduino and MAX30100 tied on the hand. 173
Figure 187 – Akenza Dashboard after the Experience. 174
Figure 188 – Sensors Data at 12:19 on Akenza. 174
Figure 189 – Use of the smartwatch to confirm the data collected from the MAX30100. 175
Figure 190 – Emergency sent when in the Wi-Fi range. 175
Figure 191 – Akenza's email of warning. 176
Figure 192 – Body Temperature data sent via LoRaWAN. 176
Figure 193 – Emergencies sent via LoRaWAN. 177
Figure 194 – Akenza's emails of warning. 177
Figure 195 – Emergency automatically sent when the patient is not ok. 178
Figure 196 – Akenza's email of warning. 178
Figure 197 – Heart Rate during Sleep phase. 179
Figure 198 – Akenza Dashboard after the Fall Experience. 180
Figure 199 – Fall detected sent to Akenza. 181
Figure 200 – Akenza's email of warning after the Fall. 181
Figure 201 – Peter Henlein's Pomander watch, The Watch 1505. 193
Figure 202 – Another Pomander watch (1530) created by Peter Henlein. 193
Figure 203 – A functioning Abacus Ring from the Qing Dynasty (1644-1912). 194

Figure 204 – Packset system. 194
Figure 205 – British Military Wristwatch..... 195
Figure 206 – Original British WWII RAF 2nd Pattern Type C Leather Flying Helmet wit. 195
Figure 207 – Sensorama Simulator..... 196
Figure 208 – Computer hidden in a shoe. 196
Figure 209 – TV Glasses. 197
Figure 210 – Pulsar Calculator Watch. 197
Figure 211 – Hewlett Packard HP-01. 197
Figure 212 – Portable Stereo Sony Walkman. 198
Figure 213 – Evolution of Steve Mann's WearComp wearable computer..... 198
Figure 214 – Seiko's UC 2000 with its keyboard..... 199
Figure 215 – Nelsonic Space Attacker Watch..... 199
Figure 216 – Private Eye head-mounted display. 199
Figure 217 – David Greaves' Active Badge (Cambridge University)..... 200
Figure 218 – Forget-Me-Not. 200
Figure 219 – PalmPilot 1000. 201
Figure 220 – Levi's Industrial Clothing Division Jacket..... 201
Figure 221 – First model of GoPro camera. 202
Figure 222 – Nike+ iPod transmitter in Nike+ Shoe. 202
Figure 223 – Fitbit Classic. 203
Figure 224 – Samsung S9110 Smart Watch. 203
Figure 225 – Google Plus Project Glass..... 203
Figure 226 – Basis personal activity tracker. 204
Figure 227 – MA300 dimensions. 205
Figure 228 – Features of the MA thermistors. 205
Figure 229 – Resistance versus Temperature for the MA thermistors. 206

TABLES

Table 1 – MCP3008 Pins Function [17]. 33
Table 2 – Full-Scale Range and Sensitivity Scale Factor of the Accelerometer [23]. 45
Table 3 – Sensitivity Scale Factor depending on the chosen Full-Scale Range (Accelerometer) [23]. 46
Table 4 – Full-Scale Range and Sensitivity Scale Factor of the Gyroscope [23]. 46
Table 5 – Sensitivity Scale Factor depending on the chosen Full-Scale Range (Gyroscope) [23]. 47
Table 6 –Interrupt Sources [23]. 48
Table 7 – Full-Scale Range of the Gyroscope [25]. 50
Table 8 – Full-Scale Range of the Accelerometers [25]. 51
Table 9 – Comparation of the Raspberry Pi 4 Model B and the Raspberry Pi 3 Model B+ [39]. 67
Table 10 – Specific functions on specific Raspberry pins. 68
Table 11 – Main features of the LILYGO TTGO LoRa32 T3 model [42]. 69
Table 12 – Connection of the LoRa Module Pins within the LoRa32. 70
Table 13 – Arduino Uno R4 Minima specifications [45] 72
Table 14 – Recap of the I^2C communication 78
Table 15 – Transmission of 2 bytes from the Master to the Slave 78
Table 16 – Transmission of 2 bytes from the Slave to the Master 78
Table 17 – Example of the Arduino reading from a specific register of the MAX30100 [31]. 79
Table 18 – Example of the Arduino writing in a specific register of the MAX30100 [31]. 79
Table 19 – SPI Modes [53]. 82
Table 20 – Configure bits for the MCP3008 [17] 84
Table 21 – Wires in the USB 2.0 cable. 86
Table 22 – IEEE 802.11 Standards. 92
Table 23 – Parameters of LoRa. 97
Table 24 – Connection of the MCP3008 pins to the Raspberry pins. 111

Table 25 – Connection of the MPU-6050 pins to the Raspberry pins. 112
Table 26 – Connection of the MAX30100 pins to the Arduino pins. 112
Table 27 –Connection of the Button pins to the Raspberry pins. 113
Table 28 – Structure of the Cayenne Low Power Payload 123
Table 29 – Data Types of the Cayenne LPP. 124
Table 30 – Example of the Cayenne LPP format. 125
Table 31 – LFT and UFT threshold values 151
Table 32 – Ten Readings of HR and SpO2. 159

LIST OF ACRONYMS

ABP	Activation By Personalization
ADC	Analog-to-Digital Converter
ALC	Ambient Light Cancellation
AP	Access Point
ATM	Automated Teller Machine
BSN	Body Sensor Network
CLK	Serial Clock
COVID19	Coronavirus Disease of 2019
CRC	Cyclic Redundancy Check
CS	Chip Select
DAC	Digital-to-Analog Converter
DEE	Departamento de Engenharia Eletrotécnica
FDA	Food and Drug Administration
FIFO	First In, First Out
GPIO	General Purpose Input/Output
H-IoT	Healthcare Internet of Things
HR	Heart Rate
I^2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
IPC	Instituto Politécnico de Coimbra
IR	Infrared Radiation
ISEC	Instituto Superior de Engenharia de Coimbra
ISO	International Standards Organization
LED	Light Emitting Diode
LLC	Logical Link Control
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LPP	Low Power Payload
LSB	Least Significant Bit
MAC	Media Access Control
MEMS	Micro-ElectroMechanical Systems
MHDR	MAC header
MIC	Message Integrity Code
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
MSB	Most Significant Bit

NTC	Negative Temperature Coefficient
OPLA	Older People Living Alone
OSI	Open Systems Interconnection
OTA	Over-the-Air
OTAA	Over-the-Air Activation
PDA	Personal Digital Assistant
PPG	PhotoPlethysmoGraphy
PTC	Positive Temperature Coefficient
PWM	Pulse-Width Modulation
RF	Radio Frequency
SAR	Successive Approximation Register
SBC	Single Board Computer
SCL	Serial Clock
SCLK	Serial Clock
SDA	Serial Data
SiP	System-in-Package
SoC	System on Chip
SPI	Serial Peripheral Interface
SPO2	Peripheral Oxygen Saturation
SS	Slave Select
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTN	The Things Network
UDP	User Datagram Protocol
USB	Universal Serial Bus

LIST OF SYMBOLS

β	NTC thermistor beta parameter

1 INTRODUCTION

The average age of the population is in continuous growth and an estimation of the United Nations shows that in 2030 15.7% of the total population will be 65 years old or above [1]. Consequently, there will be an increase in the number of elderly people who need healthcare and remote health monitoring assistance [1]. The Internet of Things (IoT), which, in the specific case is more precisely defined as Healthcare Internet of Things (H-IoT), is the best solution to this very important issue. In fact, by giving the elderly the possibility for remote health monitoring assistance, they can always be monitored while maintaining a more independent lifestyle [1].

1.1 Older People Living Alone (OPLA)

In Europe 28% of adults aged 60 and older live alone, compared to 16% of adults in the 130 countries and territories studied (World) and to less than 5% in many countries in the Asia-Pacific, sub-Saharan Africa and in the Middle East-North Africa regions (such as Afghanistan, Mali and Algeria [2]). Figure 1 presents the accommodation of people aged 60 and older [2].

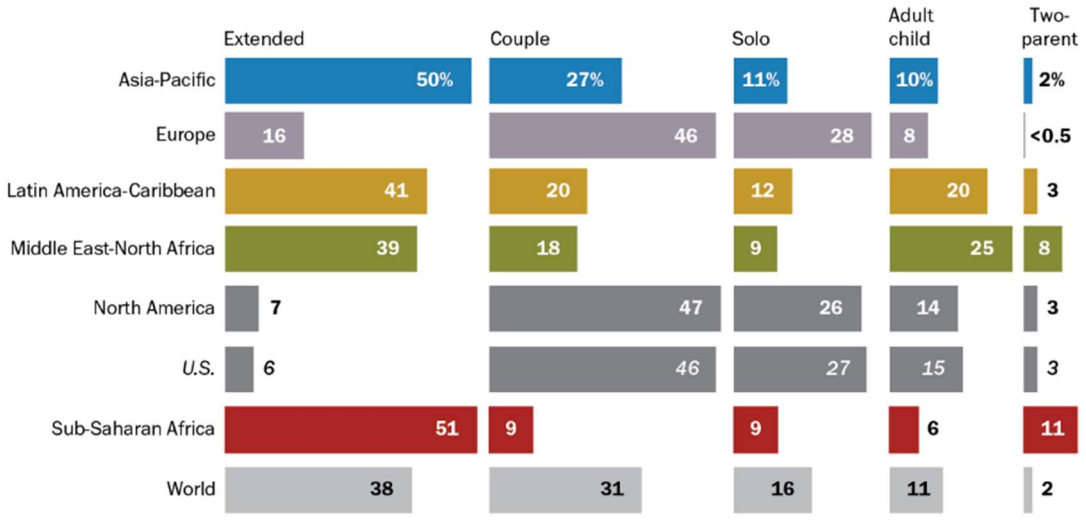

Figure 1 – Percentage of people aged 60 and older in each household type [2].

In Europe 46% of adults aged 60 and older share a home with only one spouse or partner, compared to 31% globally, while 9% of older adults in sub-Saharan Africa live in that kind of arrangement [2].

Globally, 38% of people aged 60 and older live in extended-family households that include relatives such as grandchildren, nephews and adult children's spouses, being

the most common arrangement for older adults. In Iraq, Namibia and India two-thirds or more of the older adults live in that kind of arrangement, compared to only 16% in Europe [2]. Another important aspect to consider is the household size, which is shown in Figure 2 [2].

	Ages 18-59	60+
North America	3.2	2.1
United States	*3.2*	*2.1*
Europe	3.2	2.1
Latin America-Caribbean	4.4	3.4
Asia-Pacific	4.8	3.9
Middle East-North Africa	6.0	4.7
Sub-Saharan Africa	6.3	5.3
World	4.7	3.4

Figure 2 – Average number of residents per household [2].

In average, older people in Europe and in North America live with about one other person, resulting in a household size of 2.1 people, compared to a worldwide average of 3.4 people [2].

In countries with advanced economies, such as Denmark, the UK and South Korea, households are relatively small, being of size 1.7 people, 1.9 and 2.2, respectively. Households are, however, much bigger in less economically advanced countries, such as Gambia, Senegal and Mali with sizes of 12.8, 12.5 and 11.9, respectively [2].

1.1.1 Reasons behind the different household sizes

Generally, households are smaller in more prosperous countries, due to numerous factors, among which education, longevity and economic conditions. Moreover, in more advanced countries, people tend to have relatively few children and to have them later in life and they are also more likely to live well beyond their childbearing years. The Government, also, may offer financial assistance or health care benefits to retired adults, making it more affordable for older people to stay in their own houses [2].

On the contrary, in poorer countries, extended-family households are more common, because the financial resources stretch further, and the chores are more easily accomplished if they are shared among several adults living together. Moreover, since the government provides fewer retirement benefits for older adults, their families have a greater responsibility to care for them [2].

Other important factors to consider are the cultural and religious aspects, which affect the household sizes. For example, 70% of the Hindus live in wide extended families and the vast majority of the Hindus live in India.

1.1.2 Trend

Over the next 20 years, the number of people in their 80s and 90s living alone will dramatically increase. Elderly people that live alone have greater needs for support in the home with respect to elderly people who live in couples [3].

The first baby boomers, the people who were born in North America or in Europe between the years 1946 and 1964, which is the period of the significant population increase called baby boom, will reach the age of 80 within the year 2044. In the USA by 2038, there will be 17.5 million householders in their 80s and over, more than double the 8.1 million in 2018 [3].

Most households headed by someone age 65 or over are either married couples living by themselves (37%) or single individuals (42%). With age, however, the percentage of single-person households in the USA will increase, reaching 58% among those aged 80 and over. Figure 3 shows the number (in millions) of single-person households per ranges of ages in the years 2018, 2028 and 2038.

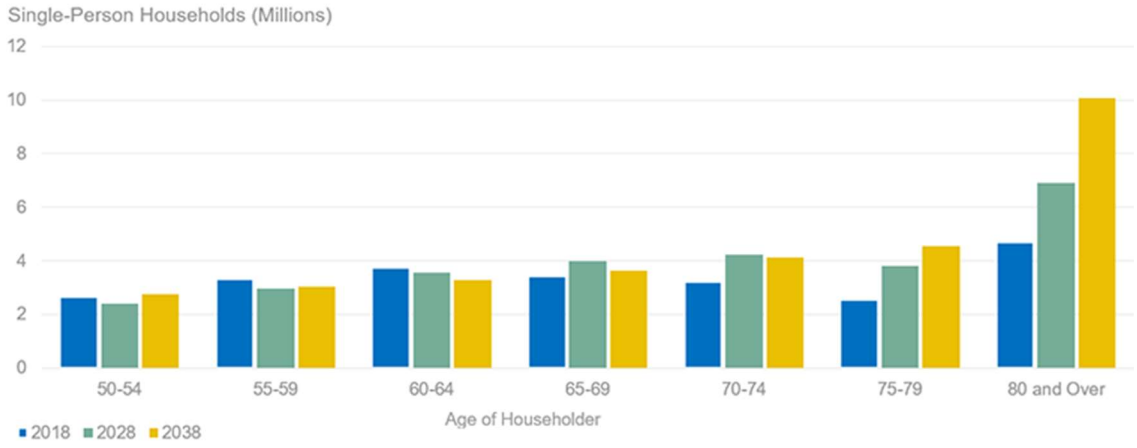

Figure 3 – Single-person households per ranges of ages in years 2018, 2028 and 2038 [3].

When the baby boomers become 80s or 90s over the next 20 years, in the USA the numbers of single-person households among the oldest age group will grow dramatically from 4.7 million households in 2018 to an estimated 10.1 million in 2038 [3].

1.1.3 Consequences of COVID-19

In spring 2020, public health initiatives encouraged limiting face-to-face contact, maintaining a safety distance of 2 meters from others and remaining at home as much as possible, with the purpose of reducing the spread of COVID-19. As a result, older adults who live alone have experienced greater isolation during the pandemic compared to the older adults that share their house with others [4]

Figure 4 summarises the impact that COVID-19 had on the elderly people [4].

Figure 4 – Elderly loneliness during COVID-19 [4].

The impact of COVID-19 on the elderly has caused a huge increase in loneliness, because of the lockdown, which has been particularly devastating for those living alone. Moreover, over half of the elderly has less likely been in any contact with their family and has experienced the fear of being left alone. The measures against the virus have impacted their ability to get essential items such as groceries [4].

1.2 Scope and motivation

As the mean age of the population and the life expectancy increase, there will be always more elderly people living alone, especially in the economically advanced countries. Thus, it is necessary to improve more and more the efficiency of the homecare monitoring practices, with the aim of letting them live in safety, preventing any possible health-related issues.

To monitor the health status of these elderly people, some parameters were selected with the aim of identifying a serious deterioration in health or the occurrence of an accident, namely:

- Body Temperature;
- Fall Detection;
- Heart Rate;
- Saturation of peripheral Oxygen (SpO2).

1.2.1 Body Temperature

Normothermia is the typical human body temperature range. This range may change depending on the age, gender, time of the day, health conditions, the state of consciousness or emotions of the person and can even vary depending on which part of the body the measurement is taken at, but it typically varies from 36.5 ºC to 37.4 ºC [5]. The human body temperature detection can be done at different parts of the body, by using infrared temperature sensors or thermistors, which are made of semiconductor material. The detection, in those cases, is possible thanks to the well-known resistance-temperature relationship that exists in the thermistors.

1.2.2 Fall Detection

A very important issue is the Fall Detection, as more than one third of 65 and above year-old people falls at least once a year. According to the World Health Organization, falls for the elderly are often the cause of significant morbidity and mortality that arise from head injury and fractures. Fall detection systems are very useful, because they allow a quick response, thus decreasing the risk of a serious medical condition. The solution stands in a wearable device equipped with an accelerometer and a gyroscope attached to the patient [6].

1.2.3 Heart Rate

The term Heart Rate (HR) or pulse rate indicates the number of times a heart beats per minute and its value depends on factors such as age, body size, movement, exercise and heart conditions. Generally, it is a value between 70 and 100 beats per minute (bpm). Since it is estimated that by 2030, in the United States, more than 8 million people will be diagnosed with heart failure, the measurement of this parameter is of fundamental importance. Generally, the heart failure treatment in hospitals or in specialized clinics is very expensive. This cost is certainly higher than that of a wearable IoT device aimed at doing measurements. Moreover, the measurements done by a wearable device are continuous and can lead to early pathology detection and prevention of health complications. Other advantages are

the non-invasiveness of the process and the fact that this data can even be collected and stored for further analysis made by a medical specialist. The heartbeat rate detection is done by the heart rate detector [7].

1.2.4 SpO2

In order to live, the human body needs a certain amount of oxygen in the blood, otherwise it experiences an energy loss with the consequence of not working efficiently. The SpO2 is the level of oxygen saturation in the blood and it normally is around 95%-100%. The oxygen saturation level is defined as the value in the blood, of the percentage of haemoglobin that carries oxygen, in respect to the total haemoglobin that is in the blood. The technique aimed at measuring the oxygen saturation level is the Pulse Oximetry, which is non-invasive and is performed by a device that calculates the oxygen saturation named pulse oximeter. The pulse oximeter uses a photodetector and two different light wavelengths: the infrared light, with a wavelength of 660 nm, and the red light, with a wavelength of 880 nm. The finger of the patient is placed on the SPO2 sensor, so that the amount of light absorbed by the finger in those two wavelengths is analysed. The principle on which this method works is that the infrared light is absorbed more by the oxygenated blood and less by the deoxygenated blood, while for the red light it is exactly the opposite situation [8].

1.3 Objectives

The objective of this project is to develop, implement and test a Health Monitoring System for Elderly People, based on a Body Sensor Network (BSN) and with a wearable system architecture. The proposed system is suited for patients that are not in critical condition but need a continuous or a periodical monitoring.

The system is able to monitor the body temperature, the heartbeat, the SpO2 and to check if the patient has fallen. The collected data is, then, stored in an external server, so that the doctor can remotely analyse the patient's health condition, without the need for the patient to go to the doctor or to the hospital, whenever possible. The detected data is also stored in a local database.

Three non-invasive sensors detect the main vital signs:

- Body Temperature. This contact-sensor can be applied to the skin to detect any temperature-related abnormal activities, such as fever or hypothermia.
- Fall Detection. Based on accelerometers and gyroscopes, this sensor consists of micro-machined structures on a silicon wafer designed to measure acceleration.

- Heart Rate Detector and Pulse Oximeter. It measures the number of times the heart beats per minute and the SpO2 percentage in the blood. The light is emitted using light-emitting diodes (LEDs) and, then, the backscattered received light intensity is measured using a photodetector.

The system also has a dual-purpose button, which can be used by the patient to interact with the system. Normally it is an emergency button and, when pressed, has the aim to send the information of emergency to the patient's doctor or relative. In addition, in pre-set intervals of time and in a certain range of hours, the system asks the patient about the health conditions. In these occasions the button is a confirmation button, and the patient has to press it within a timer in order to confirm that everything is alright. If the patient does not press the button, then an information of emergency is sent. Figure 5 represents the basic architecture of the system developed in this project, necessary to achieve the desired objectives.

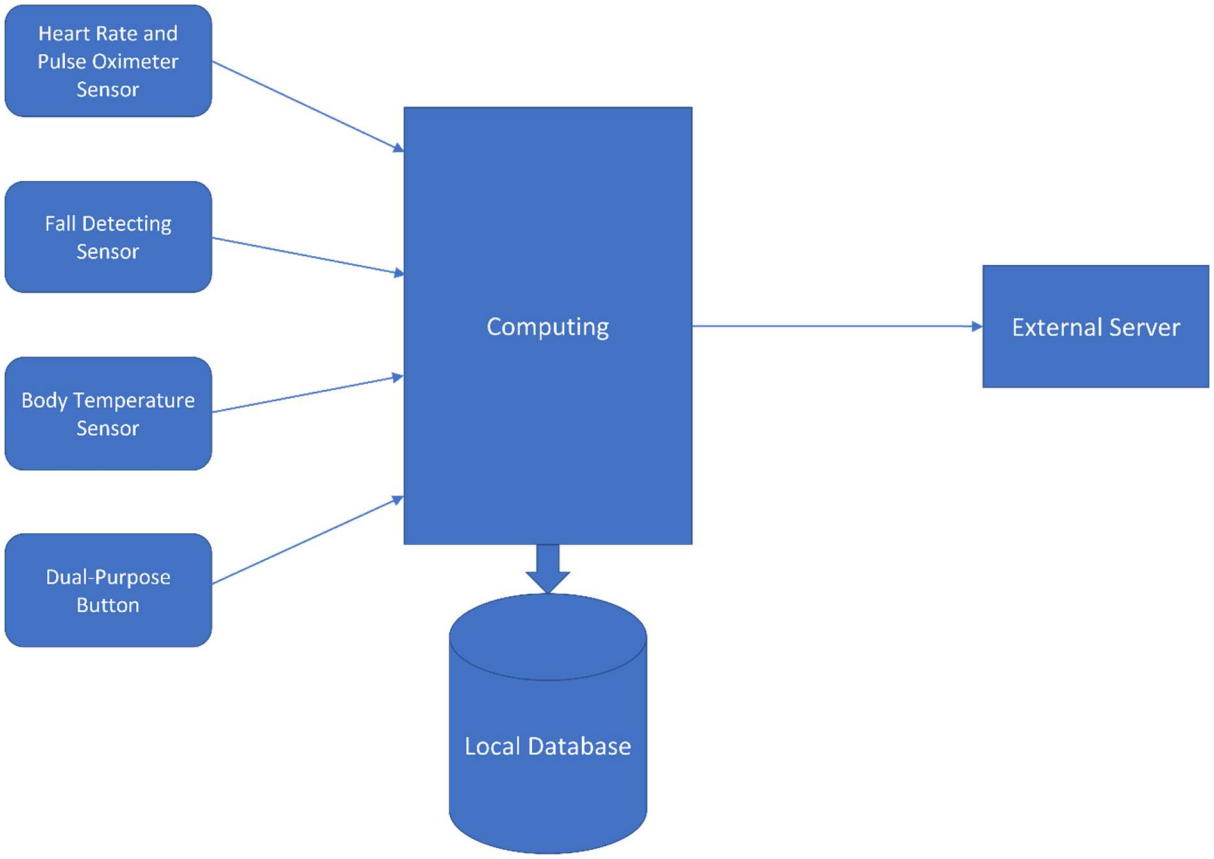

Figure 5 – Essential Architecture of the System.

In the next chapters each component of this architecture is going to be analysed in detail, specifying which are the devices and the servers used.

1.4 Document structure

This project report is structured in the following chapters:

Chapter 1, "Introduction", deals with the continuous increase of the mean age of the population and the situation of the elderly people living alone and estimates which is going to be the trend for the immediate future. Subsequently, it describes the consequences that COVID-19 had on the elderly people living alone. Lastly, it presents the scope and motivation of this project and its objective.

Chapter 2, "State of the Art", focuses on wearable devices of the Close-to-medical area. Subsequently, it presents Internet of Things and its three-layered architecture.

Chapter 3, "Architecture of the Planned System", describes and graphically illustrates the architecture of this project and the implemented IoT system.

Basing on the IoT three-layered architecture, Chapter 4, "Things or Devices Layer", deals with the first layer. Belonging to this layer, the sensors are described in detail together with their operating principles. The edge computing is also part of this layer, so this chapter continues with a description of the Raspberry Pi 4, the TTGO LoRa32 and the Arduino R4 Minima.

Chapter 5, "Serial Communication Protocols", analyses the protocols used in the internal communications between the various components of the developed system. Firstly, there is a description of the protocols used for the respective serial communications between the sensors and the Raspberry or the Arduino (I^2C and SPI). Next, the chapter explores the USB serial communication between the Raspberry and the LoRa32 and between the Arduino and the Raspberry.

Chapter 6, "Connectivity and Application Layers", describes the second and the third levels of the IoT three-layered architecture. Belonging to the second layer, there are all the protocols that allow the developed system to communicate with the external servers. Therefore, after a description of the ISO OSI Model, the chapter explores the protocols Wi-Fi, TCP, IP, MQTT, LoRa and LoRaWAN. This chapter presents also the last level of the IoT three-layered architecture. Therefore, it deals with the two external servers used in this project: The Things Network and Akenza and also the integration between them.

Chapter 7, "Implemented System", begins with the illustration of how the different components are concretely connected in the implemented architecture. Subsequently, the chapter describes all the steps necessary for the initial configuration of the Raspberry and Akenza. Lastly, it analyses in detail the software modules running on the Arduino, the Raspberry and the LoRa32.

Chapter 8, "Experimental Results from the Developed Prototype", presents all the partial tests that were individually made on the sensors to demonstrate and analyse their functioning. Firstly, the chapter deals with the tests on the USB communication

between the Arduino and the Raspberry and graphically shows the Heart Rate and SpO2 plots, that were obtained from them. Subsequently, it explains the behaviour of the LoRa32 when receiving from the Raspberry different types of information with different priorities, namely Fall Detection, Emergency, Message and Sensors data. Subsequently, the chapter presents the tests on the fall detection software and graphically shows the obtained plots of the modulo of the acceleration. The chapter shows that the algorithm always detects a fall and never mistakes a normal daily activity for a fall.

Chapter 9, "Field Test Results Analysis from Platform Dashboard", presents the use of the prototype for two hours, demonstrating its functioning in real-life situations (including a fall and some emergency situations). This chapter also shows the Akenza Dashboard with all the data that the implemented system sent to Akenza and all the warning emails that Akenza sent in a fall detected or in emergency situations.

Finally, Chapter 10, "Conclusions and Future Work", delineates the most relevant conclusions of the work and some suggestions for future work.

2 STATE OF THE ART

As seen in the previous chapter, the objective of this project is to develop a wearable Health Monitoring System device for elderly people, based on several sensors and supported by IoT technology. The state of the art on these wearable devices is going to be presented in this chapter. After an introduction on the history of these wearables, the focus is going to be on the wearables belonging to the specific area of this project. Subsequently, the three-layered architecture of IoT systems is going to be examined in detail.

2.1 Wearable devices

The terms wearables, wearable devices and wearable technology are used to identify small electronic and mobile devices (generally provided with wireless communications capability), which can be worn. Some of them are incorporated into accessories or clothes, while others are invasive, as for example micro-chips or smart tattoos. Nowadays, the number of wearables is increasing more and more and, as their technologies are in continuous development, they are able to collect more information and to produce always more detailed data, in order to provide an even more personalised experience [9]. Wearable technology is used in many areas, such as the industrial, healthcare and sports environments [9].

In the healthcare area, wearables can monitor the patient thanks to their sensory physiological functions such as the biometry-related ones and thanks to their wireless communication and energy-efficient computing; moreover, they have the advantages to be convenient, seamless and portable [9].

2.1.1 Historical perspective on wearables

In this section, Subsection 2.1.1., together with Annex A, present a far back-in-time historical perspective on general applications of wearables, while Subsection 2.1.2 and Subsection 2.1.3 focus on more recent medical and close-to-medical device development.

Figure 6 represents the evolution of the wearables from the 13th century to the recent days [9]. A detailed description of general applications of a significant number of representative wearable devices is further presented in Annex A.

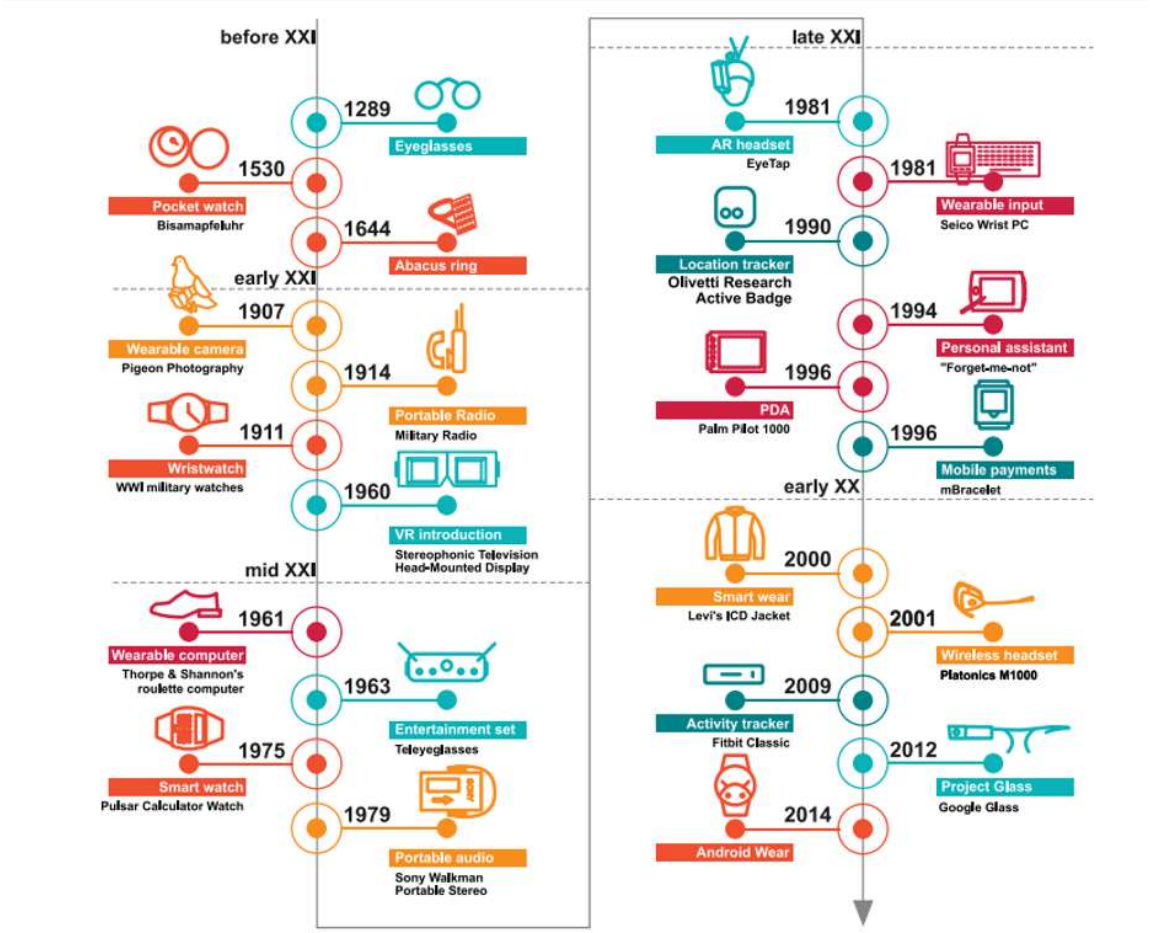

Figure 6 – Evolution of wearables from the 13th century to nowadays [9].

As seen in this historical timeline, the different wearables cover the most various application areas. The area of interest of this project is the close-to-medical area, that consists of the wearable devices that aim at detecting some health-related issues, although they are not certified as medical ones.

2.1.2 Medical and Close-to-Medical area

An example of close-to-medical wearables is the Chronolife smart vest presented at the Consumer Electronics Show (CES) 2019 event that, being designed to prevent medical emergencies in patients with congestive heart failure, is able to predict a heart-attack beforehand (Figure 7).

Figure 7 – Chronolife smart vest.

Some devices have the capability to monitor the posture of the user, such as Upright and Lumo Lift. Many companies, such as POLAR, Garmin and Fitbit have the heart rate monitoring feature but are not interested in receiving the Food and Drug Administration (FDA) approval, while other medical wearable devices have succeeded in receiving it. Among the latter category there are, as represented in Figure 8, the SYNCHRONY and SYNCHRONY 2 cochlear implants, which have also received the approval for people with single-sided deafness and for people with asymmetric hearing loss.

Figure 8 – SYNCHRONY 2 cochlear implants.

The Neofect's powered glove, presented in Figure 9, was specifically designed for people with a paralysed hand caused by the stroke, multiple sclerosis or amyotrophic lateral sclerosis. It allows performing simple everyday tasks such as brushing their teeth, opening doors, or drinking from a cup.

Figure 9 – Neofect's powered glove.

Other wearable researchers are interested in creating an artificial nervous system; among them the Researchers from NUS Materials Science and Engineering developed the touch equivalent to human skin with the Asynchronous Coded Electronic Skin, which can provide tactile feedback in prosthetics (Figure 10).

Figure 10 – Asynchronous Coded Electronic Skin.

Many researchers are interested in the Parkinson's disease, such the Austrian EVER Pharma, which has received CE Mark approval and is releasing its D-mine Pump in Europe (Figure 11).

Figure 11 – D-mine Pump.

Nowadays, among the devices whose features are the most similar to those of the device developed in this project, certainly, there is the Apple Watch.

2.1.3 Apple Watch Series 10

In 2014 the Android Wear, which is currently known as Wear Operating System, was introduced. Wear OS was the first operating system specifically designed for wearable devices, particularly for smartwatches, and it was the beginning of Google's move towards taking a vital position in the wearable market [9].

On the other hand, Apple Watch has a more tragic background story [9]. Before 2011, Steve Jobs unsuccessfully fought pancreatic cancer and experienced the difficulties for the nursing staff in communicating with the patient, in monitoring his health and in collecting the data remotely outside the hospital, while the existing activity trackers were not suitable for this purpose. This situation led Steve Jobs to think that the existing medical care could be improved with technology. Thus, in 2015, four years after his death, Apple released the Watch (Figure 12), a device equipped with a powerful software and cloud platform and aimed at solving the problem of collecting and structuring data to facilitate the remote monitoring from the doctor [9].

Figure 12 – Apple Watch Series 10.

The current version of the Apple Watch is the Series 10 which, thanks to its sensors and apps, allows to take, day and night, on-demand readings of the blood oxygen and background readings. Moreover, with the ECG app, Apple Watch Series 10 is capable of generating an electrocardiogram. Another great feature of the Apple Watch Series 10 is the detection of a hard fall (Figure 13).

Figure 13 – Detection of a hard fall in Apple Watch Series 10.

In fact, if the patient falls while wearing the watch, it taps on the wrist, sounds an alarm and displays an alert, so that the patient can choose to contact emergency services or to dismiss the alert by pressing the Digital Crown, tapping "Close" or tapping "I'm OK" [10].

2.2 Internet of things (IoT)

The term Internet of Things (IoT) was firstly used by Kevin Ashton in 1999, when talking about the supply chain management with radio frequency identification RFID-tagged or barcoded items, which offered a greater efficiency to businesses. In the RFID Journal of 22 June 2009, Ashton wrote:

"If we had computers that knew everything there was to know about things – using data they gathered without any help from us – we would be able to track and count everything, and greatly reduce waste, loss and cost. We would know when things needed replacing, repairing or recalling, and whether they were fresh or past their best" [11].

In the same year, in his work "When Things Start to Think", Gershenfeld described the evolution of the World Wide Web as a state in which "things start to use the Net so that people don't need to" [11].

The more general term "Net" is more adequate than "Internet," since not all the communication occurs via the Internet. Similarly, since the communications do not exclusively happen between things, but also between things and people, the use of terms such as "Net of Everything" would be more correct than "Internet of Things" [11]. IoT is in continuous evolution, so its definition is also in a continuous developing and the IEEE IoT even gives its community members an opportunity to contribute to its definition [11].

"An IoT is a network that connects uniquely identifiable 'Things' to the Internet (through the utilization of standard communication protocols). The 'Things' have sensing/actuation and potential programmability capabilities. Through the exploitation of unique identification and sensing, information about the 'Thing' can

be collected and the state of the 'Thing' can be changed from anywhere, anytime, by anything" [11].

IoT is a world of interconnected things capable of sensing, actuating and communicating among themselves and with the environment, which are able to take data from the physical world events and to share it with the other things with the purpose of creating services. The things offer services, with or without human intervention and these services are exploited using intelligent interfaces [11].

2.2.1 Architecture of the IoT

Figure 14 shows a three-layered stack of IoT technologies: the things or devices layer, the connectivity layer and the application layer.

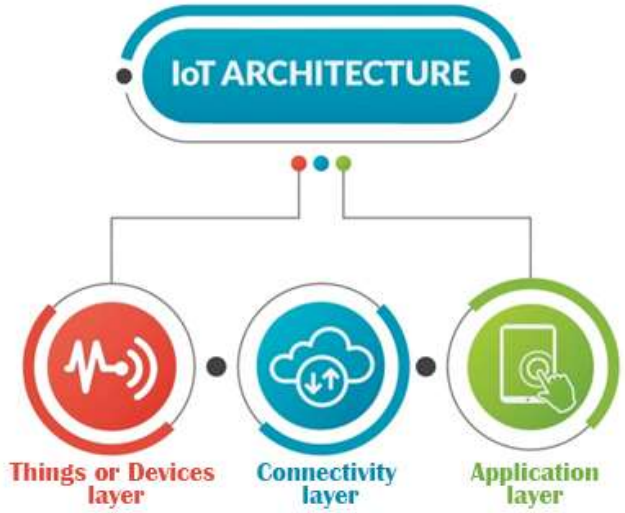

Figure 14 – Three-layered IoT Architecture. Adapted from [12].

The Things or Devices Layer

This layer collects real-world information through the IoT devices. The devices are the sensors, the actuators and the microcontrollers, which represent the physical or perceptual IoT layer. Since the IoT devices are at the "edge" of the IoT network, they are also called "edge nodes". Most of the devices are simple sensors which have to be interfaced with a single-board computer, such as the Arduino or the Raspberry, by using the conventional analog or serial connections. On the contrary, some newer devices are IoT-ready, as they already have a microcontroller inside and the access to the Internet [13].

The Connectivity Layer

This layer implements all the protocols that are needed to transport the gathered real-world data from the devices to the cloud. The devices communicate with the cloud through wire or localized Radio Frequency (RF) networks [13].

The Application Layer

The Application layer is the Cloud, which communicates with the device, typically over wired or wireless Internet. It provides servers and databases that allow data storage, big data processing, filtering, analytics, alerts, monitoring and user interfaces [13].

The main focus of this chapter was on the wearables of the Close-to-medical area, in particular on the Apple Watch 10, which has many features in common with the device of this project. Finally, the IoT was described, as it is the base of the Health-monitoring System developed in this project. The next chapter is going to analyse in detail the architecture of the planned system.

3 ARCHITECTURE OF THE PLANNED SYSTEM

In this chapter the architecture of the planned system is presented, and all the hardware components are graphically identified, also specifying the communication between them. Figure 15 presents the architecture of the projected and implemented IoT system.

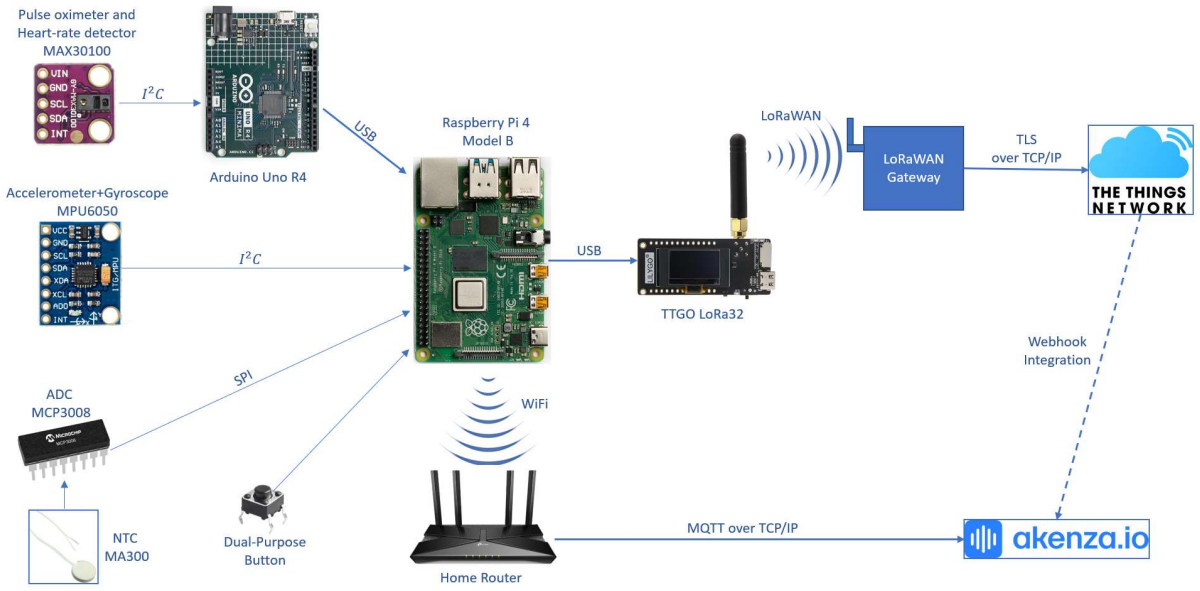

Figure 15 – Architecture of the Implemented System.

By referring to the three-layers IoT architecture, it is possible to distinguish three levels: the things or devices layer, the connectivity layer and the application layer.

3.1 The things or devices layer

Belonging to this level, there are the sensors and the devices that perform the edge computing and send the data to the cloud.

3.1.1 Sensors

In order to record the body temperature, a MA300 NTC thermistor was selected and is used in the system. Since the Raspberry is not equipped with an ADC, the use of an external one is necessary; in the specific case a MCP3008 ADC is used. The communication between the ADC and the Raspberry is of type SPI.

The MPU-6050 Six-Axis (Accelerometer + Gyroscope) is used to detect the movements of the patient and, thus, to understand if the elderly has fallen. The communication between this sensor and the Raspberry is of type I^2C.

The MAX30100 sensor is the Heart Rate Sensor and Pulse Oximeter. This sensor was selected and implemented in the final prototype to measure the heartbeat per minute and to analyse the level of oxygen in the blood of the patient. Since in the Raspberry there are no libraries to communicate with the MAX30100, the use of an Arduino R4 Minima was necessary. The Arduino reads the data through the I^2C communication from the MAX30100 and then sends it to the Raspberry through USB communication.

3.1.2 Dual-purpose Button

Among the devices there is also a dual-purpose button connected to an input pin of the Raspberry. More specifically, the button is connected using an internal pull-up resistor. This button is normally an emergency button and, only on some occasions and for a very short time, it becomes a confirmation button.

3.1.3 Edge computing

The edge computing is the task that the Raspberry has to accomplish. The Raspberry collects the data from the sensors, stores it in its local database and sends it:

a) directly to the cloud of Akenza, if in the Wi-Fi range;

b) to the LoRa32, through the USB connection, if not in the Wi-Fi range.

3.2 The connectivity layer

There are two different data paths, in which different protocols are used for the communication with the cloud.

a) Whenever the device is within Wi-Fi range, the Raspberry uses the Wi-Fi connection to send the collected data to the home router. Then, the home router sends the data directly to Akenza through the protocol MQTT over TCP/IP.

b) If the device is out of range of the Wi-Fi network, the Raspberry sends, through the USB connection, the collected data to the LoRa32, which uses the LoRaWAN technology to send the data to a LoRaWAN gateway. Subsequently, this LoRaWAN Gateway sends the data to TTN through the protocol TLS over TCP/IP. Then, TTN sends the data to Akenza, through a webhook integration.

3.3 The application layer

As already seen, two clouds are employed in this project:

a) Akenza's cloud is always the final destination.

b) TTN's cloud when not in the Wi-Fi range which, either way, uses Akenza as a webhook integration.

Akenza allows the final user to graphically see the data collected from the system.

On the basis of the three-layered IoT architecture, this chapter showed all the components of the architecture of the planned system. Each layer is going to be described in detail in the next chapters, beginning with the things or devices layer, handled in Chapter 4.

4 THINGS OR DEVICES LAYER

This chapter is about the things or devices layer, thus all the sensors with their operating principles and the Raspberry Pi 4, the TTGO LoRa32 and the Arduino R4 Minima are analysed.

4.1 NTC Thermistor for Body Temperature Measurement

This section begins with a detailed analysis of the body temperature, then there is a description of the operating principle of the NTC and a description of the MA300, which is the NTC used in this project.

4.1.1 Body Temperature

There are two different temperatures: the core body temperature and the body shell temperature.

Core body temperature

When talking about body temperature, it usually means the core body temperature, that is to say the temperature that prevails in the body core (the inside of the head, chest and abdominal cavity). The human core body temperature is largely constant, thanks to the interaction of heat generation, heat absorption and heat release [5].

In a healthy adult, the normal core body temperature is approximately 36.5 °C to 37.4 °C [14]. However, this can deviate temporarily depending on the time of day, activity, state of health or hormonal status (menstrual cycle in women). When exposed to high heat, physical work and sports, for example, the core body temperature can rise to around 38 °C to 39 °C [5].

In some organs, the temperature is higher than the average normal core body temperature. For example, the temperature in the human heart is about 38.8 °C and even higher in the liver [5].

Body shell temperature

In addition, there is the surface temperature or body shell temperature. This is measured on the surface of the skin and results from the ambient temperature and the temperature inside the body. It is dependent on blood circulation, activity and external temperature and serves as a temperature equaliser for the body. Unlike the

core body temperature, the human body shell temperature can therefore vary considerably (approximately 28 °C – 37 °C) [5].

Figure 16 shows the effects of the ambient temperature on the body shell temperature [5].

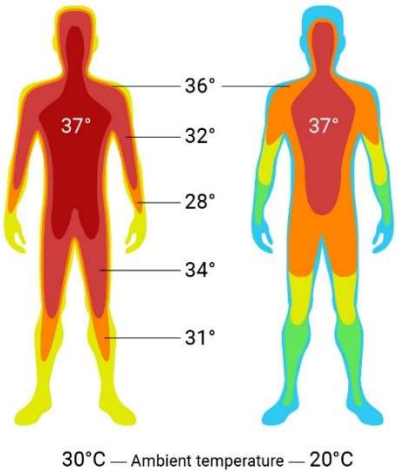

Figure 16 – Temperatures in the human body [5].

Depending on the outside temperature, there can be large differences between the temperature of the body shell and that of the core body.

Hyperthermia, Fever and Hypothermia

The core body temperature can assume values that are not in the normal range. When the core body temperature is too high, hyperthermia or fever can occur. A core body temperature of over 42.6 °C is fatal, as organs and tissues damage occur above this value. On the contrary, when the core body temperature is too low hypothermia takes places. A body temperature below 27 °C can be fatal [5].

Hyperthermia

Hyperthermia occurs when the body core temperature rises above 37.4 °C and there is no setpoint increase in it from the thermoregulation centre and the body's own thermoregulation is counteracted (in contrast to what happens in case of fever) [5].

The causes can be:

- genetic defects (malignant hyperthermia);
- short-term heat development due to fluid deficiency in newborns (transitory hyperthermia);
- disorders of the thermoregulatory centre (central hyperthermia);
- a lack of heat release (heat stroke);
- an artificial hyperthermia in the context of a cancer treatment.

As a rule, hyperthermia occurs during high physical performance in a hot environment, whereas it is rather rare in the case of only external heat stress without physical work [5].

Hyperthermia manifests itself, in addition to an increased body temperature, through symptoms such as a reddened, overheated, initially usually dry skin, an increased respiratory rate, heart palpitations (tachycardia) and sometimes through changes in consciousness. Rarely, seizures or febrile convulsions occur as sequelae [5].

Fever

If the core body temperature exceeds 37.4 °C and the increase in the core body temperature takes place as a result of a setpoint adjustment of the thermoregulation centre, this is referred to as fever. This means that the body "intends" to increase the core body temperature. Heat production is increased and, at the same time, heat release is kept constant or reduced [5].

In many cases, fever takes place due to an immune response of the body (for example, to defend against flu viruses). Thus, the activity of many immune cells is increased when the temperature is elevated. At the same time, the growth of some pathogens is inhibited. However, if the fever rises above 41 °C, the body's own proteins may be destroyed (denaturation) and blood clotting may be disturbed [5].

Hypothermia

Hypothermia is the term used for a drop in core body temperature below 35 °C. Localized hypothermia, e.g. of the hands or feet, can cause frostbite. If general hypothermia of the entire body takes place, severe health damage or even death from frostbite may result [5].

Depending on the temperature and the accompanying symptoms, four stages of hypothermia are distinguished:

Defensive stage (excitation), 35-32 °C: No loss of consciousness but restlessness, muscle tremors, hyperventilation, hypertension, palpitations.

Exhaustion stage (adynamia), 32-30 °C: Apathy and confusion, arrhythmic and shallow breathing, slowed heartbeat and decreased blood pressure, increasing stiffness of muscles and joints.

Paralytic stage (paralysis), 30-27 °C: Unconsciousness, mydriasis (dilatation of pupils), shallow breathing, extreme slowing of heartbeat, low blood pressure.

Apparent death (vita reducta), below 27 °C: Deep unconsciousness with dilated light-starved pupils; possible respiratory arrest, ventricular fibrillation of the heart or cessation of cardiac actions.

The causes of hypothermia are manifold. For example, an incompletely functioning thermoregulatory system may be responsible for the severe reduction in core body temperature (e.g., triggered by burns, paraplegic injury, brain damage, poisoning). In addition, other causes may include malnutrition, reduced metabolism, illness or injury, reduced to no ability to produce heat with the help of cold shivering, vasodilatation, sweating in a cold environment, and large fluid or blood losses (e.g., during surgery) [5].

Methods of Temperature Measurement

The various measurement methods can be distinguished depending on their invasiveness:

- Non-invasive: temperature measurements that are taken through the skin.
- Less invasive: In these temperature measurements, the thermometer is inserted into a natural body orifice without great comfort impairment for the patient (mouth, ear, rectum).
- Invasive: These temperature measurements take place via a body orifice with comfort impairment for the patient (oesophagus, urinary bladder, nasopharynx) or directly in the body tissue (needle probes) or in the blood vessels system (for example the pulmonary catheter). The most accurate measurement results can be obtained with the invasive methods.

4.1.2 NTC Thermistor

The term thermistor is an acronym that stands for THERMally-sensitive resISTOR and is used to indicate the resistors made from semiconductor materials whose

resistance value depends on the temperature of the material. Both the ambient temperature in which the thermistor operates, and the temperature rise due to self-heating, caused by the power dissipation within the thermistor itself, determine the value of this temperature. The thermistors are used in applications as temperature sensors. There are two types of thermistors: the thermistors that have a positive temperature coefficient of resistance (PTC) and the ones with a negative temperature coefficient of resistance (NTC), which were the first thermistors to have been developed. The difference between the two types stands in the fact that, for the ones with a positive temperature coefficient of resistance, the resistance increases as the temperature increases, while for the other ones the value of the resistance decreases with the increase of the temperature [14].

The negative temperature coefficient thermistors are manufactured from the oxides of materials such as iron, chromium, manganese, cobalt and nickel which, in their pure state, have a high resistivity. They can, however, become semiconductor materials, by adding small quantities of a metal that has a different valency. One group of NTC thermistors is manufactured from Iron(III) oxide Fe_2O_3, whose molecule is represented in Figure 17 [14].

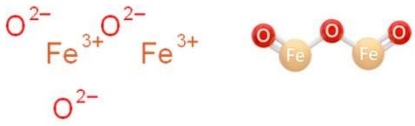

Figure 17 – Iron (III) oxide Fe_2O_3.

In this molecule a trivalent ferric ion Fe^{3+} is replaced with a tetravalent titanium ion Ti^{4+}, as shown in Figure 18 [14]. It is important to notice that the Fe atom, in the molecule, became the Fe^{3+} ion, because it gave three of its electrons to two atoms of oxygen which, as a consequence, became ions.

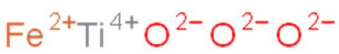

Figure 18 – Iron (III) oxide Fe_2O_3 after the replacement.

In this new molecule, since titanium is Ti^{4+}, in order to maintain the electrical neutrality, the Fe atom becomes the bivalent Fe^{2+} [14].

At low temperatures, the two electrons given by the Fe^{2+} are situated on iron ions next to the Ti^{4+} ions but, at higher temperatures, these electrons are loosened from their sites and become free charge carriers. In fact, these electrons go to the

conduction band, thus decreasing the resistivity of the material and, consequently, the resistance decreases with the increase of the temperature. If a voltage is applied on the material, these electrons can easily move in a flow, thus producing a current [14].

The first NTC thermistor was discovered in 1833 by Michael Faraday who, while analysing the semiconducting behaviour of silver sulphide, noticed that its resistance decreased dramatically with the increase of the temperature [14].

Because early thermistors were to be produced and applications for the technology were limited, their production was very difficult and only began in 1930, when Samuel Ruben invented a commercially viable thermistor [14].

By using a first-order approximation, considering that resistance and temperature are linked by a linear relationship, then their relation is:

$$\Delta R = k\Delta T$$

where

ΔR is change in resistance;

ΔT is change in temperature;

k is the first-order temperature coefficient of resistance.

In real devices this linear approximation is accurate only over a limited range of temperatures while, when considering wider ranges of temperatures, it is necessary to use the Steinhart–Hart equation, which uses a more complex third-order approximation:

$$\frac{1}{T} = a + b\ln R + c(\ln R)^3$$

In this equation a, b and c are called Steinhart–Hart parameters and are specific for each device, T is the absolute temperature and R the resistance.

Where $a = \frac{1}{T_0} - \left(\frac{1}{B}\right)\ln R_0$, $b = \frac{1}{B}$, $c = 0$.

By substituting a, b and c into the Steinhart–Hart equation, this B (or β) parameter equation is obtained:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B}\ln\frac{R}{R_0}$$

where the temperatures are in kelvin, T_0=298.15 K (25 °C) and R_0 is the nominal resistance value, the resistance of the thermistor at T_0.

4.1.3 MA300

Among other viable solutions, the MA300 Biomedical Chip NTC thermistor from Amphenol Thermometrics was selected for use in this project [15]. The main specifications and the parameters values of this NTC are presented in Appendix A.

This thermistor is specifically designed for applications in which the temperature of the patient must be monitored intermittently or continuously. Since the measurement of the temperature is very important, especially for the elderlies, a fast response is essential. This specific thermistor operates in the range of 0 °C to 50 °C and has great precision, tight tolerance, and fast response times, which are fundamental in the medical applications. The MA biomedical Chip Thermistors have their best stability when exposure and storage temperatures remain below 70 °C.

The continuous patient temperature monitoring can be done by simply using the patient's skin site as an indicator of the body temperature. These thermistors are available with the nominal resistance values of 2252 Ω, 3 kΩ, 5 kΩ and 10 kΩ at 25 °C. In this project a 10 kΩ nominal resistance (@ 25 ºC) MA300 was selected to be used as a temperature probe.

The β parameter value was obtained by applying the formula:

$$\beta = \frac{\ln\left(\frac{R_{T1}}{R_{T2}}\right)}{\frac{1}{T_1} - \frac{1}{T_2}}$$

Two temperature values 25 ºC and 50 ºC were used in that evaluation:

$T_1 = 25ºC + 273.15 = 298.15\ K$ and $T_2 = 50ºC + 273.15 = 323.15\ K$.

$R_{T1}$and R_{T2} are the values of the resistance of the NTC at respectively 25 ºC and 50 ºC and are obtained from the datasheet, column of 10 kΩ nominal resistance (see Figure 229).

By applying the previous formula:

$$\text{B}(25/50) = \beta = \frac{\ln\left(\frac{R_{T1}}{R_{T2}}\right)}{\frac{1}{T_1} - \frac{1}{T_2}} = \frac{\ln\left(\frac{10k\Omega}{3603.46\ \Omega}\right)}{\frac{1}{298.15} - \frac{1}{323.15}} = 3933.63\ K$$

Thus, the β parameter value used in this project is equal to 3933.63 K for this temperature range 25/50.

4.2 Analog-to-Digital Converter ADC

The Analog-to-Digital Converter (ADC) has the task to convert a continuous-time and continuous-amplitude analog signal to a discrete-time and discrete-amplitude digital signal. The ADC does not perform the conversion of the input signal continuously, but periodically samples it (Figure 19).

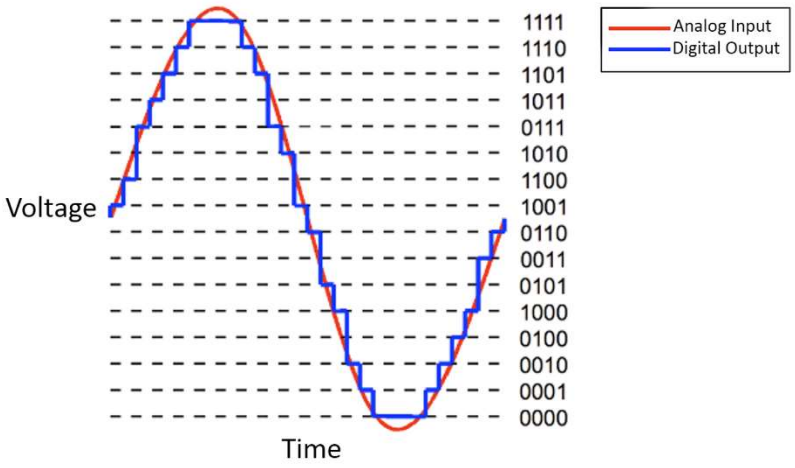

Figure 19 – Example of 4-bit ADC conversion.

This conversion unavoidably introduces a small amount of error or noise, due to the fact that the input signal is being quantized.

4.2.1 ADC Resolution

The resolution of the ADC is the number of bits that the ADC uses to express the output converted digital number. The number of the levels, that is to say the number of the discrete values that the output number can assume is given by $2^{resolution}$. For example, an ADC with a 10-bit resolution can translate the input signal value into one of 1024 different levels ($2^{10} = 1024$). Since there are 1024 levels, the output from the ADC can assume the values from 0 to 1023.

The converter used in this project is a successive-approximation ADC with a 10-bit resolution.

4.2.2 Successive-approximation ADC

In the successive-approximation ADC, the control logic is made of a Successive Approximation Register (SAR), as shown in Figure 20 [16].

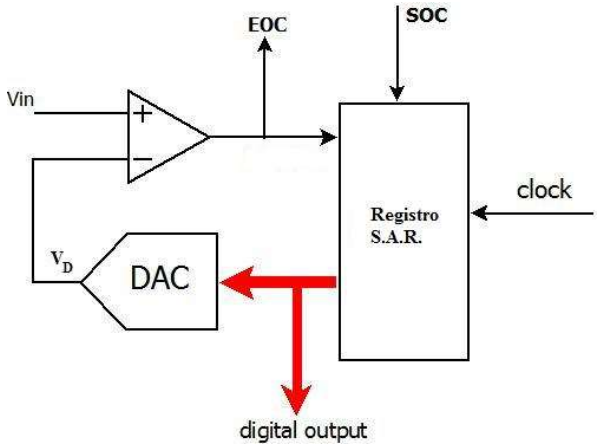

Figure 20 – Successive-Approximation ADC [16].

The conversion is made by comparing the output of a DA converter with the analog voltage that has to be converted. It operates as follows:

- The conversion begins when the SOC signal is sent to the SAR. By proceeding this way, in the SAR a number will be loaded, in which only the most significant bit (MSB) is set to 1, while all the other bits are 0. The output of the DAC, therefore, assumes the analog value that corresponds to that code.

If $V_{in} \geq V_D$, the SAR maintains MSB as 1 and sets also the immediately subsequent bit to 1.

If, instead, $V_{in} < V_D$, the SAR sets the MSB to 0 and to 1 the immediately subsequent bit.

- These steps are also repeated for the next bits from the MSB to the LSB.

For example, in a 4-bit resolution converter, at the beginning the SAR is initialised with the binary number 1000, that is to say that the MSB is set to 1, while all the other bits are set to 0. This binary number is converted by the DAC in an analog voltage and is, then, compared to the V_{in}. If V_{in} is greater than or equal to this voltage, the new binary number 1100 is generated, thus meaning that also the bit which is immediately subsequent to the MSB bit is set to 1. If, on the contrary, V_{in} is lower, the new binary number 0100 will be generated, thus meaning that the MSB is set to 0, while the bit immediately subsequent to it is set to 1. As shown in Figure 21, the conversion algorithm proceeds in the same way for the next bits.

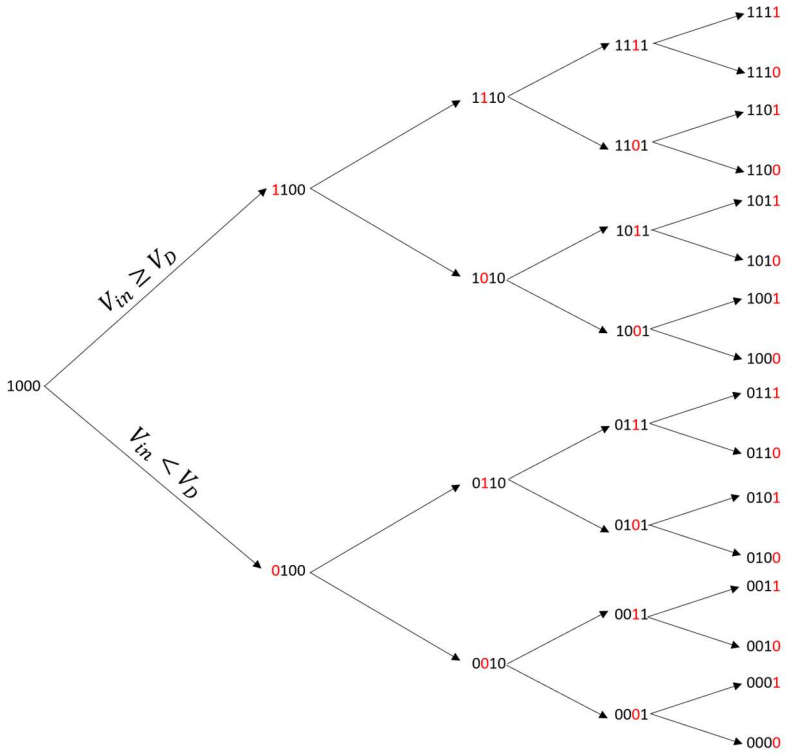

Figure 21 – ADC conversion algorithm. Adapted from [16].

The conversion time of the successive-approximation ADC is independent of the value of the sample of V_{in}. As the resolution of the ADC increases, the conversion time also increases, but this increase can be compensated by increasing the frequency of the clock.

4.2.3 MCP3008

The ADC used in this project is the MCP3008 from Microchip (Figure 22) [17].

Figure 22 – MCP3008 [17].

The MCP3008 10-bit ADC has both high performance and low power consumption (5 nA typical standby and 425 μA typical active), thus being a great choice for embedded control applications [17]. It has a successive approximation register SAR

architecture and an industry-standard SPI serial interface, with a sampling frequency of 200 ksps and 8 input channels.

Figure 23 illustrates the block diagram of the MCP3008.

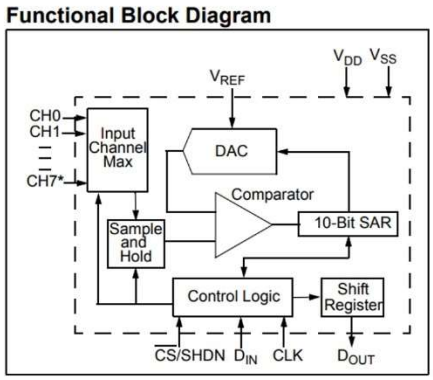

Figure 23 – MCP3008 Functional Block Diagram [17].

Description of the MCP3008 Pins

Figure 24 graphically represents how the pins of the MCP3008 are organised and Table 1 describes their use.

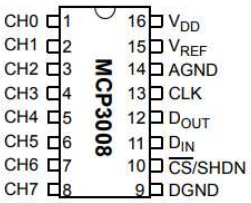

Figure 24 – MCP3008 Pins [17].

Table 1 – MCP3008 Pins Function [17].

MCP3008 PDIP, SOIC	Symbol	Description
1	CH0	Analog Input
2	CH1	Analog Input
3	CH2	Analog Input
4	CH3	Analog Input
5	CH4	Analog Input
6	CH5	Analog Input
7	CH6	Analog Input
8	CH7	Analog Input
9	DGND	Digital Ground
10	$\overline{CS}$/SHDN	Chip Select/Shutdown Input
11	D_{IN}	Serial Data In
12	D_{OUT}	Serial Data Out
13	CLK	Serial Clock
14	AGND	Analog Ground
15	V_{REF}	Reference Voltage Input
16	V_{DD}	+2.7V to 5.5V Power Supply

CH0 - CH7: They are the Analog multiplexed Inputs for the channels 0 - 7 respectively. Each pair of channels can be programmed to be used as two independent channels in single-ended mode. In this case the potential difference that is present between that channel and ground is measured and this is the method used in this project. Another way of proceeding is by using each pair of channels as a pseudo-differential input; in that case the potential difference between the two channels is measured.

V_{DD}: It is the Power supply pin. This pin has to be connected to a voltage supply of 2.7 V to 5.5 V.

V_{REF}: It is the Reference Voltage Input.

AGND: It is the Analog Ground connection for the internal analog circuitry.

CLK: It is the SPI serial Clock pin and is used to put out each bit resulting from the conversion after they are ready. This clock is generated by the master, the Raspberry, in the SPI communication.

D_{OUT}: It is the SPI Serial Data Output pin, the pin where there are one by one the bits resulting from the analog to digital conversion. The data always changes on the falling edge of each clock after the conversion takes place. It is very important to observe that these bits are not the real voltage; instead, it is a digital number that has still to be changed into the real voltage value. In particular, this digital number is:

- 0000000000_b if $V_{in} < \frac{V_{REF}}{1024}$;
- 1111111111_b if, on the contrary, $V_{in} \geq \frac{1023}{1024} \cdot V_{REF}$.

D_{IN}: It is the SPI Serial Data Input pin and is used by the Raspberry to load the configuration data into the ADC.

CS/SHDN: It is the Chip Select/Shutdown pin. When it is set to be low, then the SPI communication can begin. When, on the contrary, it is set to be high, the conversion will end and the ADC will be put in low-power standby. This pin is always 1 and the master sets it to 0 when it has the intention to begin the communication.

DGND: It is the Digital Ground connection for the internal digital circuitry.

4.3 Accelerometer and Gyroscope

This section explores the functioning of the accelerometer and that of the gyroscope, since they are present both on the same sensor module.

4.3.1 Accelerometer

An accelerometer is an electromechanical device that measures the acceleration caused by a force. This force may be static, like the constant force of gravity or dynamic, that causes the moving or vibrating of the accelerometer [18].

The MEMS (micro electromechanical systems) accelerometers are small and this made their applicability increase. The first micro machined accelerometer was designed in 1979 at Stanford University, but it took over 15 years before such devices became accepted mainstream products for large volume applications.

They provide lower power, compact and robust sensing. Multiple sensors are often combined to provide multi-axis sensing.

The acceleration is measured by sensing changes in the capacitance. Capacitive sensing is independent from the base material and relies on the variation of capacitance when the geometry of a capacitor is changing.

A MEMS accelerometer is composed of a movable proof mass with plates that is attached through a mechanical suspension system (two springs) to a reference frame. There are also fixed outer plates which, together with the movable plates of the proof mass, form sets of capacitors.

For parallel-plates capacitors, the capacitance is given by the formula:

$$C = \varepsilon \frac{A}{d}$$, where:

A is the area of the plate;

d is the distance between the two plates;

ε is the permittivity of the dielectric material that separates them;

C is the capacity value expressed in Farad (F).

A change in the parameter d will cause a change of the capacitance and this is used in the MEMS sensing.

As can be seen in Figure 25, all the upper fixed plates are wired parallel and each of them form, with the respective movable plate of the proof mass, capacitors that have a certain capacitance C_1. The same happens with the lower fixed plates and the resulting capacitors have a capacitance C_2.

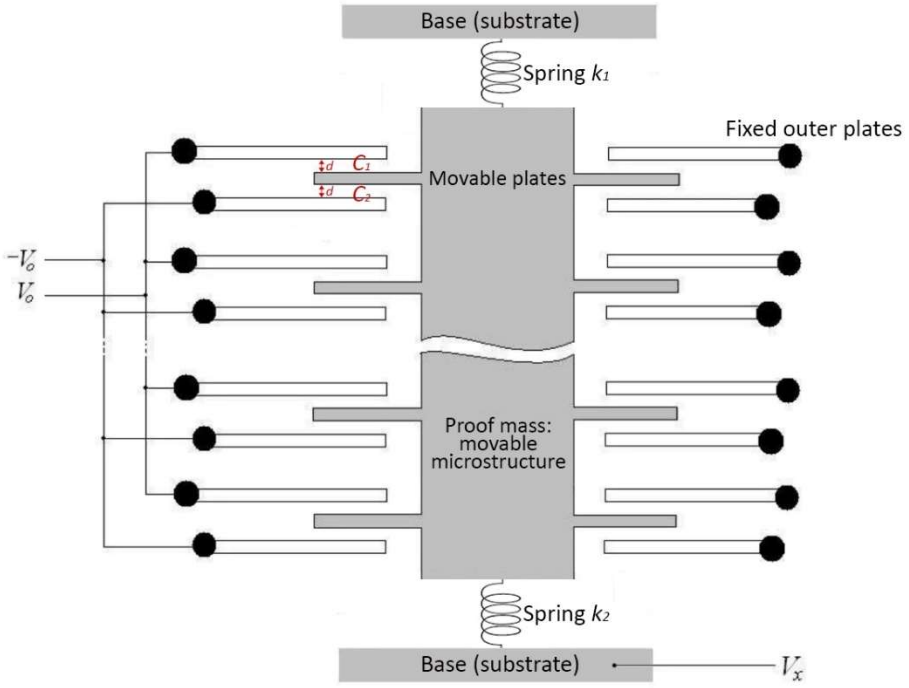

Figure 25 – Structure of the Accelerometer. Adapted from [19].

Now, considering that there is an acceleration on the accelerometer along its sensitive axis as in Figure 26:

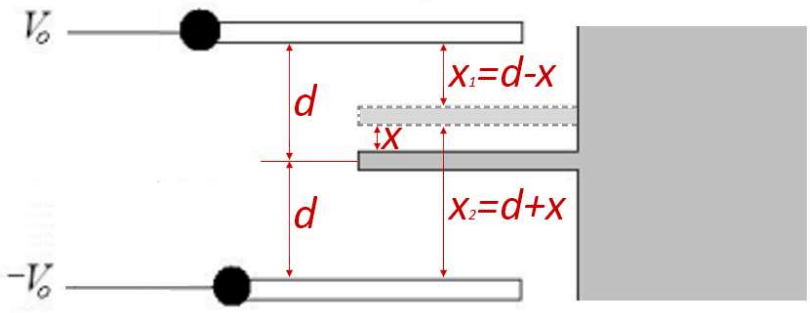

Figure 26 – Acceleration on the accelerometer along its sensitive axis. Adapted from [19].

When the sensor is subject to a linear acceleration along its sensitive axis, the proof mass tends to resist this motion due to its inertia. If an inertial frame of reference is considered, the proof mass and its movable plates will tend to remain still, while the fixed plates will move. Instead, if a non-inertial frame of reference on the fixed plates is considered, the fixed plates will remain still and the proof mass and its movable plates will tend to go to the opposite sense of the external acceleration (but this movement is only fictitious).

This means that the plates are now in a different position in respect to the initial situation and now there are x_1 and x_2 , as there has been a certain displacement x:

$x_1 = d - x$ and $x_2 = d + x$.

Now, considering a parallel-plates capacitor with a voltage source connected to its plates as in Figure 27a:

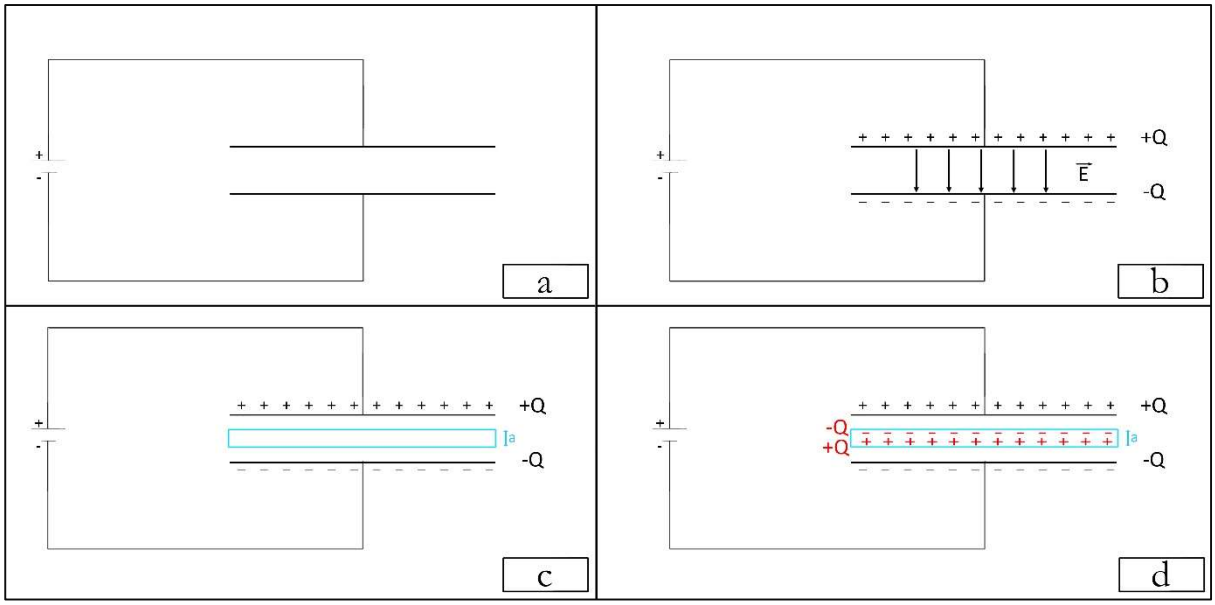

Figure 27 – Analysis of the capacitors in the accelerometer (part 1).

The electrons of the upper plate are attracted by the + of the voltage source, so the upper plate is now charged +Q. Consequently, the electrons of the lower plate are attracted by these positive charges and it becomes charged -Q. An electric field $\vec{E}$ is produced by the charges on the two plates, as shown in Figure 27b.

Now, when inserting a metallic slab with thickness $a \simeq 0$ between the plates, in the electric field produced by this capacitor (Figure 27c).

The electrons of the slab are attracted by the positive charges +Q of the upper plate, so more negative charges will accumulate on the upper side of the surface of the slab, which will become charged -Q, and the other side of the surface of the slab will be left positively charged +Q, thus generating another electric field (Figure 27d).

Now inside the conductor there is the electric field produced by the plates of the capacitor and the electric field within the conductor, which are equal because both are generated by the same charges, but with opposite sense, so the total electric field inside the conductor is $\vec{E}_{TOT} = 0$ (Figure 28a).

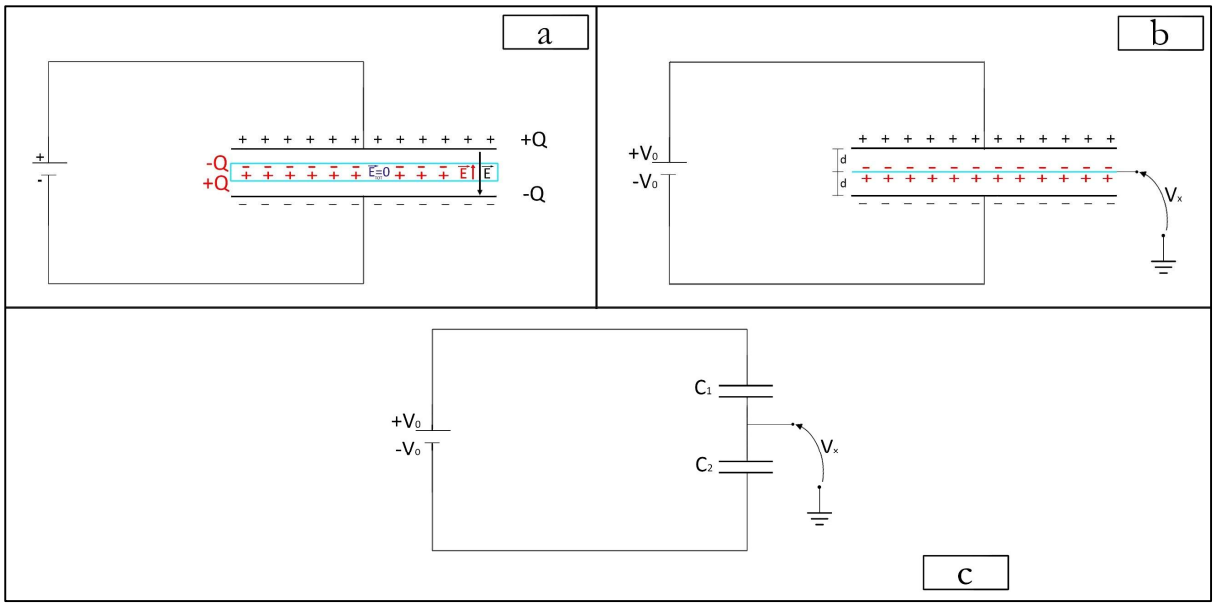

Figure 28 – Analysis of the capacitors in the accelerometer (part 2).

What happens is that the situation represented in Figure 28b is analogous to Figure 28c. This means that now there are two parallel-plates capacitors that have capacity C_1 and C_2 respectively, and they are in series with each other [20].

Since $C_1 = \frac{Q}{\Delta V_1} \Rightarrow Q = C_1 \Delta V_1 = C_1 \cdot (V_0 - V_x)$

Since $C_2 = \frac{Q}{\Delta V_2} \Rightarrow Q = C_2 \Delta V_2 = C_2 \cdot (V_x - (-V_0))$

where V_x is the potential measured at the slab.

$$\Rightarrow C_1 \cdot (V_0 - V_x) = C_2 \cdot \left(V_x - (-V_0)\right) \Rightarrow$$

$$\Rightarrow C_1 \cdot (V_0 - V_x) = C_2 \cdot (V_x + V_0) \Rightarrow$$

$$\Rightarrow C_1 \cdot (V_0 - V_x) = -C_2 \cdot (-V_x - V_0) \Rightarrow$$

$$\Rightarrow C_1 \cdot (V_0 - V_x) + C_2 \cdot (-V_x - V_0) = 0$$

Knowing that $C_1 = \varepsilon \frac{A}{x_1}$ and $C_2 = \varepsilon \frac{A}{x_2}$, and knowing that $x_1 = d - x$ and $x_2 = d + x$, then the previous equation $C_1 \cdot (V_0 - V_x) + C_2 \cdot (-V_x - V_0) = 0$ becomes:

$$\frac{\varepsilon A}{d-x}(V_0 - V_x) + \frac{\varepsilon A}{d+x}(-V_x - V_0) = 0 \Rightarrow$$

$$\Rightarrow \frac{(V_0 - V_x)}{d - x} + \frac{(-V_x - V_0)}{d + x} = 0 \Rightarrow$$

$$\frac{(V_0 - V_x)(d + x) + (-V_x - V_0)(d - x)}{(d - x)(d + x)} = 0 \Rightarrow$$

$$\Rightarrow V_0 d + V_0 x - V_x d - V_x x - V_x d + V_x x - V_0 d + V_0 x = 0 \Rightarrow$$

$$\Rightarrow 2V_0 x = 2V_x d \Rightarrow$$

$$\Rightarrow x = \frac{V_x d}{V_0}$$

The voltage V_x is measured between the slab and ground, the values of d and V_0 are known, so it is possible to calculate the displacement x.

There is a formula that links together the displacement x and the external acceleration, that can be obtained by applying Newton's second law. The sum of all the forces applied on the proof mass is equal to the mass m multiplied by the acceleration of that mass:

$$F_{TOT} = m \cdot a \Rightarrow$$

$\Rightarrow F_{external} + F_{damper} + F_K = m \cdot a$ (in the non-inertial frame of reference fixed on the fixed plates)

where:

$F_{external} = -m \cdot a_{external}$ is the external force applied on the proof mass m;

$F_{damper} = -D \cdot v$ is the force due to the damper, which oppose the movement of the proof mass (hence the sign -), D is the damper constant and v is the velocity of the proof mass. The damping effect is provided by the gas between the plates.

$F_K = -k \cdot x$ is the restoring force due to the springs. The two springs have spring constants k_1 and k_2 respectively, and they can be considered as a single spring that has spring constant $k = k_1 + k_2$. The sign $-$ is due to the fact that this force opposes the movement of the proof mass.

So, the previous equation $F_{external} + F_{damper} + F_K = m \cdot a$ becomes:

$$-m \cdot a_{external} - D \cdot v - k \cdot x = m \cdot a$$

This is the second-order differential equation for a mass-spring-damper system and, considering that $a = \frac{d^2x}{dt^2}$ and that $v = \frac{dx}{dt}$, it can also be written as:

$$-m \cdot a_{external} - D \cdot v - k \cdot x = m \cdot a \Rightarrow$$

$$\Rightarrow m \cdot a + D \cdot v + k \cdot x = -m \cdot a_{external} \Rightarrow$$

$$\Rightarrow m \cdot \frac{d^2x}{dt^2} + D \cdot \frac{dx}{dt} + k \cdot x = -m \cdot a_{external}$$

This second-order differential equation shows the displacement x as a function of the external acceleration that has been applied on the proof mass.

At the equilibrium, after a short transient state, the terms $\frac{d^2x}{dt^2}$ and $\frac{dx}{dt}$ are both equal to 0, so the equation becomes:

$$k \cdot x = -m \cdot a_{external} \Rightarrow$$

$$\Rightarrow a_{external} = \frac{-k \cdot x}{m}$$

When the sensor is subject to an acceleration along its sensitive axis, the proof mass tends to resist this motion due to its inertia, therefore the mass and its plates become displaced respect to the fixed plates, so the external acceleration causes a displacement x. This displacement x, in turn, induces a variation in the capacitances between the moving and fixed plates, which is proportional to the applied external acceleration. This change in the capacitances causes a variation of the voltage V_x , that is measured with a high-resolution ADC and then it is possible to calculate x from the previous formula $\left(x = \frac{V_x d}{V_0}\right)$ and, finally, the external acceleration $\left(a_{external} = \frac{-k \cdot x}{m}\right)$.

4.3.2 Gyroscope

The gyroscope bases its functioning on the Coriolis Effect [21]. The Coriolis force is the fictitious force experienced by an object moving in a non-inertial frame of reference that is fixed on a rotating plane. This force is perpendicular to the direction of motion and to the axis of rotation (Figure 29).

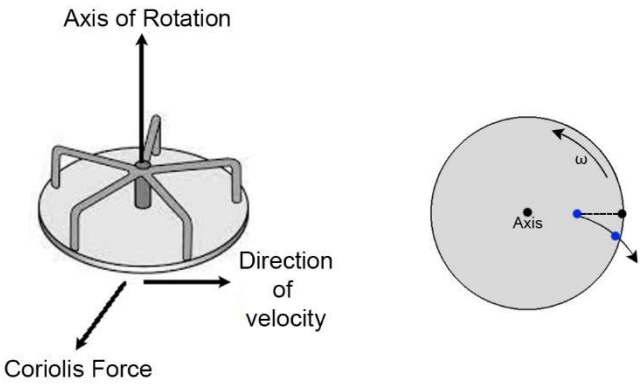

Figure 29 – Direction of the Coriolis force. Adapted from [22].

Figure 30 illustrates a mass m that moves with velocity $\vec{v}$ on the X-axis. In the non-inertial frame of reference, when an angular velocity $\vec{\omega}$ is applied on the plane XY, the Coriolis fictitious force is applied on the mass along the Y-axis. This force is equal to:

$$\vec{F}_{Coriolis} = -2m\vec{\omega} \times \vec{v}$$

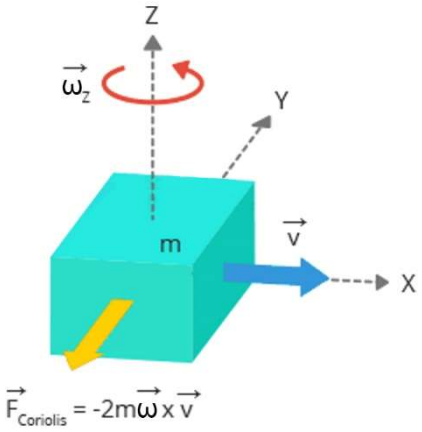

Figure 30 – Coriolis force on one moving mass [21].

Figure 31 represents two masses oscillating in opposite senses along the X-axis at a constant frequency. When an angular velocity is applied, the Coriolis effect is produced on each mass, but in opposite senses.

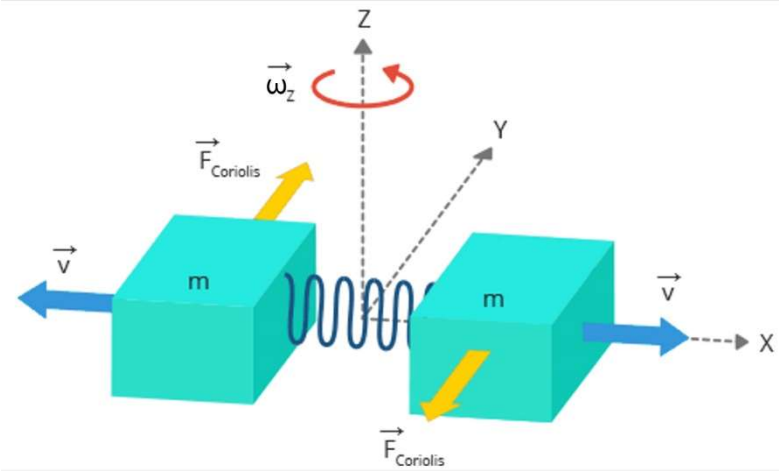

Figure 31 – Coriolis force on two oscillating masses [21].

The MEMS sensor consists of four proof masses: M1 and M3 are maintained in a continuous opposite oscillating movement along the X-axis, while M2 and M4

oscillate along the Y-axis, so that they can respond to the Coriolis effect. These oscillations are illustrated in Figure 32.

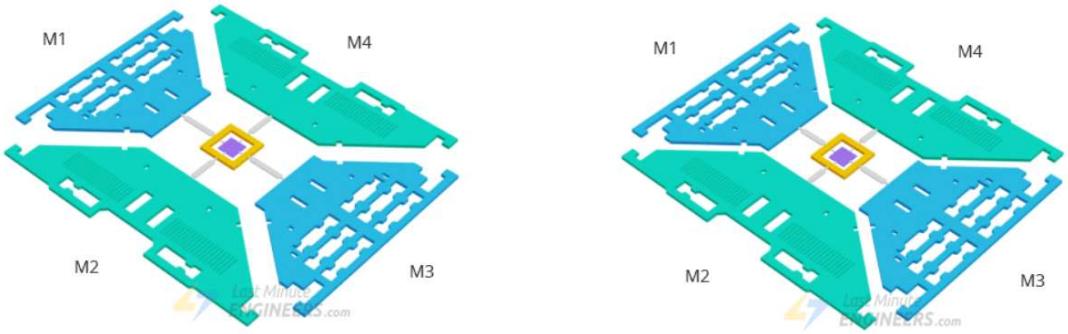

Figure 32 – The four proof masses of the MEMS gyroscope [21].

When the gyroscope starts to rotate, the Coriolis force acts on the moving proof masses.

There are three modes depending on the axis along which the angular rotation is applied.

Roll Mode

When an angular velocity is applied around the X-axis, M1 and M3 will move respectively up and down out of the plane XY due to the Coriolis effect (Figure 33).

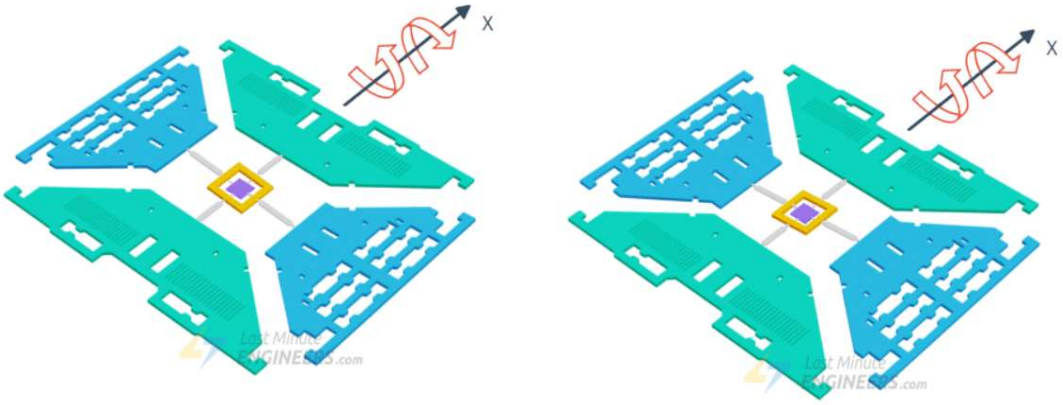

Figure 33 – Roll mode [21].

Pitch Mode

When an angular velocity is applied along the Y-axis, M2 and M4 will move respectively up and down out of the plane XY (Figure 34).

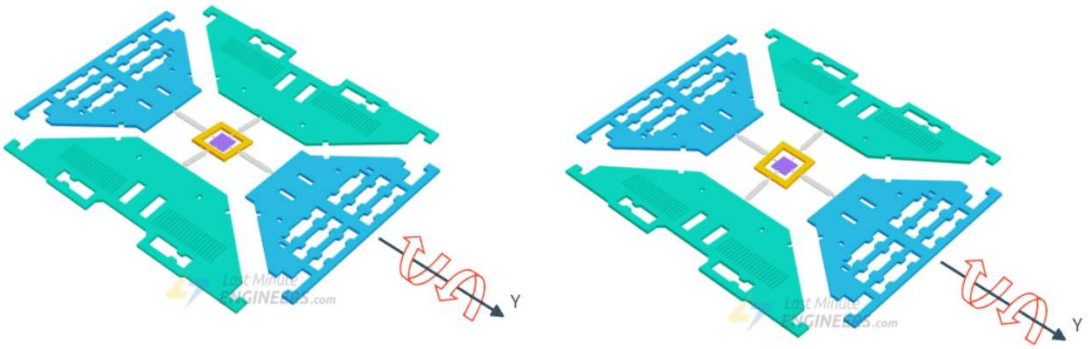

Figure 34 – Pitch mode [21].

Yaw Mode

When an angular velocity is applied along the Z-axis, the Coriolis effect on M1 and M3 will respectively be in opposite sense on the X-axis. On the contrary, the Coriolis effect on M2 and M4 will respectively be in opposite sense on the Y-axis (Figure 35).

Figure 35 – Yaw mode [21].

The movements of the masses due to the Coriolis effect produce a variation of the capacities of set of capacitors, so it is possible to apply the same procedure already described for the accelerometer. In this case, however, the aim is to measure the angular velocity.

4.3.3 MPU-6050

The MPU-6050 sensor module is a complete 6-axis Motion Tracking Device, equipped with a 3-axis Gyroscope, a 3-axis Accelerometer, a Digital Motion Processor, an additional feature of on-chip temperature sensor and a I^2C interface for the communication with the Raspberry (Figure 36) [23] .

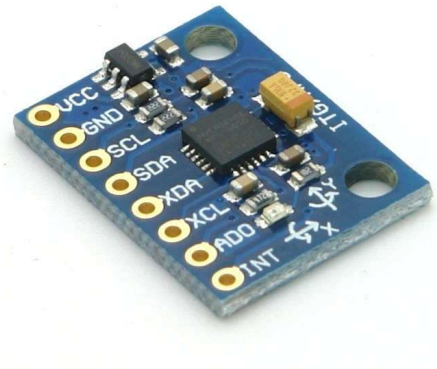

Figure 36 – MPU-6050 Sensor Module [23].

The block diagram of the MPU-6050 is shown in Figure 37.

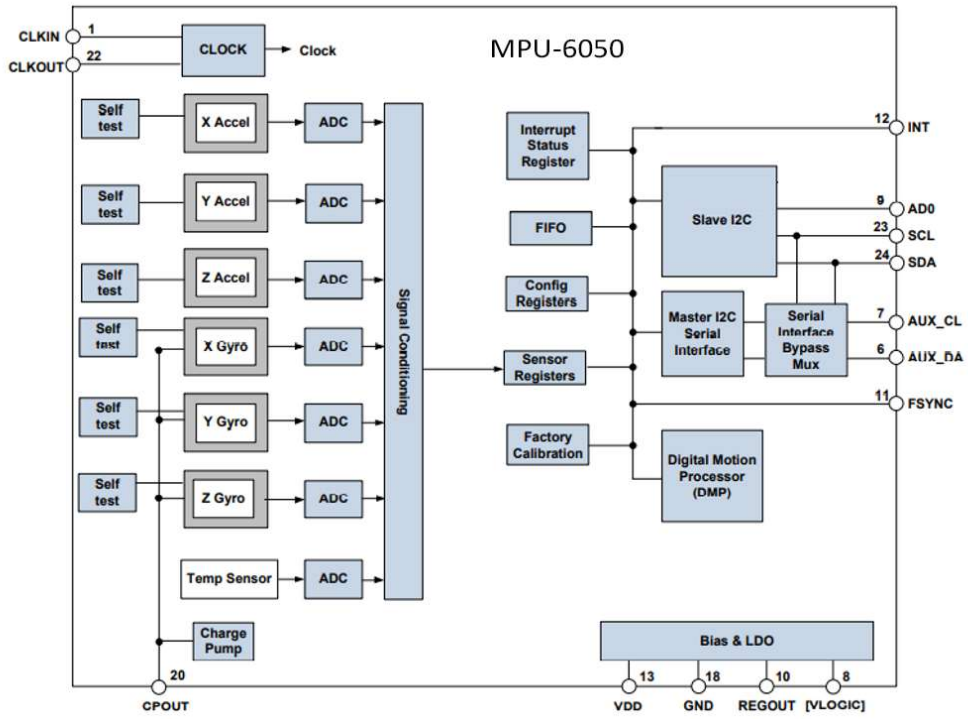

Figure 37 – MPU-6050 Functional Block Diagram [23].

Three-Axis MEMS Accelerometer with 16-bit ADCs

The 3-Axis accelerometer of the MPU-6050 uses separate proof masses for each axis. The acceleration along a particular axis induces a displacement on the corresponding proof mass and the capacitive sensors detect this displacement. When placed on a flat surface, the MPU-6050 measures 0g on the X and Y axes and +1g on the Z axis (Figure 38).

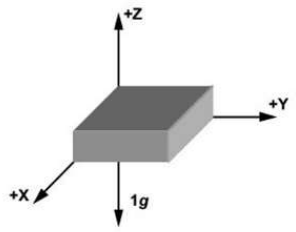

Figure 38 – MPU-6050 3-Axis Accelerometer [23].

The accelerometers' sensitivity scale factor is calibrated at the factory. The full-scale range of the digital output can be adjusted to ±2g, ±4g, ±8g, or ±16g (Table 2). Each sensor has a dedicated ADC for providing digital outputs.

Table 2 – Full-Scale Range and Sensitivity Scale Factor of the Accelerometer [23].

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
ACCELEROMETER SENSITIVITY					
Full-Scale Range	AFS_SEL=0		±2		*g*
	AFS_SEL=1		±4		*g*
	AFS_SEL=2		±8		*g*
	AFS_SEL=3		±16		*g*
ADC Word Length	Output in two's complement format		16		bits
Sensitivity Scale Factor	AFS_SEL=0		16384		LSB/*g*
	AFS_SEL=1		8192		LSB/*g*
	AFS_SEL=2		4096		LSB/*g*
	AFS_SEL=3		2048		LSB/*g*

The output value from each ADC is a 16-bit number in two's complement format.

This value has to be converted to decimal and, thereafter, divided by the sensitivity scale factor, in order to obtain the acceleration expressed in *g*. It is important to observe that there is a certain Sensitivity Scale Factor depending on the chosen Full-Scale Range. Their values are indicated in Table 3.

Table 3 – Sensitivity Scale Factor depending on the chosen Full-Scale Range (Accelerometer) [23].

AFS_SEL	Full Scale Range	LSB Sensitivity
0	±2*g*	16384 LSB/*g*
1	±4*g*	8192 LSB/*g*
2	±8*g*	4096 LSB/*g*
3	±16*g*	2048 LSB/*g*

Three-Axis MEMS Gyroscope with 16-bit ADCs

The MPU-6050 has a vibratory MEMS rate gyroscope aimed at detecting the rotation on the X, Y, and Z axes respectively (Figure 39).

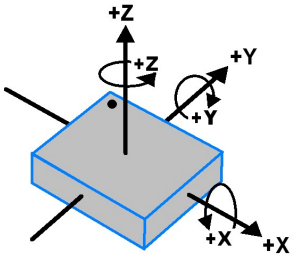

Figure 39 – MPU-6050 3-Axis Gyroscope [23].

When the gyroscope is rotated around any of the sense axes, the Coriolis Effect causes changes in the capacitances and the resulting output is a voltage proportional to the angular velocity. Individual on-chip 16-bit ADCs are used to sample the resulting voltage signal from each axis and to convert it to a digital value, respectively. The full-scale range of the gyroscope can be digitally programmed to ±250, ±500, ±1000, or ±2000 °/s (Table 4).

Table 4 – Full-Scale Range and Sensitivity Scale Factor of the Gyroscope [23].

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
GYROSCOPE SENSITIVITY					
Full-Scale Range	FS_SEL=0		±250		°/s
	FS_SEL=1		±500		°/s
	FS_SEL=2		±1000		°/s
	FS_SEL=3		±2000		°/s
Gyroscope ADC Word Length			16		bits
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)
	FS_SEL=1		65.5		LSB/(°/s)
	FS_SEL=2		32.8		LSB/(°/s)
	FS_SEL=3		16.4		LSB/(°/s)

Also in this case the output value from these ADCs is in two's complement format. Now, as in the previous case, the output value from each ADC has to be divided by the Sensitivity Scale Factor, in order to obtain the result expressed in °/s. There is a

certain Sensitivity Scale Factor depending on the chosen Full-Scale Range. Their values are indicated in Table 5.

Table 5 – Sensitivity Scale Factor depending on the chosen Full-Scale Range (Gyroscope) [23].

FS_SEL	Full Scale Range	LSB Sensitivity
0	± 250 °/s	131 LSB/°/s
1	± 500 °/s	65.5 LSB/°/s
2	± 1000 °/s	32.8 LSB/°/s
3	± 2000 °/s	16.4 LSB/°/s

Digital Motion Processor

The embedded Digital Motion Processor (DMP) located within the MPU-6050 offers computation of motion processing algorithms, with the aim of offloading timing requirements and processing power from the host processor. After the DMP acquires data from accelerometers, gyroscope and additional 3rd party sensors such as magnetometers and processes it, the resulting data can be read from the DMP's registers or can be buffered in a FIFO. The DMP has access to one of the MPU's external pins, which can be used for generating interrupts.

The DMP and FIFO are not used in this project, since the values are directly taken from the registers.

Primary I^2C Interface

The MPU-6050 communicates always as a slave with the Raspberry by using an I^2C serial interface. The I^2C address of the sensor is already predetermined and it is a 7-bit number. If in the I^2C communication two of these sensors are simultaneously used, both of the sensors would have the same I^2C address. To solve this situation, the LSB of the I^2C address can be changed through pin9 (AD0).

Auxiliary I^2C Serial Interface

The MPU-6050 has an auxiliary I^2C bus for communicating to an off-chip 3-Axis digital output magnetometer or other sensors. In this project only the primary I^2C interface is used, and not the auxiliary I^2C interface.

Self-Test

Self-test allows for the testing of the mechanical and electrical portions of the sensors.

Internal Clock Generation

The MPU-6050 has a flexible clocking scheme and various internal or external clock sources can be used for the internal synchronous circuitry.

Sensor Data Registers

The sensor data registers are read-only registers, which are accessed via the serial interface and contain the latest accelerometer, gyroscope, auxiliary sensor and temperature measurement data. The data from these registers can be read anytime and, when new data is available, the interrupt function can be used.

FIFO

The MPU-6050 contains a 1024-byte FIFO register that is accessible via the Serial Interface. The FIFO configuration register determines which data is written into the FIFO, such as accelerometer data, gyroscope data and temperature readings. A FIFO counter says how many bytes of valid data are contained in the FIFO.

Interrupts

The MPU-6050 has a programmable interrupt system, which can generate an interrupt signal on the INT pin. Status flags indicate the source of an interrupt (Table 6). In the register Interrupt Enable, the sources of the interrupt may be enabled and disabled individually.

Table 6 –Interrupt Sources [23].

Interrupt Name	Module
FIFO Overflow	FIFO
Data Ready	Sensor Registers

Digital-Output Temperature Sensor

An on-chip temperature sensor and ADC are used to measure the MPU-6050 temperature. The readings from the ADC can be read from the FIFO or the Sensor Data Registers.

Bias and LDO

The bias and LDO section generates the internal supply and the reference voltages and currents required by the MPU-6050. Its two inputs are VDD of 2.375 to 3.46 V and a VLOGIC logic reference supply voltage of 1.71 V to VDD.

Charge Pump

An on-board charge pump generates the high voltage required for the MEMS gyroscope.

Description of the MPU-6050 Pins

The MPU-6050 module has 8 pins, which are organised as shown in Figure 40.

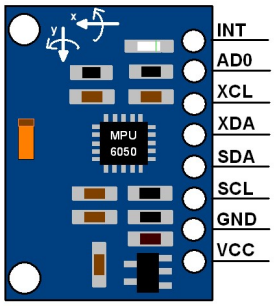

Figure 40 – MPU-6050 Pins [24].

INT: It is the Interrupt digital output pin.

AD0: It is the I²C Slave Address LSB pin. This is 0th bit in the seven bits of the slave address of the device. If this pin is connected to VCC, then it becomes the logic 1 and the slave address changes.

XCL: It is the Auxiliary Serial Clock pin. This pin is used to connect other I²C interface enabled sensors SCL pin to the MPU-6050.

XDA: It is the Auxiliary Serial Data pin. This pin is used to connect other I²C interface enabled sensors SDA pin to the MPU-6050.

SCL: It is the Serial Clock pin. This pin has to be connected to the SCL pin of a microcontroller.

SDA: It is the Serial Data pin. This pin has to be connected to the SDA pin of a microcontroller.

GND: It is the Ground pin. This pin has to be connected to Ground.

VCC: It is the Power supply pin. This pin has to be connected to the +5 V DC supply.

4.3.4 Registers of the MPU-6050

These are the most important registers which the MPU-6050 is configured with [25].

Register 27 – Gyroscope Configuration GYRO_CONFIG

The structure of this register is represented in Figure 41.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1B	27	XG_ST	YG_ST	ZG_ST	FS_SEL[1:0]		-	-	-

Figure 41 – Register 27 – Gyroscope Configuration GYRO_CONFIG [25].

This register is used to trigger the gyroscope self-test and to configure its full-scale range.

XG_ST. Setting this bit causes the gyroscope to perform self-test on the X axis.

YG_ST. Setting this bit causes the gyroscope to perform self-test on the Y axis.

ZG_ST. Setting this bit causes the gyroscope to perform self-test on the Z axis.

FS_SEL. 2-bit unsigned value. It selects the full-scale range of the gyroscope and can assume the values in Table 7.

Table 7 – Full-Scale Range of the Gyroscope [25].

FS_SEL	Full Scale Range
0	± 250 °/s
1	± 500 °/s
2	± 1000 °/s
3	± 2000 °/s

Register 28 – Accelerometer Configuration ACCEL_CONFIG

The structure of this register is represented in Figure 42.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1C	28	XA_ST	YA_ST	ZA_ST	AFS_SEL[1:0]		-		

Figure 42 – Register 28 – Accelerometer Configuration ACCEL_CONFIG [25].

This register is used to trigger the accelerometer self-test and configure the accelerometer full scale range.

XA_ST. When set to 1, the X- Axis accelerometer performs self-test.

YA_ST. When set to 1, the Y- Axis accelerometer performs self-test.

ZA_ST. When set to 1, the Z- Axis accelerometer performs self-test.

AFS_SEL. 2-bit unsigned value. It selects the full-scale range of the accelerometers and can assume the values in Table 8.

Table 8 – Full-Scale Range of the Accelerometers [25].

AFS_SEL	Full Scale Range
0	± 2*g*
1	± 4*g*
2	± 8*g*
3	± 16*g*

Register 56 – Interrupt Enable INT_ENABLE

The structure of this register is represented in Figure 43.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
38	56		-		FIFO _OFLOW _EN	I2C_MST _INT_EN	-	-	DATA _RDY_EN

Figure 43 – Register 56 – Interrupt Enable INT_ENABLE [25].

FIFO_OFLOW_EN. When set to 1, this bit enables a FIFO buffer overflow to generate an interrupt.

I2C_MST_INT_EN. When set to 1, this bit enables any of the I^2C Master interrupt sources to generate an interrupt.

DATA_RDY_EN. When set to 1, this bit enables the Data Ready interrupt, which occurs each time a write operation to all of the sensor registers has been completed.

Register 58 – Interrupt Status INT_STATUS (Read-Only)

The structure of this register is represented in Figure 44.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3A	58	-	MOT_INT	-	FIFO _OFLOW _INT	I2C_MST _INT	-	-	DATA _RDY_INT

Figure 44 – Register 58 – Interrupt Status INT_STATUS (Read-Only) [25].

This register shows the interrupt status of each interrupt generation source. Each bit will clear after the register is read.

MOT_INT. This bit automatically sets to 1 when a DMP interrupt has been generated.

FIFO_OFLOW_INT. This bit automatically sets to 1 when a FIFO buffer overflow interrupt has been generated.

I2C_MST_INT. This bit automatically sets to 1 when an I^2C Master interrupt has been generated.

DATA_RDY_INT. This bit automatically sets to 1 when a Data Ready interrupt is generated.

Registers 59 to 64 – Accelerometer Measurements

The structure of these registers is represented in Figure 45.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Figure 45 – Registers 59 to 64 – Accelerometer Measurements [25].

These registers store the most recent accelerometer measurements.

ACCEL_XOUT. 16-bit Two's complement value. It stores the most recent X axis accelerometer measurement.

ACCEL_YOUT. 16-bit Two's complement value. It stores the most recent Y axis accelerometer measurement.

ACCEL_ZOUT. 16-bit Two's complement value. It stores the most recent Z axis accelerometer measurement.

Registers 65 and 66 – Temperature Measurement

The structure of these registers is represented in Figure 46.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
41	65	TEMP_OUT[15:8]							
42	66	TEMP_OUT[7:0]							

Figure 46 – Registers 65 and 66 – Temperature Measurement [25].

These registers store the most recent temperature sensor measurement.

TEMP_OUT. 16-bit value. It stores the most recent temperature sensor measurement.

Registers 67 to 72 – Gyroscope Measurements

The structure of these registers is represented in Figure 47.

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Figure 47 – Registers 67 to 72 – Gyroscope Measurements [25].

These registers store the most recent gyroscope measurements.

GYRO_XOUT. 16-bit Two's complement value. It stores the most recent X axis gyroscope measurement.

GYRO_YOUT. 16-bit Two's complement value. It stores the most recent Y axis gyroscope measurement.

GYRO_ZOUT. 16-bit Two's complement value. It stores the most recent Z axis gyroscope measurement.

4.4 Pulse Oximeter and Heart Rate Detector

The Heart Rate detection is based on Photoplethysmography (PPG), which is a non-invasive optical technique that uses the light to detect in peripheral circulation the changes in the volume of the blood, which happen due to the cardiovascular pulsations [26]. This method is based on the absorption of the light by the blood and by the tissues and requires a light source and a light detector to function. The Heart Rate is monitored by using a Heart Rate detector.

The SpO2 indicates the Saturation of peripheral Oxygen level and its detection also makes use of the Photoplethysmography. However, in the case of the Pulse oximetry, it requires the use of two light sources with different wavelengths and a light detector. The SpO2 is monitored by using a Pulse Oximeter.

4.4.1 Heart and blood circulation

The heart is composed of four chambers (Figure 48): two on top (called left and right atrium respectively), separated by a wall called interatrial septum and two on bottom (called left and right ventricle), separated by a wall called interventricular septum [27].

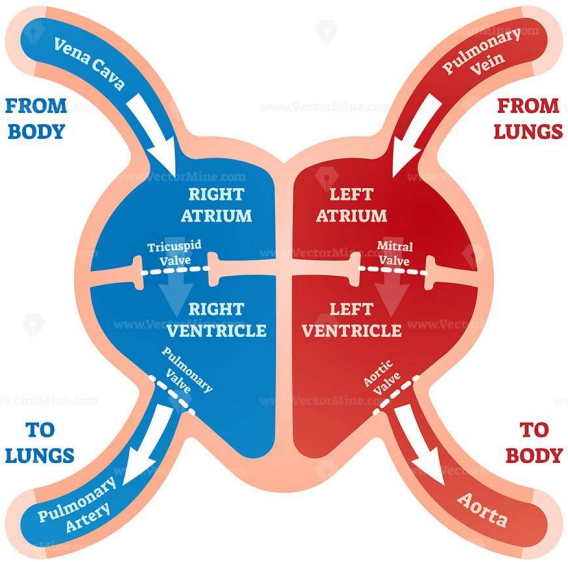

Figure 48 – Heart structure [28].

Moreover, the mitral valve separates the left atrium from the left ventricle, while the tricuspid valve separates the right atrium from the right ventricle.

The blood, thanks to the heart, circulates in the entire organism to supply the useful substances and to take away from it the waste substances.

The blood circulation happens in two different flows: the systemic and the pulmonary circulation, which are illustrated in Figure 49.

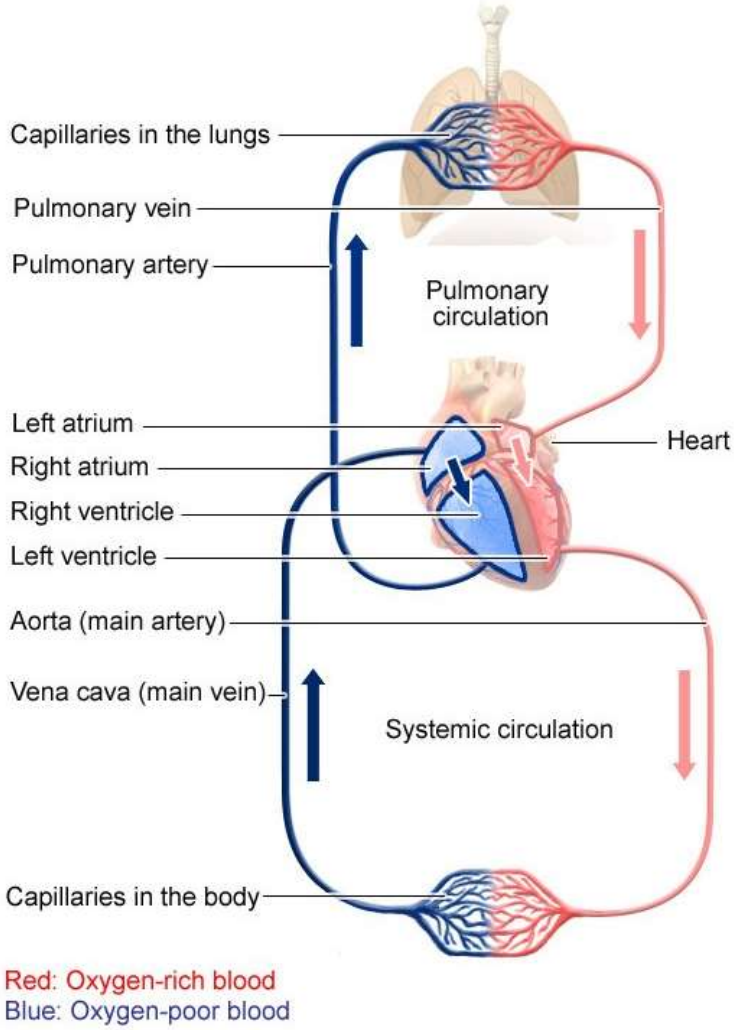

Figure 49 – Systemic and pulmonary circulations [29].

Systemic circulation

Regarding the systemic circulation, it starts in the left atrium, which pushes the oxygenated blood through the mitral valve into the left ventricle [27]. This event is called atrial systole. Subsequently, the left ventricle pushes the oxygenated blood through the aortic valve, which is between the left ventricle and the aorta and the blood goes to the aorta. This event is called ventricular systole. The aorta, then, branches off into smaller arteries, some of which reach the superior parts of the body, while the others reach the inferior parts of the body. These smaller arteries get even smaller and they branch off into the arterioles and finally into the capillaries,

which reach all the parts of the body. In the arterial capillaries, the arterial blood gives nutrients and oxygen to the cells and takes from them the waste products. Subsequently, the blood goes through the venous capillaries. The venous capillaries then lead into small veins, which then lead to larger and larger veins as the blood approaches the heart and they finally lead to two large veins, superior vena cava and inferior vena cava, into the right atrium of the heart. The names superior and inferior are due to their positions above and below the heart, respectively.

Pulmonary circulation

As regards the pulmonary circulation, the right atrium of the heart pushes the venous blood (rich in carbon dioxide) through the tricuspid valve into the right ventricle. Subsequently, the right ventricle pushes the venous blood through the pulmonary valve, which is between the right ventricle and the pulmonary artery [27]. The pulmonary artery reaches the lungs and it branches off into smaller arteries, which lead to the capillaries that are in the pulmonary alveolus, which are small air sacks into the lungs. Here the venous blood gives the carbon dioxide in order to breathe it out and takes the oxygen. Finally, the arterial blood goes through the venous capillaries towards always bigger veins until the pulmonary vein that reaches the left atrium of the heart.

It is important to notice that in the systemic circulation the arterial blood is in the arteries and the venous blood is in the veins, but in the pulmonary circulation the venous blood is in the arteries (from the heart towards the lungs), while the arterial blood is in the pulmonary veins (from the lungs towards the heart).

Veins and arteries are not distinguished based on which blood flows through them, but based on their structure and position.

In the human blood circulation, the two blood flows never stop and the circulation is double, as the arterial and the venous blood are never mixed and it is complete, as the two flows happen at the same time.

Systole and Diastole

There are two types of systoles: the atrial systole, which corresponds to the contraction of the atria and has the purpose of pushing the blood from the atria towards the ventricles and the ventricular systole, which corresponds to the contraction of the ventricles and has the purpose of pushing the blood from the ventricles to the blood vessels. Similarly, there are two types of diastoles: the atrial diastole, which is the consequent relaxing of the atria before the new atrial systole and the ventricular diastole, which is the relaxing of the ventricles before the new

ventricular systole [30]. Therefore, systole and diastole come in succession as in Figure 50.

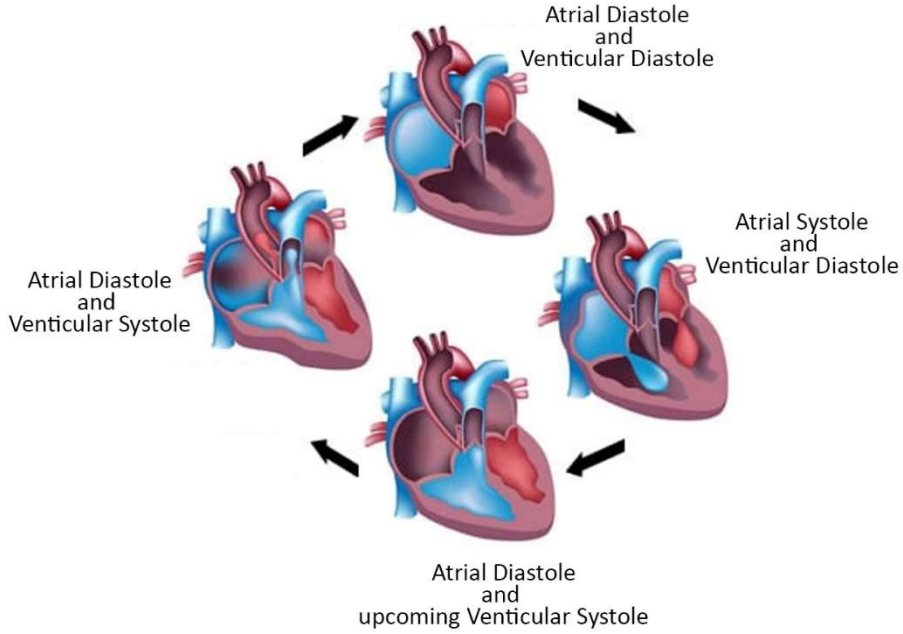

Figure 50 – Cardiac cycle. Adapted from [30].

At first there are an atrial and a ventricular diastole, next there is an atrial systole while the ventricular diastole is continuing, then there are an atrial and a ventricular diastole and lastly, before the beginning of a new cycle, there are an atrial diastole and a ventricular systole.

4.4.2 Analysis of the absorbed light

Photoplethysmography devices have two LEDs that emit at different wavelengths: one Infrared (IR) LED emits a light with wavelength of 940 nm and the other red LED emits a light with the wavelength of 660 nm [31]. The two lights are always shot one at a time. The finger of the patient has to be placed on the sensor on both the two LEDs and the light detector, as indicated in Figure 51.

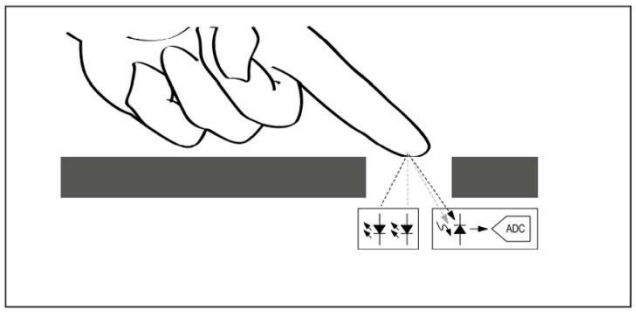

Figure 51 – Finger on the sensor. Adapted from [31].

The light arrives at the finger, where it is partly absorbed, while some of it is reflected and detected by the light detector. Since the only blood that flows into the finger is that of the systemic circulation, in this specific case the arterial blood is in the arteries and the venous blood is in the veins.

By analysing the intensity of one of the detected reflected lights (for example the infrared light), it is possible to make observations on the absorbed light.

Figure 52 indicates how the intensity of the absorbed light changes with the time.

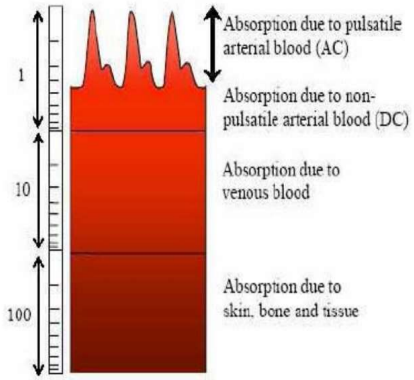

Figure 52 – AC and DC components of the absorbed light in a finger [32].

The DC component of the signal is primarily the light absorbed by the skin, the bones and the tissues [32].

Another DC component is the light absorbed by the venous blood. In the systemic circulation, in fact, the blood that flows in the veins (which goes towards the heart) is non-pulsatile, as it is never pushed back by the heart, but it is a smooth flow with low pressure and returns to the heart only because of its diastole.

Finally, the other DC component of the absorbed light is due to the non-pulsatile arterial blood. The arteries are, in fact, never empty, as there is always a certain constant amount of arterial blood in them.

The AC component of the signal consists of the absorption by the pulsating arterial blood (the heart pumps and this is how the blood pulsates).

The above also happens for the red light.

4.4.3 Heart Rate

To detect the Heart Rate it is not necessary to use both of the LEDs, but only the IR light. The AC component depicts the changes in the volume of the blood, which are caused by the cardiac activity and depend on the systolic and diastolic phases. The ventricular systolic phase (also called "rise time") starts with a valley and ends in correspondence of the dicrotic notch [33]. The dicrotic notch is the beginning of

a secondary upstroke in the descending part. It is caused by the closure of the aortic valve, which produces a transient increase in the volume of the blood. The diastolic phase starts immediately after the dicrotic notch and ends just before the start of the new wave. These phases are represented in Figure 53.

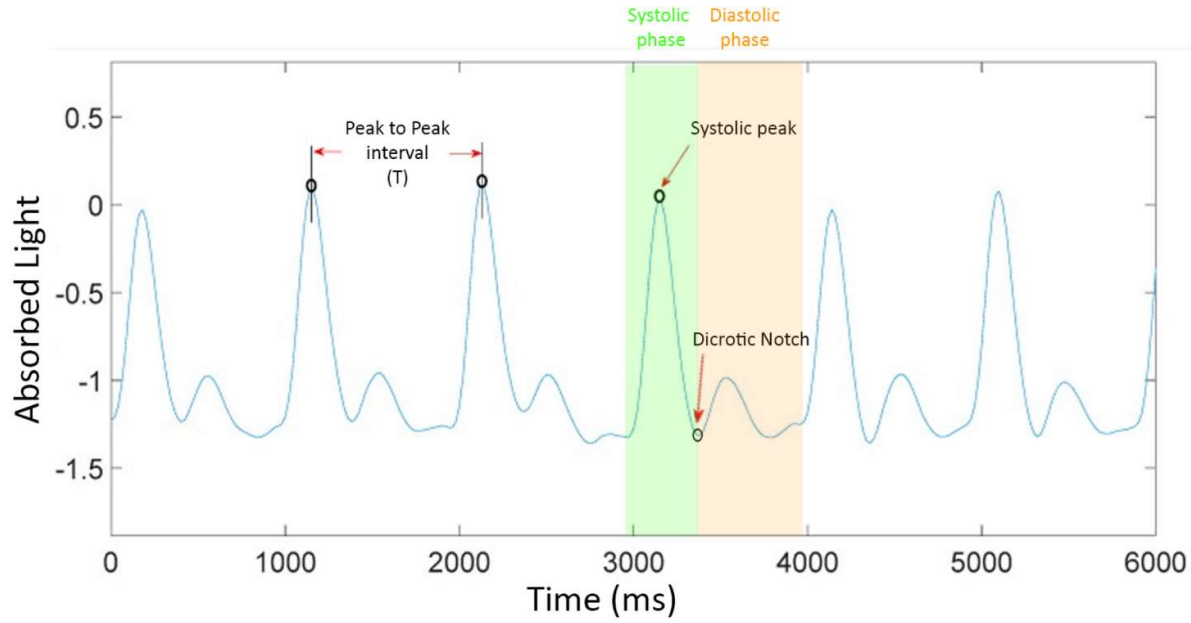

Figure 53 – Heart Rate measurement. Adapted from [33].

The frequency of a signal indicates the number of waves in a second. In this specific case each wave has only one peak, which is caused by a heartbeat. This means that in this situation the frequency indicates the number of heartbeats per second (or minute with the necessary conversion), which is exactly the output of the Heart Rate detector. The frequency is given by $\frac{1}{T}$ and the period T in this case is the peak-to-peak interval.

4.4.4 Oxygen Saturation

The Oxygen saturation indicates the percentage of haemoglobin molecules that are carrying oxygen through the body with respect to the total haemoglobin. In the lungs there are millions of microscopic air sacs called alveoli. The walls of the alveoli share a membrane with the capillaries, a network of small blood vessels, allowing the diffusion of oxygen from the lungs into the bloodstream and carbon dioxide molecules from the bloodstream in the opposite sense. When oxygen molecules pass through the alveoli, these bind to the haemoglobin molecules, thus causing the blood to be saturated with oxygen. As the haemoglobin circulates, the haemoglobin gives the oxygen to the cells of the body, while it receives from them the carbon dioxide. Thereafter, the carbon dioxide is transported back to the alveoli, so the cycle can begin again.

The normal oxygen saturation for most healthy adults is usually between 95% and 100% and a lower level indicates that organs, tissues and cells aren't receiving the amount of oxygen they need to function properly. Such situation needs immediate medical attention [26].

The estimation of oxygen saturation (SpO2) is defined as the proportion of oxyhaemoglobin (HbO2) in the total haemoglobin measured.

$$SpO_2 = \frac{HbO_2}{HbO_2 + Hb} \times 100$$

where:

SpO_2 is the Oxygen Saturation;

HbO_2 is the Oxyhaemoglobin, that is to say the saturated haemoglobin;

Hb is the De-Oxyhaemoglobin, that is to say the non-saturated haemoglobin.

This formula is the theoretical definition of the SpO2, that describes the percentage of saturated haemoglobin with respect to the total haemoglobin. To obtain the value of SpO2 it is necessary to use both of the two LEDs.

The absorption of the two different lights (with different wavelengths) differs significantly between blood loaded with oxygen and blood lacking oxygen (Figure 54).

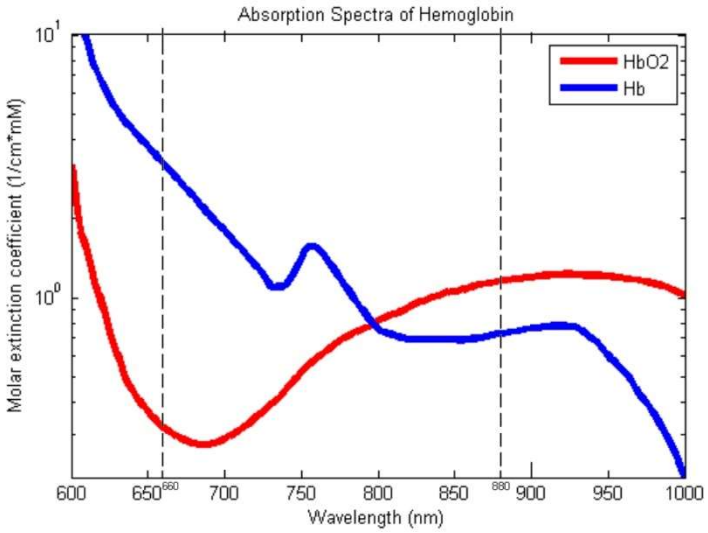

Figure 54 – Absorption of Haemoglobin as a function of Wavelength. Adapted from [34].

The infrared light is absorbed more by the oxygenated blood and less by the deoxygenated blood, while for the red light it is exactly the opposite.

On this principle the new parameter R is defined [26]:

$$R = \frac{AC_{660nm}/DC_{660nm}}{AC_{880nm}/DC_{880nm}}$$

Knowing the value of R, it is possible to use the graph in Figure 55 to obtain the value of the SpO2 [35].

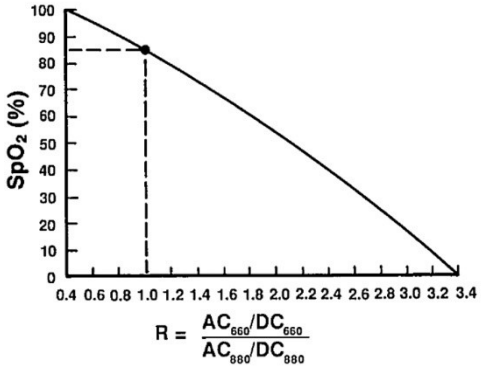

Figure 55 – Relation of R and SpO2 [35].

4.4.5 MAX30100

The MAX30100 [31] is the integrated Heart Rate and Pulse Oximeter sensor solution that was selected for use in the project's prototype. It combines two LEDs, a photodetector, optimized optics and low-noise analog signal processing to detect the HR and the SpO2. This device is used in Wearable Devices, Fitness Assistant Devices and Medical and Close-to-Medical Monitoring Devices. The MAX30100 is fully configurable through software registers. The device is equipped with a 16-cells FIFO in which the digital output data (both IR and RED ADC data) is stored. Each sample consists of one IR word (2 bytes) and one RED word (2 bytes), so there are 4 bytes of data for each sample, and therefore, 16·4=64 total bytes of data can be stored in the FIFO. Figure 56 shows the structure of the FIFO graphically.

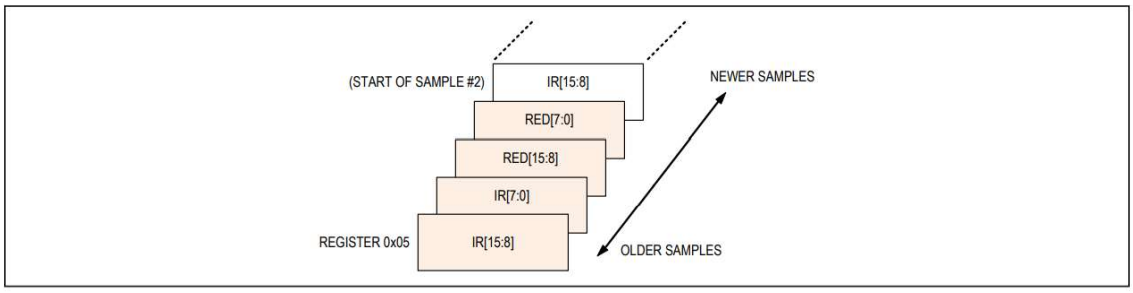

Figure 56 – Graphical Representation of the FIFO Data Register [31].

The FIFO is managed through the write pointer and the read pointer. The write pointer is the register that contains the value of the cell of the FIFO where the sensor has to write. The read pointer is the register that contains the value of the cell of the FIFO from where the Arduino has to read.

The write pointer increments every time a new sample is added to the FIFO, while the read pointer is incremented automatically every time a sample is read from the FIFO.

Figure 57 is the functional diagram of the MAX30100.

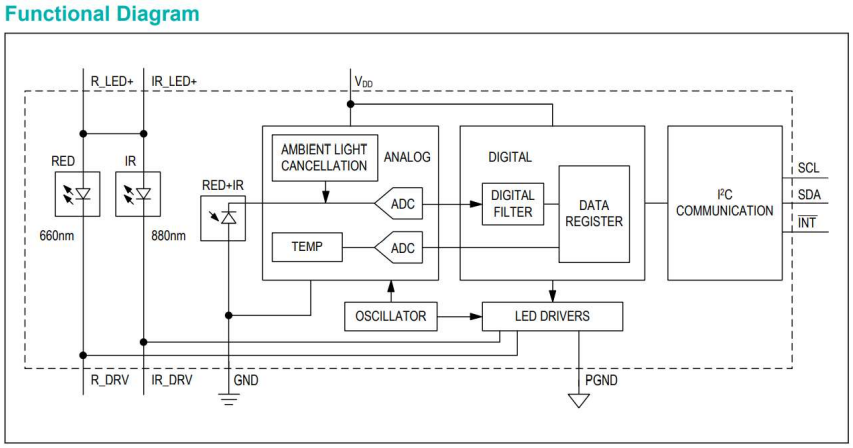

Figure 57 – MAX30100 Functional Block Diagram [31].

SpO2 Subsystem

The SpO2 subsystem in the MAX30100 is composed of Ambient Light Cancellation (ALC) and an ADC with up to a 16-bit resolution.

Temperature Sensor

The MAX30100 has an on-chip temperature sensor for optionally calibrating the temperature dependence of the SpO2 subsystem.

LED Driver

The MAX30100 integrates infrared and red LED drivers to drive LED pulses for HR and SpO2 measurements. The LED current can be programmed from 0 mA to 50 mA. The LED pulse width can be programmed from 200 μs to 1.6 ms.

Description of the MAX30100 Pins

The MAX30100 module has 5 pins, which are organised as illustrated in Figure 58.

Figure 58 – MAX30100 Pins.

VIN: It is the Power supply pin. This pin has to be connected to the +3.3 V DC supply.

GND: It is the Ground pin. This pin has to be connected to Ground.

SCL: It is the Serial Clock pin. This pin has to be connected to the SCL pin of a microcontroller.

SDA: It is the Serial Data pin. This pin has to be connected to the SDA pin of a microcontroller.

INT: It is the Interrupt digital output pin.

4.4.6 Registers of the MAX30100

Below there are the main registers of the MAX30100 [31].

Interrupt Status (0x00)

This is an 8-bit read-only register, whose address is 0x00. The structure of this register is represented in Figure 59.

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Interrupt Status	A_FULL	TEMP_ RDY	HR_RDY	SPO2_ RDY				PWR_ RDY	0x00	0X00	R

Figure 59 – Interrupt Status Register [31].

Each bit indicates the status of a certain interrupt.

Bit 7: FIFO Almost Full Flag (A_FULL). This interrupt triggers when the FIFO write pointer has the value of the FIFO read pointer minus one, which means that the FIFO has only one unwritten space left. If the FIFO is not read within the next conversion time, the FIFO becomes full and the future data is lost.

Bit 6: Temperature Ready Flag (TEMP_RDY). This interrupt triggers when an internal temperature conversion is finished and the temperature data registers can be read.

Bit 5: Heart Rate Data Ready (HR_RDY). This interrupt triggers after every data sample is collected. A heart rate data sample consists of one IR data point only. This bit is automatically cleared when the FIFO data register is read.

Bit 4: SpO2 Data Ready (SPO2_RDY). Only when the SpO2 mode is enabled, this interrupt triggers after every data sample is collected. An SpO2 data sample consists of one IR data and one RED data. This bit is automatically cleared when the FIFO data register is read.

Bit 3, Bit 2, Bit 1. RESERVED

Bit 0: Power Ready Flag (PWR_RDY). This interrupt is triggered on power-up or after a brownout condition, when the supply voltage VDD assumes values from below the UVLO voltage to above the UVLO voltage, thus indicating that the sensor is powered up and ready to collect data.

When there is an interrupt, the pin INT becomes 0.

Interrupt Enable (0x01)

The structure of this register is represented in Figure 60.

REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
Interrupt Enable	ENB_A_FULL	ENB_TEMP_RDY	ENB_HR_RDY	ENB_SO2_RDY					0x01	0X00	R/W

Figure 60 – Interrupt Enable Register [31].

Each source of interrupt, with the exception of power ready, can be disabled in this register. When an interrupt enable bit is set to zero, the corresponding interrupt appears as 1 in the interrupt status register, but the INT pin is not pulled low.

4.5 Dual-Purpose Button

Pin16 of the Raspberry was chosen as an input pin and was configured in the software, so that the Raspberry could internally create the connection of pin16 through an internal pull-up resistor to an internal voltage of 3.3 V. In the architecture, a button was connected between pin16 and ground (Figure 61).

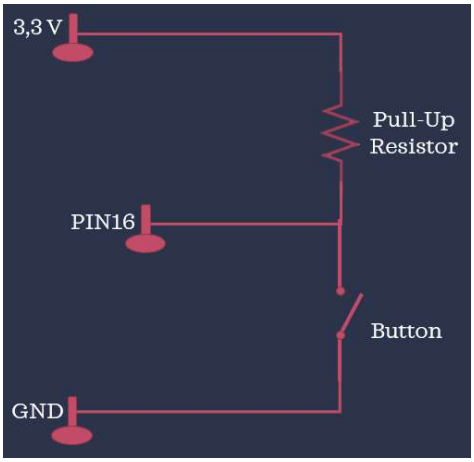

Figure 61 – Connection of the Button.

When the button is not pressed, 3.3 V are present on pin16 due to the pull-up resistor. If the button is pressed, on PIN16 there are 0 V (GND). Moreover, in this case there is a closed circuit and the Pull-Up Resistor is necessary to limit the value of the current.

This single button implements two different functionalities: an emergency button and a confirmation button to respond to certain system actions.

By proceeding this way, it was not necessary to use two separate buttons, which can be confused by the elderly patient.

Normally, the button is an emergency button so, if the patient presses the button, the request for emergency is sent to Akenza (either way via Wi-Fi and Internet or via LoRaWAN). In this case, as soon as the emergency reaches Akenza, the latter generates an email of warning (Figure 62) and sends it to the doctor and/or to the patient's relatives.

Figure 62 – Email due to Akenza Rule.

On the contrary, when the button is a confirmation button, the Raspberry must not send any emergency.

4.6 Edge Computing

Edge computing is the computing that takes place at or near the physical location of the source of the data, hence it results in lower latency [36]. Since the computing happens close to the data source, in fact, the data produced by IoT sensors and devices can be quickly analysed, without the need to wait for the data to travel to a central site where that analysis can take place. This results in a faster possibility of action. Moreover, the edge computing also saves bandwidth, because without it all the raw data is put in the network. On the contrary, by using edge computing, some raw data is already analysed and only the resulting data is put in the network.

In the developed system the edge computing is made by the Raspberry Pi, which also uses its Wi-Fi connection to send data online. Moreover, the LoRa32 is also used for the transmission through LoRaWAN. Besides these two devices, also the Arduino Uno takes part in the edge computing.

4.6.1 Raspberry Pi

The core of the developed system is the Raspberry Pi 4 Model B, represented in Figure 63.

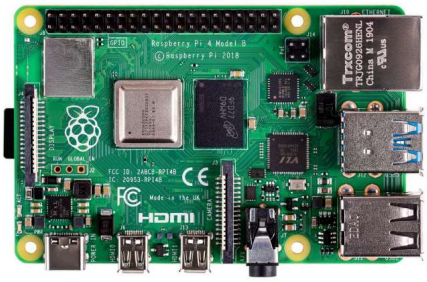

Figure 63 – Raspberry Pi 4 Model B [37].

The Raspberry Pi is a credit card-sized computer from Raspberry and it is the third best-selling computer brand in the world [38].

More precisely, the model used in this project is the Raspberry Pi 4 Model B, which is one of the latest products in the Raspberry Pi range of computers and will remain in production until at least January 2026. Compared to the prior-generation Raspberry Pi 3 Model B+, the Raspberry Pi 4 Model B offers great increases in the processor speed, multimedia performance, memory and connectivity, while maintaining the compatibility and similar power consumption (Table 9) [39].

Table 9 – Comparation of the Raspberry Pi 4 Model B and the Raspberry Pi 3 Model B+ [39].

Product	Pi 4 Model B	Pi 3 Model B+
SoC	BCM2711	BCM2837B0
Speed	1500 MHz	1400 MHz
RAM	8 GB	1 GB
USB Ports	2xUSB / 2xUSB3	4
Ethernet	1000Base-T	1000Base-T
Wireless	802.11ac/n	802.11ac/n
Bluetooth	5.0	4.2

It also has a Micro SD card slot for loading operating system and for data storage. The Raspberry is powered by 5 V DC through the USB-C connector or by 5 V DC through the GPIO header.

A very important feature of the Raspberry Pi is the presence of a row of 40 GPIO (general-purpose input/output) pins along its top edge as shown in Figure 64 [40].

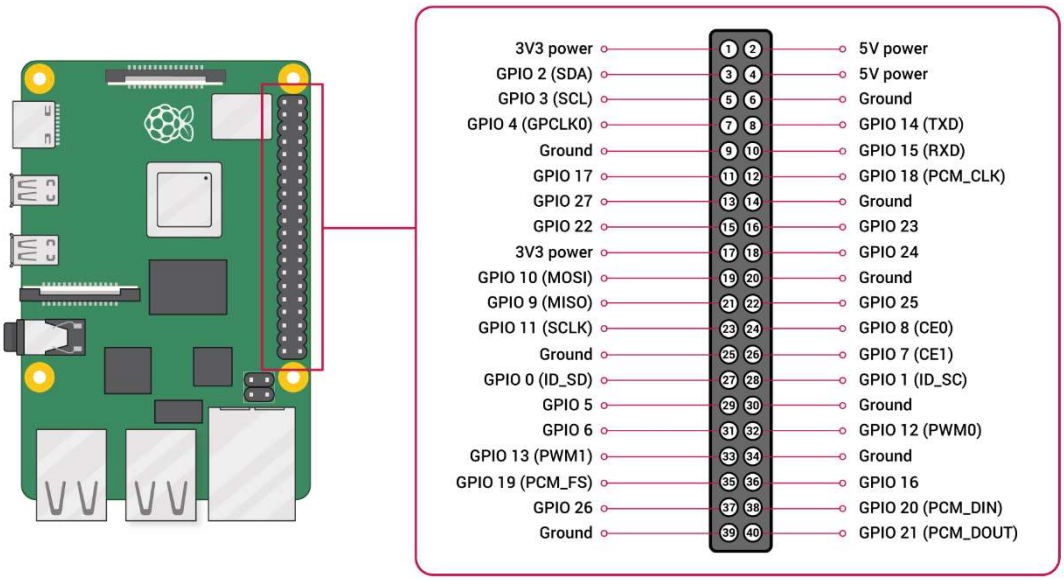

Figure 64 – Raspberry Pins [40].

Among its 40 pins, the Raspberry has two 5 V pins and two 3.3 V pins, as well as ground pins (0V), which are unconfigurable. It is possible to set in software the configurable GPIO pins to be input or output pins. A GPIO pin designated as an

output pin can be set to high (3V3) or low (0V), while a GPIO pin designated as an input pin can be read as high (3V3) or low (0V). Some specific functions require the use of certain specific GPIO pins (Table 10), while for other functions it is possible to use all the pins indifferently.

Table 10 – Specific functions on specific Raspberry pins.

PWM (pulse-width modulation)
Software PWM available on all the configurable pins Hardware PWM available on GPIO12 (PWM0), GPIO13 (PWM1)
SPI
GPIO10 (MOSI), GPIO9 (MISO), GPIO11 (SCLK), GPIO8 (CE0), GPIO7 (CE1)
I^2C
GPIO2 (SDA), GPIO3 (SCL)
Serial (UART)
GPIO14 (TXD), GPIO15 (RXD)

4.6.2 LILYGO TTGO LoRa32

Since the Raspberry does not offer the possibility to have LoRa communication, the usage of an external device was required for this purpose. After an analysis of the devices available on the market, the LILYGO TTGO LoRa32 (Figure 65) was selected to be used in the developed system prototype [41].

TTGO was the previous name of the team, that now took the name LILYGO.

Figure 65 – LILYGO TTGO LoRa32.

There are many versions of this module; the T3 model, version 2.1 revision 1.6, (which on the module is indicated as T3_V1.6) was used in this project. Table 11 presents its main features [42].

Table 11 – Main features of the LILYGO TTGO LoRa32 T3 model [42].

Working voltage:	1.8~3.7 V
Transmit current:	120 mA@+20 dBm 90 mA@+17 dBm 29 mA@+13 dBm
Operating frequency:	868 MHz/915 MHz
Transmit power:	+20 dBm
Receive sensitivity:	-136dBm @LoRa & B=125kHz & SF=12 -118dBm @LoRa & B=125kHz & SF=6
FIFO space:	64 bytes
Sleep current:	0.2 uA @ SLEEP 1.5 uA @ IDLE
Operating temperature:	-40℃- +85℃

The device is composed of five parts, as shown in Figure 66.

Figure 66 – Rear and frontal view of the LoRa32 T3 model.

- The core of this device is the ESP32-PICO-D4. It is a System-in-Package (SiP), which is a module consisting of two or more electronic components enclosed in a carrier package. Its main component is the SoC ESP32, which is analysed in detail in the next paragraph.

The LoRa module, which is connected to the ESP32 with SPI communication through the pins listed in Table 12.

Table 12 – Connection of the LoRa Module Pins within the LoRa32.

IO18 = SS
IO5 = SCLK
IO27 = MOSI
IO19 = MISO

A 0.96 inches OLED display connected to the ESP32 via I^2C (GPIO21 as SDA and GPIO22 as SCL).

- An antenna for both Wi-Fi and Bluetooth, which is already assembled on the module.
- An antenna for LoRa, which can be screwed on the module through a SMA connector.

SoC ESP32

The ESP32 is a low-cost, low-power system on a chip (SoC) [43]. A SoC is an integrated circuit that integrates all the components of an electronic system, including a microprocessor, on a chip. It is equipped with Wi-Fi and Bluetooth capabilities and was created by Espressif Systems for mobile, wearable electronics and Internet-of-Things (IoT) applications. At its core, the ESP32 has a dual-core or single-core Tensilica Xtensa LX6 microprocessor with a clock rate of up to 240 MHz.

It is possible to wake up the ESP32 periodically if a certain condition is detected. Since the ESP32 has been specifically designed for mobile devices, wearable electronics and IoT applications, it is able to achieve ultra-low power consumption.

4.6.3 Arduino Uno R4 Minima

The Arduino UNO R4 Minima is represented in Figure 67 [44]. It is based on the 32-bit microcontroller from Renesas, which uses a 48 MHz Arm Cortex-M4 microprocessor.

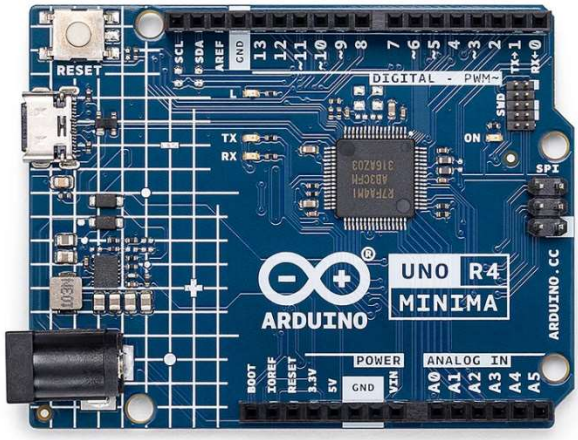

Figure 67 – Arduino Uno R4 Minima [44].

The UNO R4 Minima maintains the same form, pins and an operating voltage of 5 V as its predecessor, the UNO R3, but it has a faster clock speed and an increased memory: 8 times more flash memory (256 kB) and 16 times more RAM (32 kB). The board features 14 digital input/output pins, 6 analog channels, and 2 dedicated pins for I^2C. Among the 14 digital pins, certain pins can be used for the SPI and UART connections. Figure 68 illustrates the Arduino Uno R4 Minima pins. It is possible to power the Arduino Uno R4 Minima board through the VIN pin with any voltage between 6-24 V. The default reference voltage for the internal ADC is 5 V. The AREF (Analog Reference) pin can be used to provide an external reference voltage for the internal ADC.

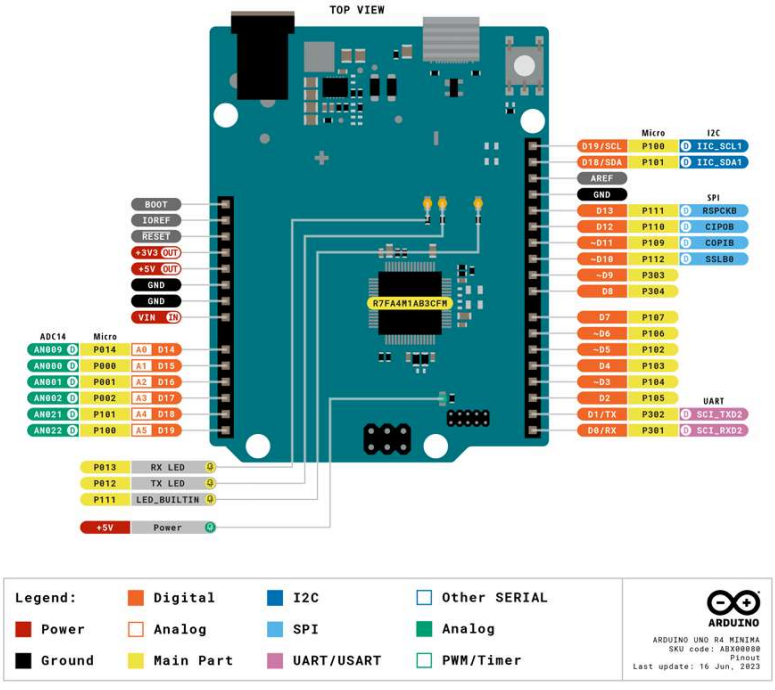

Figure 68 – Arduino Uno R4 Minima Pins [44].

Table 13 summarises the main specifics of the Arduino Uno R4 Minima.

Table 13 – Arduino Uno R4 Minima specifications [45].

Board	**Name**	Arduino UNO R4 Minima
	SKU	ABX00080
Microcontroller	Renesas RA4M1 (Arm Cortex-M4)	
USB	USB-C	Programming Port
Pins	Digital I/O Pins	14
Pins	Analog input pins	6
	DAC	1
	PWM pins	6
Communication	UART	Yes, 1x
	I²C	Yes, 1x
	SPI	Yes, 1x
	CAN	Yes 1 CAN Bus
Power	Circuit operating voltage	5 V
	Input voltage (VIN)	6-24 V
	DC Current per I/O Pin	8 mA
Clock speed	Main core	48 MHz
Memory	RA4M1	256 kB Flash, 32 kB RAM
Dimensions	Width	68.85 mm
	Length	53.34 mm

Arduino IDE

The Arduino Integrated Development Environment 2 (IDE 2) is an improvement of the classic IDE with increased performance, improved user interface and new features, such as autocompletion and a built-in debugger [46].

Figure 69 shows the main features of the Arduino IDE.

Figure 69 – Arduino IDE 2.0. Adapted from [46].

- The Sketchbook is where the code files are stored. The Arduino sketches are saved as .ino files and must be stored in a folder that carries the exact name of the .ino file.
- The Boards Manager contains packages for the boards. In the board manager it is possible to search for the desired board and to install it. A board is always required when compiling and uploading a code.
- The Library manager allows the user to search for the desired libraries, which can be installed in order to use them in the code.
- The Serial Monitor is a tool that allows to view the output data coming from the device, for example via the *Serial.print()* command.

The Arduino IDE is necessary to program both the Arduino and the LoRa32.

This chapter analysed in detail each sensor (the MA300 NTC, the MPU-6050 Accelerometer, the MAX30100 Heart Rate Detector and Pulse Oximeter), also describing the physical principles they are based on and also the devices aimed at the edge computing (the Raspberry, the LoRa32 and the Arduino). The next chapter is going to explore the Serial Communication Protocols, which allow the communications between the sensors and the device aimed at the edge computing.

5 SERIAL COMMUNICATION PROTOCOLS

This chapter firstly describes the protocols used for the respective serial communications between the sensors and the Raspberry or the Arduino (SPI and I²C). Next, the USB serial communication between the Raspberry and the LoRa32 and between the Arduino and the Raspberry is explored.

5.1 Inter-Integrated Circuit serial protocol (I²C)

The Inter-Integrated Circuit (I²C) is a serial bifilar system of communication used between integrated circuits [47]. The standard I²C communication is composed by at least one master and one slave, as shown in Figure 70. Its most common structure involves one single master and more slaves; multimaster and multislave structures can, however, be used in more complex systems. The I²C bus was designed by Philips in the early 80s to allow an easy communication between components which are present on the same circuit board.

The original communication was defined to be at a maximum speed of 100 kbps. Many applications don't require faster transmissions but, for the applications that need a faster communication, there is a 400 kbps fastmode option and, since 1998, a high speed 3.4 Mbps option.

The communication only needs two wires, the Serial Data line (SDA) is for the data and the Serial Clock line (SCL) is for the clock, which is generated by the master (Figure 70).

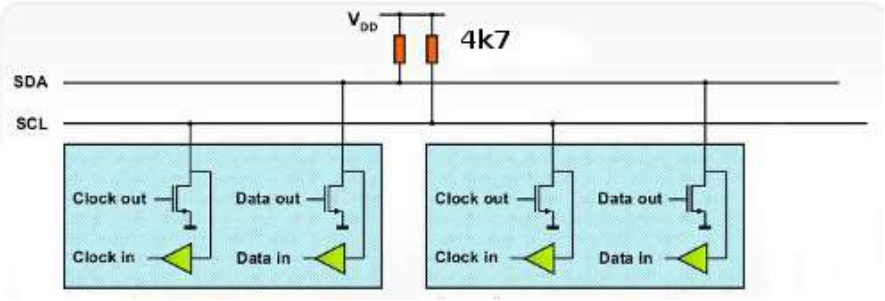

Figure 70 – I²C connection of one Master and one Slave devices [48].

SDA and SCL are open-drain, meaning that the I²C master and slave devices can only drive these lines low or leave them open.

Now, considering the SDA pin. Inside both the master and the slave there is a transistor, more exactly an N-channel MOSFET, whose symbol and structure are illustrated in Figure 71.

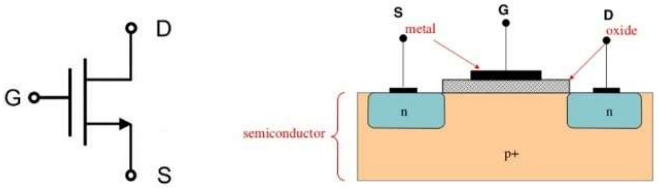

Figure 71 – N-Channel MOSFET symbol and structure. Adapted from [49].

The drain is connected to the SDA line, while the source is connected to ground. If no potential V_G is applied to the gate, this means that the voltage on the SDA wire is VDD (digital 1). There is no current in the pull-up resistor, so there is no voltage drop across the resistor and the line voltage remains unchanged.

On the contrary, if the correct positive potential is applied to the Gate ($V_{GS}>V_t$), the electrons of the p+ substrate will be attracted by the positive potential V_G, thus creating an N-channel. At this point a very low resistance path appears below the gate, the current passes through the channel and a significant voltage drop is observed across the pull-up resistor, the entire SDA wire becomes directly linked to a voltage near to ground (digital 0).

In general, there are two different ways of operating:

- The master transmits and the slave receives. The master controls the clock and sends data to the slave;
- The master receives and the slave transmits. The master controls the clock, but receives data from the slave.

Each node (each slave and each master) has a 7-bit address (B1, B2, B3, B4, B5, B6, B7). B1 is the most significative bit and B7 the less significative one [50]. An update of the protocol foresees, however, 10-bit addresses. With a 7-bit address there is a total of 128 addresses, 16 of which are, however, reserved, meaning that the maximum number of devices linked to the same bus can be of at maximum 112.

5.1.1 How the I²C communication works

Figure 72 illustrates how the I²C communication works [51].

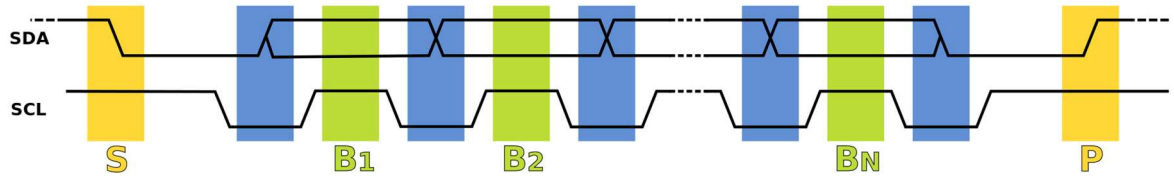

Figure 72 – I²C communication procedure [51].

1. The master begins the exchange of information by sending the start bit S: the SDA line is forced to be low by the master, while the clock SCL is at the high digital level.

2. The master sends the address of the slave which it wants to communicate with (B1, B2, …, B7).

The exchange of the bits happens in this way: when the clock line SCL is low (blue zone), the sending device puts on the SDA line the value that it wants to send (1 or 0). When SCL becomes high (green zone), the other device can read the value on SDA. This is a serial communication, so the bits are transmitted one by one.

3. The Master sends another bit (B8) with whom the master expresses to the slave its intention to read from that slave (SDA=1) or its intention to send information to that slave (SDA=0).

4. At this point the slave takes control of the line and, when SCL becomes high, the slave forces the SDA to be 0. By proceeding this way, the slave has sent an acknowledge.

5. Now there can be two situations:

   a) if the master is the one who wants to write, then it puts the desired bits on SDA and the slave sends to the master an acknowledge for each received byte;

   b) Instead, if the master wants to read, it is now the slave that has to put the bits of the data on SDA, while the master now is the one sending an acknowledge for each received byte.

6. Finally, to end the communication, the master sends a STOP bit (P). If, instead, it wants to keep control of the bus for another exchange of data, then it sends a START bit (S).

Table 14 summarises what was described above.

Table 14 – Recap of the I²C communication.

START	Slave Address	Read/Write	ACK	Data	ACK	Data	ACK	STOP
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

Table 15 presents the transmission of 2 bytes from the master to the slave.

Table 15 – Transmission of 2 bytes from the Master to the Slave.

START (0)	Slave Address (B1, B2, …, B7)	Write (0)	ACK (0)	Data	ACK (0)	Data	ACK (0)	STOP (1)
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

The grey cells contain the data sent by the master, while the white cells contain the data sent by the slave.

Table 16 shows the situation of the receiving of 2 bytes from the slave:

Table 16 – Transmission of 2 bytes from the Slave to the Master.

START (0)	Slave Address (B1, B2, …, B7)	Read (1)	ACK (0)	Data	ACK (0)	Data	ACK (0)	STOP (1)
1 bit	7 bits	1 bit	1 bit	8 bits	1 bit	8 bits	1 bit	1 bit

5.1.2 I²C Communication between the Arduino and the MAX30100

In order to read from a specific register of the MAX30100, the Arduino (which is the master) has to proceed as shown in Table 17 [31].

Table 17 – Example of the Arduino reading from a specific register of the MAX30100 [31].

MASTER ARDUINO	**SLAVE** MAX30100
START	
Slave Address (MAX30100 I²C Address)	
WRITE mode (0)	
	ACK
It writes the address of the register from which it wants to read	
START	
Slave Address (MAX30100 I²C Address)	
READ mode (1)	
	ACK
	It writes the required data
STOP	

On the contrary, Table 18 shows the situation when the Arduino wants to write in a register of the MAX30100.

Table 18 – Example of the Arduino writing in a specific register of the MAX30100 [31].

MASTER ARDUINO	**SLAVE** MAX30100
START	
Slave Address (MAX30100 I²C Address)	
WRITE mode (0)	
	ACK
It writes the address of the register in which it wants to write	
It writes the value it wants to write	
STOP	

In both situations there will be an ACK after each byte of transmission.

5.2 Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) protocol or, more precisely, SPI interface, was originally designed by Motorola (now Freescale) as a support for its own microprocessors and microcontrollers [52]. Differently from the I^2C standard, which was designed by Philips, the SPI interface has never been standardised; despite that, it became a standard. The SPI interface identifies a Master-Slave, synchronous and full-duplex communication. The clock of the communication is transmitted on a dedicated line and it is possible to both transmit and receive data at the same time. A basic scheme of the connection between two devices that make use of the SPI interface is shown in Figure 73.

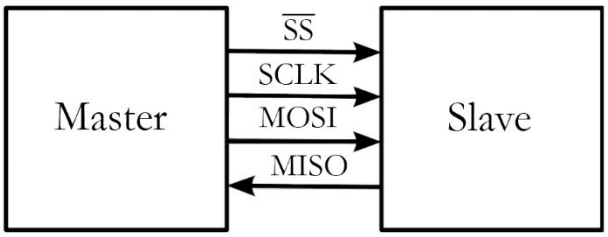

Figure 73 – SPI single Master and single Slave interface. Adapted from [52].

Since the interface has four connection lines, the SPI standard is also known as Four Wire Interface. The Master has the task of starting the communication and providing the clock of the communication to the Slave. This is the nomenclature of the different lines present in the SPI interface:

- MOSI: Master Output Slave Input;
- MISO: Master Input Slave Output;
- SCLK: Serial Clock (generated by the Master);
- SS: Slave Select (selection of the Slave).

In addition to this nomenclature, there are other acronyms:

- the MOSI line is also called Serial Data Out (SDO);
- the MISO line is also called Serial Data In (SDI);
- the SCLK line is also called CLK;
- the SS line is also called Chip Select (CS).

From these different nomenclatures it is possible to understand that there is not a real standard.

If there are more Slaves, it is necessary to have an SS line for each Slave, so that the Master can choose which Slave it wants to communicate with.

5.2.1 How the SPI communication works

As shown in Figure 74, both the Master and the Slave have their own Shift Register with the purpose of effectuating the exchange of data. It is important to observe that this is a serial communication, so the bits are exchanged one at a time.

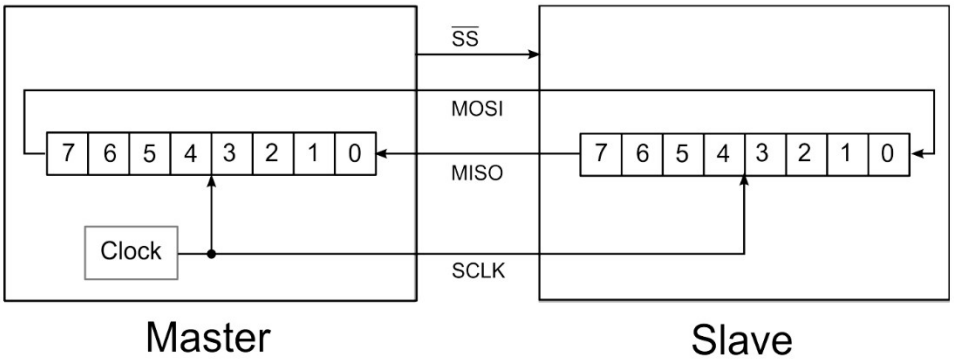

Figure 74 – Shift Registers in the SPI communication [52].

The SPI interface can be set to work in four different modes. The selected mode has to be the same for both the Master and the Slave [53]. The four different modes are set through the use of two parameters, called Clock Polarity (CPOL) and Clock Phase (CPHA).

The polarity consists in having or not a NOT port on the line of the Clock:

- CPOL=0 the clock is normally 0;
- CPOL=1 the clock is normally 1.

The CPHA parameter is used to choose on which clock edge the data has to be shifted:

- CPHA=0 the shift happens on the falling edge of the clock;
- CPHA=1 the shift happens on the rising edge of the clock.

Depending on the CPOL and CPHA bit selection, the four SPI modes are available, as shown in Table 19.

Table 19 – SPI Modes [53].

SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase used to shift the data
0	0	0	Logic low	Data shifted on the falling edge
1	0	1	Logic low	Data shifted on the rising edge
2	1	1	Logic high	Data shifted on the rising edge
3	1	0	Logic high	Data shifted on the falling edge

The functioning mode used by the Raspberry and by the ADC in this project is the mode CPOL=0 and CPHA=0. Figure 75 shows an example of the SPI Mode 0 [53].

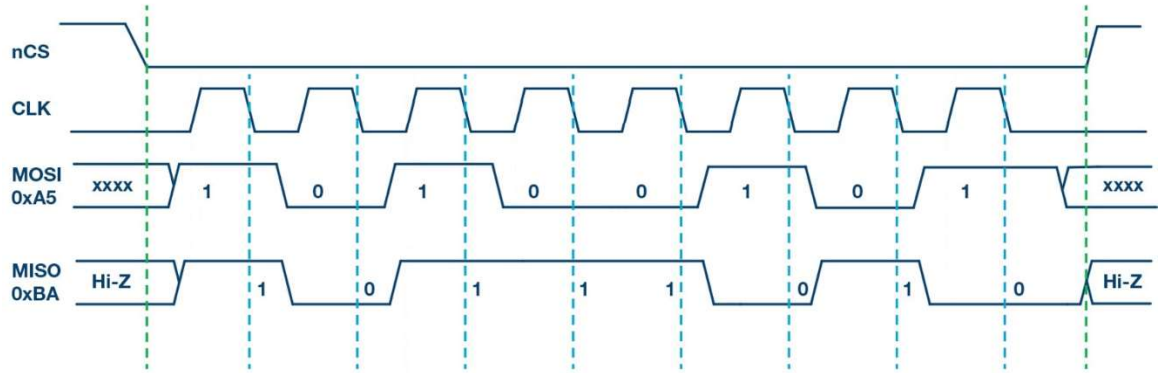

Figure 75 – Example of SPI communication with CPOL=0 and CPHA=0 [53].

- The Master activates the Slave Select.
- The Master activates the Clock.
- On the falling edge (blue) of the clock, all the bits of the two registers are shifted.
- On the new falling edge of the clock, a new bit is shifted.
- The procedure continues until the last bit.

Unlike the I^2C communication, in the SPI communication there is no Acknowledge.

For the ADC, the SS line, other than representing the Slave Select line, also has the task of starting the conversion.

5.2.2 SPI Communication between the Raspberry and the ADC

In Figure 76 there is an analysis of the communication between the Raspberry (master) and the ADC (slave).

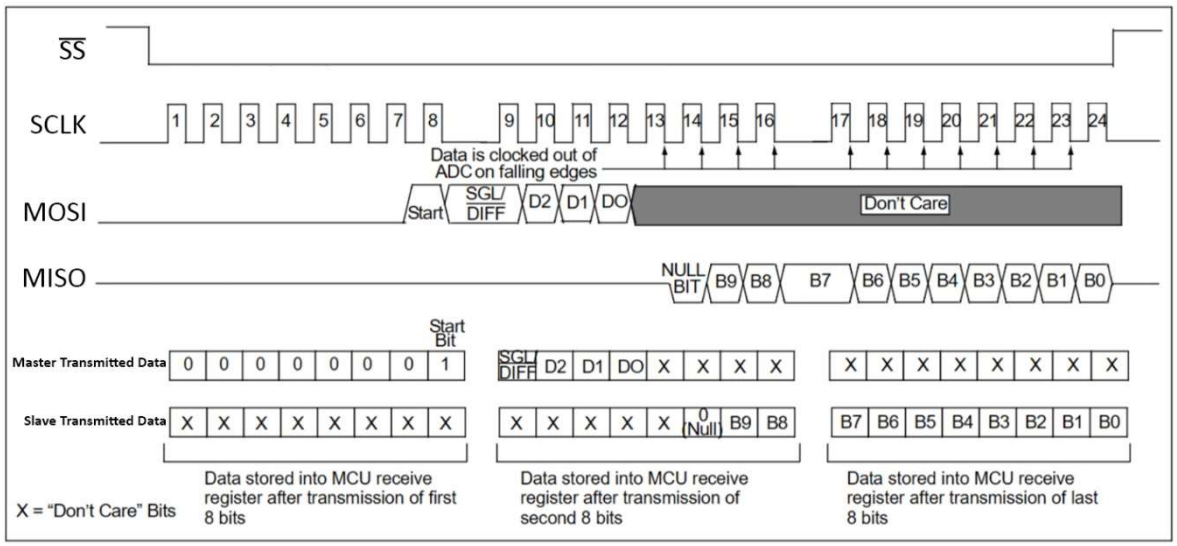

Figure 76 – SPI Communication with the MCP3008 (Mode 0,0) [17].

To start a communication, the Master pulls low the SS line.

The SPI communication foresees the shift of 8 bits between the two registers so, before the master sends the Start bit, it has to send seven '0'.

At this point the Master sends the Start bit: it sets the MOSI to 1 during one clock period.

The next bit sent by the Master is the SGL/DIFF, whose function is to choose whether the ADC has to be used in single mode (1) or in differential mode (0). Then the Master sends the bits D2, D1, D0 to choose which channel, if in single mode, or pair of channels, if in differential mode, of the ADC it wants to utilise. All the possible combinations are illustrated in Table 20.

Table 20 – Configure bits for the MCP3008 [17].

Control Bit Selections				Input Configuration	Channel Selection
Single /Diff	D2	D1	D0		
1	0	0	0	single-ended	CH0
1	0	0	1	single-ended	CH1
1	0	1	0	single-ended	CH2
1	0	1	1	single-ended	CH3
1	1	0	0	single-ended	CH4
1	1	0	1	single-ended	CH5
1	1	1	0	single-ended	CH6
1	1	1	1	single-ended	CH7
0	0	0	0	differential	CH0 = IN+ CH1 = IN-
0	0	0	1	differential	CH0 = IN- CH1 = IN+
0	0	1	0	differential	CH2 = IN+ CH3 = IN-
0	0	1	1	differential	CH2 = IN- CH3 = IN+
0	1	0	0	differential	CH4 = IN+ CH5 = IN-
0	1	0	1	differential	CH4 = IN- CH5 = IN+
0	1	1	0	differential	CH6 = IN+ CH7 = IN-
0	1	1	1	differential	CH6 = IN- CH7 = IN+

At this point, starting from the 14th falling edge, the ADC starts to send (starting from the MSB) the 10 bits of the acquired sample amplitude value.

5.3 Universal Asynchronous Receiver-Transmitter (UART) over Universal Serial Bus (USB)

On the Raspberry, the Arduino and the LoRa32, there are the serial libraries, which allow the Universal Asynchronous Receiver-Transmitter (UART) communication over USB. In this section there is going to be an analysis on the USB standard on the aspects that concern the cable and the USB connectors. Subsequently, the aspects of the UART communication protocol are analysed.

5.3.1 USB Cable and Connectors

Over time the USB standard has progressed in terms of the shape of the connector. Figure 77 represents the main existing USB connectors [54].

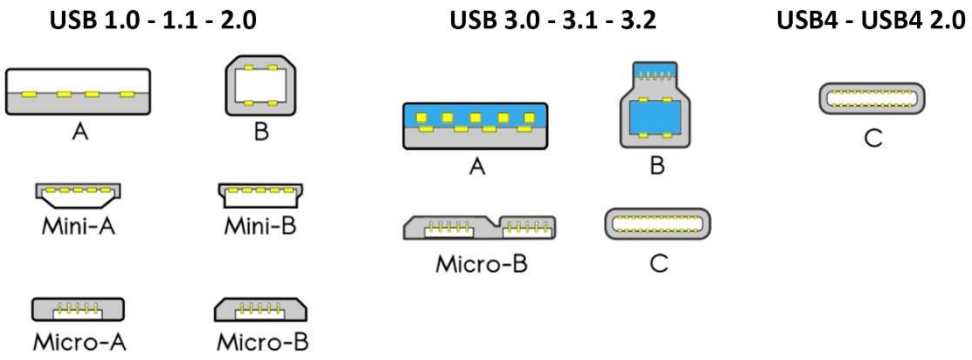

Figure 77 – USB connectors. Adapted from [54].

The type C connector is the most innovative one and introduces new great innovations, such as the possibility of recharging devices that require more than 25 W through two new pins and the reversibility in inserting it in the device. The main advantage of the USB of type C connector stands, in fact, in its being symmetrical, because it facilitates the physical connection. The Raspberry supports this type of connector, in fact its power supply is connected through it. The LoRa32, however, only has one USB 2.0 type Micro-B port, so this version is going to be analysed in detail.

Figure 78 shows an image of a USB 2.0 cable, of its standard colours and of its isolation.

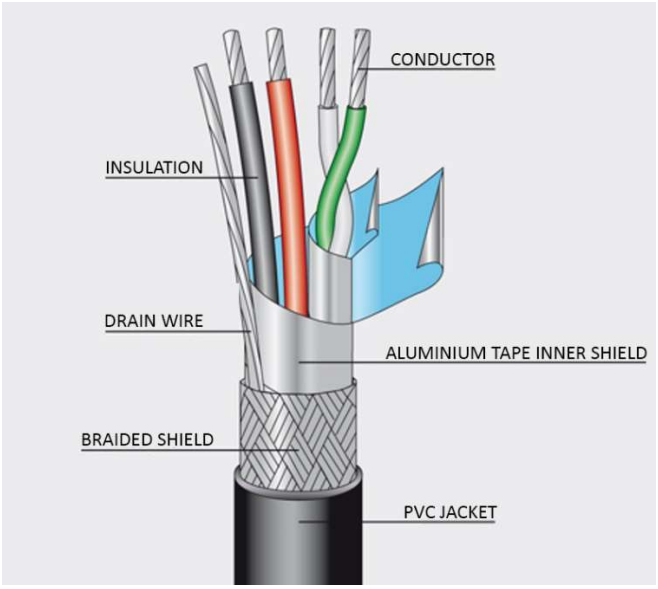

Figure 78 – USB 2.0 cable. Adapted from [55].

The USB 2.0 cable utilises four isolated conductors, two of whom are destined to the power supply (+5 V e GND), while the other two conductors are used for the

transmission of data. The last two conductors form a pair of twisted conductors with the aim of limiting the unwanted external electromagnetic interference. In this way, in fact, the voltage that is induced in each of the two wires is the same, so it can be cancelled at the receiver by detecting the difference signal between the two conductors.

Table 21 indicates the four wires of the USB 2.0 cable.

Table 21 – Wires in the USB 2.0 cable.

Pin	Wire Name	Wire Colour	Wire Description
1	VBUS	RED	+5 V
2	D-	WHITE	Data -
3	D+	GREEN	Data +
4	Ground	BLACK	GND

The USB 2.0 utilises a differential pair for the data transmission (D- e D+). The logic level '1' is transmitted when D+ is brought to the voltage 2.8 V and D- is brought to the voltage 0.3 V. Vice versa, the contrary happens in the case of the transmission of the logic level '0'. VBUS and Ground have the task to power the device.

In this project, D+ and D- are used as connection wires between the TX and the RX pins.

5.3.2 Universal Asynchronous Receiver-Transmitter (UART)

Universal Asynchronous Receiver-Transmitter (UART) is a serial communication protocol for sending serial data over USB or via TX/RX pins [56]. This protocol is used in the communication between the Arduino and the Raspberry and in the communication between the Raspberry and the LoRa32. Both of them use the USB cable, but the exploited protocol is UART over USB.

UART operates by transmitting data as a series of bits serially. The protocol operates asynchronous, which means that a shared clock signal does not exist. Instead, it uses predefined baud rates to determine the speed at which the data is transmitted over the communication channel. The baud represents the number of bits transmitted in one second (bps). For the communication to work, both the transmitting and receiving devices have to use the same baud rate. Figure 79 shows the UART frame format.

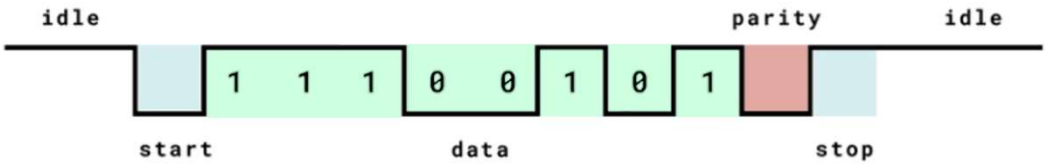

Figure 79 – UART frame format. Adapted from [56].

The TX pin of one device is connected with a wire to the RX pin of the other device and vice versa. The TX idle state is the logic 1. At the beginning of each UART frame, one start bit (blue) is transmitted to indicate the start of the data transmission and prepare the receiver for the data reception. The start bit is a logic 0. Then, the actual data (green) is sent one bit at a time. The number of bits of the actual data in a UART frame can vary, but a common configuration is 8 bits. This implies that there are 2^8=256 different combinations, so with a single frame it is possible to send one out of the 256 ASCII characters. The feature "parity" of the frames can be set to "odd" or "even" and is an error-checking mechanism. For example, in Figure 79 the parity feature is set to "odd", meaning that the number of bits of the frame (the bits of the actual data + the parity bit) whose value is the logic 1, must be odd. Since the actual data already contains an odd number of 1s, then the parity bit (red) is 0. On the contrary, if the actual data contained an even number of 1s, then the parity bit would have been 1, in order to obtain an odd number of 1s. The presence of a parity bit allows the receiver to verify the integrity of the received data. If the number of 1s does not match the expected parity, an error is detected. The frame ends with the stop bit (blue), followed by the idle state (the logic 1).

Identification of the devices connected via USB to the Raspberry

The Arduino and the LoRa32 are both connected to the Raspberry via USB. This means that, in order to use each of them in the Raspberry software, it is firstly necessary to uniquely identify them. In the command prompt of the Raspberry, the line *ls /dev/tty** was entered with the aim of listing all the connected devices, while neither the Arduino nor the LoRa32 were connected to the Raspberry. Figure 80 illustrates what was obtained.

Figure 80 – *ls /dev/tty** while neither the Arduino nor the LoRa32 were connected to the Raspberry.

Subsequently, the Arduino was connected via USB to the Raspberry and the line *ls /dev/tty** was entered again, now producing the resulting Figure 81.

Figure 81 – *ls /dev/tty** while only the Arduino was connected to the Raspberry.

How shown in Figure 81, the device ttyACM0 is now in the list, meaning that this identifies the Arduino.

Thereafter, also the LoRa32 was connected to the Raspberry via USB and the line *ls /dev/tty** was entered again (Figure 82).

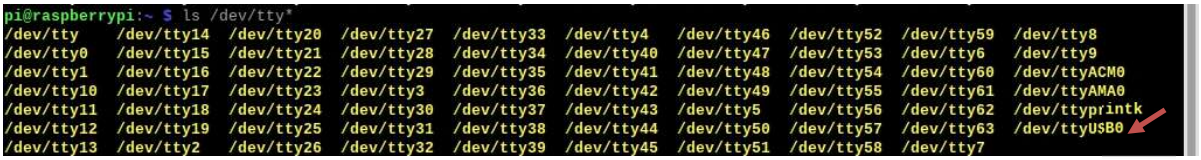

Figure 82 – *ls /dev/tty** while both the Arduino and the LoRa32 were connected to the Raspberry.

The device ttyUSB0 appeared now on the list, identifying the LoRa32.

This chapter analysed the different protocols used for the serial communications between the sensors and the Raspberry or the Arduino (I^2C and SPI). Subsequently, it explored the UART over USB communication between the Raspberry and the LoRa32 and between the Arduino and the Raspberry. The next chapter is going to deal with the protocols that allow the entire device to communicate with the external servers and is also going to describe the external servers used.

6 CONNECTIVITY AND APPLICATION LAYERS

As already mentioned in previous chapters, there are two different paths for the transmission of the data: one is used when in the Wi-Fi range and the other when the Wi-Fi is out of range. In this chapter the ISO OSI model is analysed, since all the communication protocols used in this project are based on it. Subsequently, there is an analysis of the protocols used in the first path (to send data from the end-device to the Akenza network server), namely: Wi-Fi, Transmission Control Protocol (TCP), Internet Protocol (IP) and MQTT. Finally, there is an analysis of the protocols used in the second path (to send data from the end-device to the TTN network server), namely: Long Range (LoRa) and Long Range Wide Area Network (LoRaWAN).

Later in this chapter, there is an analysis of the servers and integration used in this project: The Things Network and Akenza.

6.1 ISO OSI Model and TCP/IP Model

The OSI model was developed by the International Standards Organization (ISO). It is a conceptual model that provides a common basis that has to be followed in the development of the standards of communication in a systems interconnection. It is defined OSI, that stands for "Open Systems Interconnection", because it is referred to the connection of open systems, as they are "open" to the communication with other systems. The OSI model comprises seven layers, namely: Physical, Data Link, Network, Transport, Session, Presentation and Application. Figure 83 illustrates the ISO OSI model on the left and the TCP/IP model on the right [57].

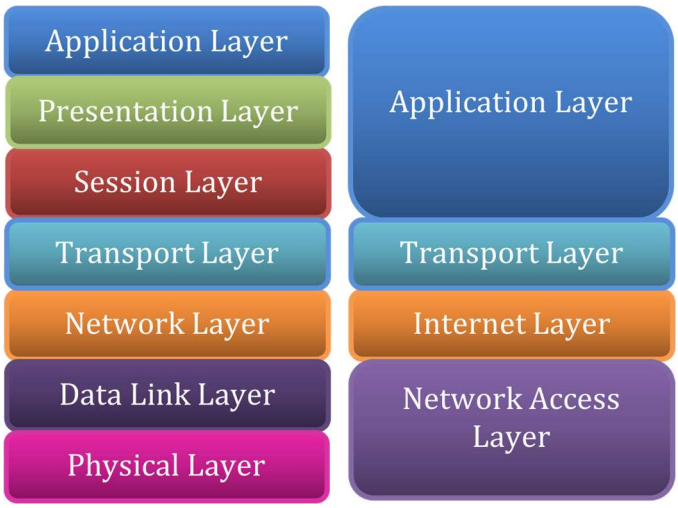

Figure 83 – OSI and TCP/IP Models. Adapted from [57].

On the right, the representation of the TCP/IP model consists of four layers: Network Access, Internet, Transport and Application.

In the transmission of a message, the data from a certain upper layer is divided and inserted (encapsulated) into the data of the respective lower layer (and the new data takes a different denomination). This operation called "encapsulation" is the process of adding headers and trailers around the data of the respective upper layer.

In the reception of the message the reverse process takes places and, as the message goes from a certain lower layer to the respective upper layer, each layer analyses its own header and gives the remaining information to the respective upper layer. This means that the information regarding the addresses of a particular layer is removed when going to the respective upper layer and, at the application layer, the message only consists of the useful data.

The application layer consists of the applications and the programs with whom the user interacts during the data transmission between the sender and the receiver.

The data sent through an application will first move from the application layer up to the transport layer, where there could be one of two possible transmission protocols: TCP or User Datagram Protocol (UDP). In the transport layer the data is divided and encapsulated into segments (if the TCP protocol is used) or datagrams (if the UDP protocol is used).

TCP guarantees check for errors and guarantees that all the segments will be delivered from the source to the destination, because it provides the retransmission of lost data segments. Therefore, TCP is slower than UDP. This means that TCP is the perfect protocol for transferring information like images, data files, and web pages, because in these situations the integrity of the image is more important than having the image as soon as possible. On the contrary, UDP is preferred for streaming videos, because in this case the speed of the transmission is more important. In this project the protocol under MQTT is TCP, as the transmission of the correct data is more important. TCP also provides control flow, in order to guarantee that a source which is faster than the receiver cannot congest the receiver with an amount of data that is greater than the data that the receiver can manage [58].

In the TCP/IP model the transport layer will pass its information to the Internet Layer (Network layer in the ISO OSI model), which divides and encapsulates the information into packets. The created packet contains the IP addresses of the source and of the destination. This layer is responsible for the routing, which comprises the selection of the optimal path between the source and the destination, in order to ensure the transmission of the packets. This choice is made by the routers through the routing algorithm and the routing tables [57].

In the ISO OSI model the Data Link layer is composed of two sublayers: the Logical Link Control (LLC) and the Media Access Control (MAC). LLC divides the received packets and encapsulates them with its layer information into frames of the LLC layer. The task of LLC is to hide from the network layer the differences that exist between the different IEEE 802 family protocols, so that the network layer can operate in the same way for all the transmission means of communication. Finally, the data goes from the LLC sublayer to the MAC sublayer, which has access to the medium. The Media Access Control encapsulates the LLC frames into frames appropriate for the specific transmission medium and, in the header of its frame, there are the MAC address of the source and of the destination, which are in the same network. Lastly the data is passed to the physical layer.

In the ISO OSI model the Physical layer takes care of the transmission of bits between network nodes on the communication channel. At the physical layer different protocols can be used, as the communication channel can be a specific physical cable or even a wireless connection.

The Physical layer is responsible for the transmission of the raw data, which is a series of 0s and 1s and for the bit rate control. Of course, it is required that the received bits have to be of the same value of the correspondent transmitted bits.

In the TCP/IP model the two layers Data Link and Physical are one single layer called Network Access Layer.

6.2 Wi-Fi

In 1997 the IEEE released the IEEE 802.11 standard for wireless LAN (WLAN), which has a data transfer rate of from 1 Mbps up to 2 Mbps and it maintains compatibility with existing LAN hardware and software infrastructure [59]. This standard defines protocols for MAC layer and physical transmission in the unlicensed 2.4 GHz radio band. After the successful implementation by commercial companies, improvements were made for a better performance in the same year and, as a result, the IEEE 802.11b protocol was standardized in 1999. The IEEE 802.11b has higher data transfer rates of up to 11 Mbps. Moreover, it differs from the 802.11 in the MAC layer even though it retains compatibility with its predecessor and the physical layer is unchanged. The IEEE 802.11b standard was very successful in the commercial domain.

In 1999, several visionary companies formed a global non-profit association with the goal of improving this new wireless networking technology [60]. In 2000, the group adopted the term "Wi-Fi" as the proper name for its technical work and announced that its official name would have been Wi-Fi Alliance.

As it was declared by Phil Belanger, cofounder of the Wi-Fi Alliance, the term "Wi-Fi" does not have a specific meaning, but it simply represents the commercial name used to indicate the group of protocols IEEE 802.11 [61].

After the b version, other standards were developed in the 802.11 family, as summarised in Table 22.

Table 22 – IEEE 802.11 Standards.

Standard IEEE	Rate (Mbps) min/max	Frequencies (GHz)	Release year
IEEE 802.11ax (Wi-Fi 6)	600 – 9608	2.4/5	2019
IEEE 802.11ac (Wi-Fi 5)	433 – 6933	5	2014
IEEE 802.11n (Wi-Fi 4)	72 – 600	2.4/5	2009
IEEE 802.11g	3 – 54	2.4	2003
IEEE 802.11a	1.5 – 54	5	1999
IEEE 802.11b	1 – 11	2.4	1999

The currently used version is the Wi-Fi 6 (IEEE 802.11ax). The new Wi-Fi 7 technology should be released by the year 2024 and it should provide a 30 Gbps data transfer rate.

6.2.1 Wi-Fi Architecture

The Wi-Fi network is a telecommunications network, eventually interconnected to the Internet network. It works using the Access Point (AP). The AP is a network device that, when connected to a LAN, permits a device to access the LAN in a wireless way (without use of cables). Normally, the AP function is already present in the router. The Service Set Identifier (SSID) is the name with whom the Wi-Fi network identifies itself to the users. The APs are usually configurated in order to announce their SSID continuously, so that the Wi-Fi devices can create a list of all the wireless networks that are available in the place they are at the moment. This list can be shown to the user, so that it is possible to choose the desired network.

6.3 MQTT

MQTT is a standard messaging protocol used on Internet of Things (IoT) applications [62]. The first version of the protocol was designed by Andy Stanford-Clark and Arlen Nipper in 1999. The protocol was used to monitor oil pipelines.

MQTT was designed for connections between devices that have resource constraints, such as little battery power or that have limited network bandwidth, such as in the Internet of Things. MQTT is a protocol in the application layer and typically runs over TCP, since this transport protocol provides ordered, lossless and bidirectional connections. This is represented in Figure 84.

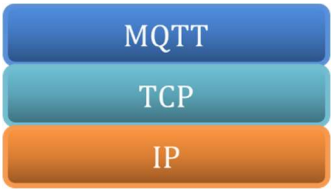

Figure 84 – MQTT on TCP/IP.

In software architecture, publish–subscribe is a messaging pattern where the senders of the messages are called publishers, while the receivers are called subscribers. The publishers do not send the messages directly to the specific subscribers, instead they categorise the published messages into classes, called topics without knowledge of any possible subscribers. Publish is a mechanism used to send messages. Similarly, the subscribers express interest in (subscribe to) one or more topics and only receive a copy of all the messages that are in those specific topics, without knowledge of any possible publishers. Subscribe is a mechanism used to receive messages.

The topics are named logical channels represented with strings separated by a forward slash. Each forward slash indicates a topic level and the name of the topic is case-sensitive.

For example, the subscriber Akenza is subscribed to the topic:

/up/hcvu8mlrcg6o2otd5g10mnp71bs4dvkk/id/3AEB0E59DE79BAEC

This means that, in order to send a message to the subscriber Akenza, the publisher has to categorise that message in this specific model of topics. In this topic-based system the publisher is responsible for defining the topics to which the subscribers can subscribe. The MQTT communication is represented in Figure 85.

Figure 85 – MQTT Topic [62].

In a publish-subscribe system, between the publishers and the subscribers there is an intermediary (a software running on a computer) called message broker, as represented in Figure 86.

Figure 86 – MQTT Broker. Adapted from [62].

Publishers post messages to a message broker and subscribers register subscriptions with that broker. The message broker receives all the messages and performs a store and forward function on the messages. The broker, in fact, knows who is interested in that topic, so it sends the messages to all the subscribed clients to that topic.

A MQTT client is any device that runs a MQTT library and connects to a MQTT broker. Both the publishers and the subscribers are MQTT clients.

In this specific case, the Raspberry is the MQTT publisher client and Akenza is the MQTT subscriber client and the broker is the Akenza MQTT Broker.

The payload is the actual message or data sent from the client publisher to the client subscriber.

A single broker can have multiple subscribers clients to a topic (one to many capability) and a single client can subscribe to topics with multiple brokers (many to one capability).

MQTT is a bidirectional communication protocol, as represented in Figure 87.

Figure 87 – MQTT bidirectional communication. Adapted from [62].

This means that each client can both send or receive data by respectively publishing or subscribing.

6.4 LoRa

LoRa is a wireless low-power wide-area network modulation technique based on the Chirp Spread Spectrum (CSS) technology. It was developed by Cycleo of Grenoble, in France, and bought in 2012 by Semtech, founding member of the LoRa Alliance [63].

LoRa modulated transmission is very robust against noise and other channel impairments allowing communications at great distances.

The LoRa physical layer is private, hence free official documentation about it does not exist, even though Semtech has made available an overview of the modulation and of other specific features.

By using LoRa, data can be transmitted at a longer range compared to technologies like Wi-Fi, Bluetooth or ZigBee, making LoRa a great choice for sensors and actuators [64]. LoRa operates in sub-gigahertz unlicensed bands, namely [65]:

- EU433 (433.05-434.79MHz) and EU863-870 (863-870/873 MHz) in Europe;
- AU915-928/AS923-1 (915-928 MHz) in Australia;
- US902-928 (902-928 MHz) in North America;
- IN865-867 (865-867 MHz) in India;
- AU915-928/AS923-1 and EU433 Southeast Asia;

6.4.1 Chirp Spread Spectrum.

LoRa modulated signals use the Chirp Spread Spectrum modulation scheme.

The term Spread Spectrum indicates that the transmitted signal is spread over a wide frequency band, much wider than the bandwidth required to transmit the message. This technique provides robustness to noise and interference [66].

The term Chirp is used to indicate signals that have a constant amplitude and whose frequency varies in the whole bandwidth from one end to another end in a certain time [67]. If the frequency changes from the lowest to the highest, it is defined up-chirp; on the contrary, if the frequency changes from the highest to the lowest, it is a down-chirp.

The chirp spread spectrum is a spread spectrum system, because to send a symbol it does not use one single frequency, but the entire bandwidth. Figure 88 shows a linear up-chirp modulated signal in the time domain.

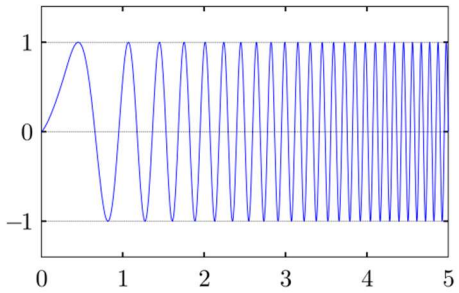

Figure 88 – Linear up-chirp [67].

The parameters of the Chirp Spread Spectrum are [68]:

The bandwidth $B = f_{max} - f_{min}$, where f_{max} and f_{min} are the maximum and the minimum frequency respectively;

The Spreading Factor SF, that corresponds to the number of bits per symbol;

The number of symbols $n_{symbols} = 2^{SF}$;

The time of duration of each symbol $t_s = \frac{2^{SF}}{B}$;

The Symbol Rate (symbol per second) $R_s = \frac{1}{t_s} = \frac{B}{2^{SF}}$;

The Bitrate (bits per second), which is obtained by dividing SF by the time of duration of each symbol: $R_b = \frac{SF}{t_s} = \frac{SF}{\frac{2^{SF}}{B}} = SF \cdot \frac{B}{2^{SF}}$.

Table 23 summarises the parameters of LoRa.

Table 23 – Parameters of LoRa.

Name	Formula	Unit of measurement
Bandwidth	$B = f_{max} - f_{min}$	Hz
Spreading Factor	SF	bit/symbol
Number of symbols	$n_{symbols} = 2^{SF}$	symbols
Time of duration of each symbol	$t_s = \frac{2^{SF}}{B}$	s/symbol
Symbol Rate	$R_s = \frac{1}{t_s} = \frac{B}{2^{SF}}$	symbols/s
Bit Rate	$R_b = \frac{SF}{t_s} = SF \cdot \frac{B}{2^{SF}}$	bit/s

For example, when representing two bits at once, $SF = 2$ and there are $n_{symbols} = 2^{SF} = 2^2 = 4$ symbols. What differentiates a symbol from another one is the beginning frequency. As it is possible to notice from Figure 89, the signal for the symbol 00 begins from the minimum frequency $f_0 = f_{min}$ and in the time t_s it reaches the maximum frequency f_{max}. The symbol 01, instead, begins with the frequency f_1 and, after reaching the maximum frequency f_{max}, it begins again from the minimum frequency f_{min} until it reaches the frequency f_1. All of this always happens in the time t_s. Instead of simply representing the symbol 00 with only the frequency f_0, this system uses all the bandwidth B, as it is a spread spectrum system.

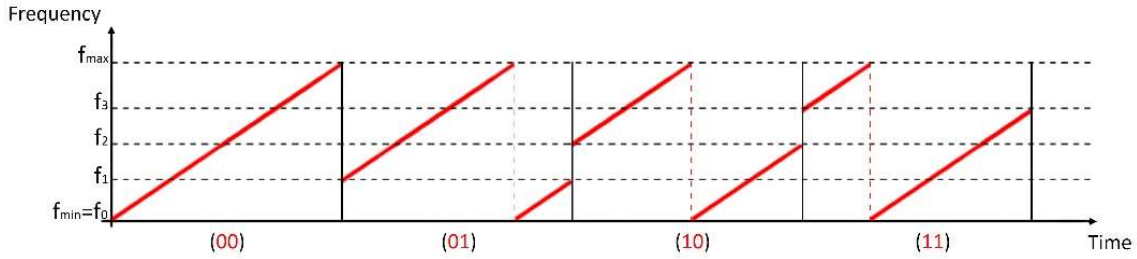

Figure 89 – Example of LoRa signal with SF=2.

LoRa can use one of three different bandwidths: 125 kHz, 250 kHz and 500 kHz and the Spreading Factor SF values range from 7 up to 12 [67]. Figure 90 represents the symbol 0 using different values of SF (always using a bandwidth B=125 kHz).

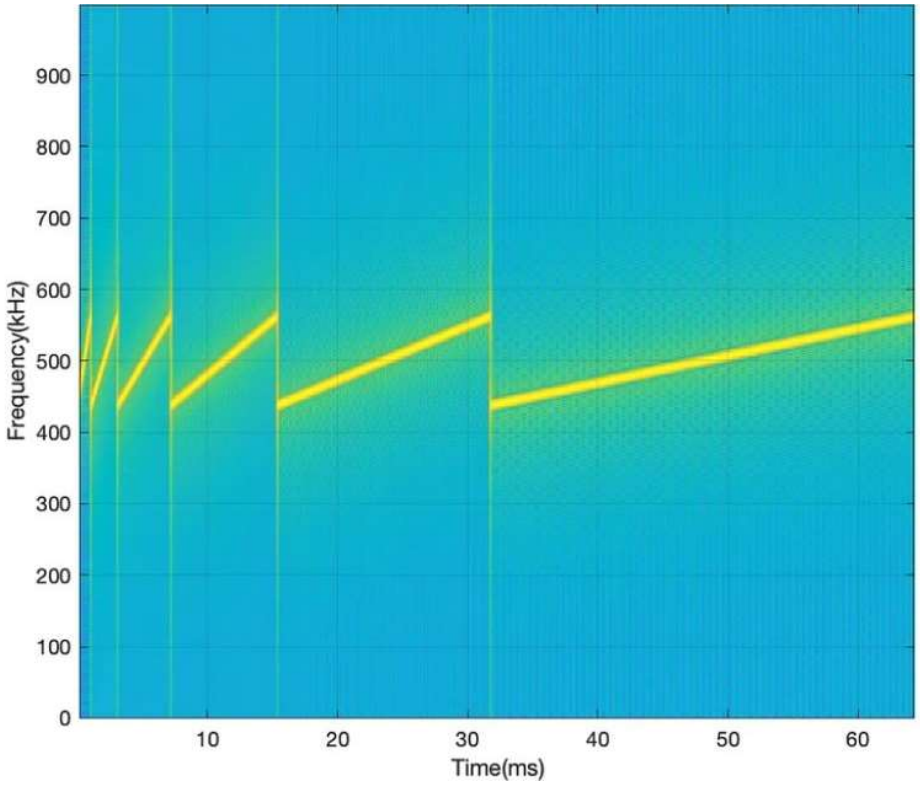

Figure 90 – Comparison of LoRa Spreading Factors: SF=7 to SF=12 in the symbol 0 [67].

As it is possible to notice, the time duration of each symbol t_s increases with the increasing of SF.

6.4.2 LoRa Message Format

LoRa Physical Layer message, which is shown in Figure 91, is composed of Preamble, PHDR (Physical Header), PHDR_CRC (Header Cyclic Redundancy Check CRC), PHYPayload and CRC (only in uplink) [69].

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC

Figure 91 – LoRa Physical layer Message [69].

The Preamble is used to detect the LoRa signals. LoRa supports a variable length preamble between 6 and 65535 symbols followed by the synchronization symbols and two and a quarter down-chirp symbols (2.25 symbols).

- PHDR (Physical Header) contains information about payload size and CRC.
- PHDR_CRC (Header CRC) contains an error detecting code for correcting errors in PHDR.

- The Payload contains the message, which corresponds to the entire data of the upper MAC layer.
- The CRC is used to check errors in the physical payload. In LoRaWAN, the CRC is present only in uplink, but in general it can be enabled or disabled in the PHDR.

Figure 92 shows an example of the transmission of a LoRa physical layer message.

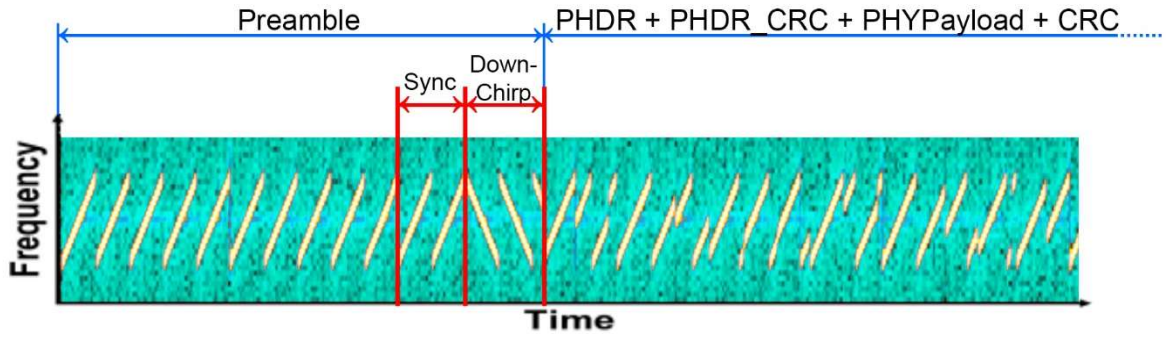

Figure 92 – LoRa Physical layer Message (Frequency versus Time). Adapted from [70].

In Figure 92 the Preamble is composed of: ten symbols 0 (which begin from $f_0=f_{min}$), followed by two symbols 0 (Sync) and 2.25 down-chirp symbols 0. The other symbols (which begin from a certain frequency and, after reaching f_{max}, they begin again from f_0 to complete the remaining bandwidth until that frequency) are for PHDR, PHDR_CRC, PHYPayload and CRC.

6.5 LoRaWAN

As seen before, LoRa is the physical layer and allows a long-range communication. LoRaWAN is a Media Access Control (MAC) layer protocol built on top of LoRa Physical layer [71]. LoRaWAN is a software layer which defines how devices use the LoRa hardware, for example when they transmit, and the format of the frames. LoRaWAN was created by the LoRa Alliance and its first version was published in 2015, followed by its second version in 2017.

LoRaWAN is technically not suitable for high bandwidth like transmission of real-time videos and images. Instead, it is only suitable for sending small amounts of data, few bytes, at low data rates ranging from 300 bps to around 50 kbps [71].

LoRaWAN networks are asynchronous, meaning that the nodes only communicate when they have data to send. This affects the battery lifetime of the devices, in fact LoRa technology has a long battery lifespan.

LoRaWAN networks are typically laid out in a star-of-stars topology in which gateways relay messages between end-devices and a central Network Server [72]. As can be seen in Figure 93, there is a main star topology in which the nodes are the LoRaWAN gateways, and these are individually connected to a central connection point, which is the Network Server. All the gateways are, then, individually a central connection point for all the connected end-devices.

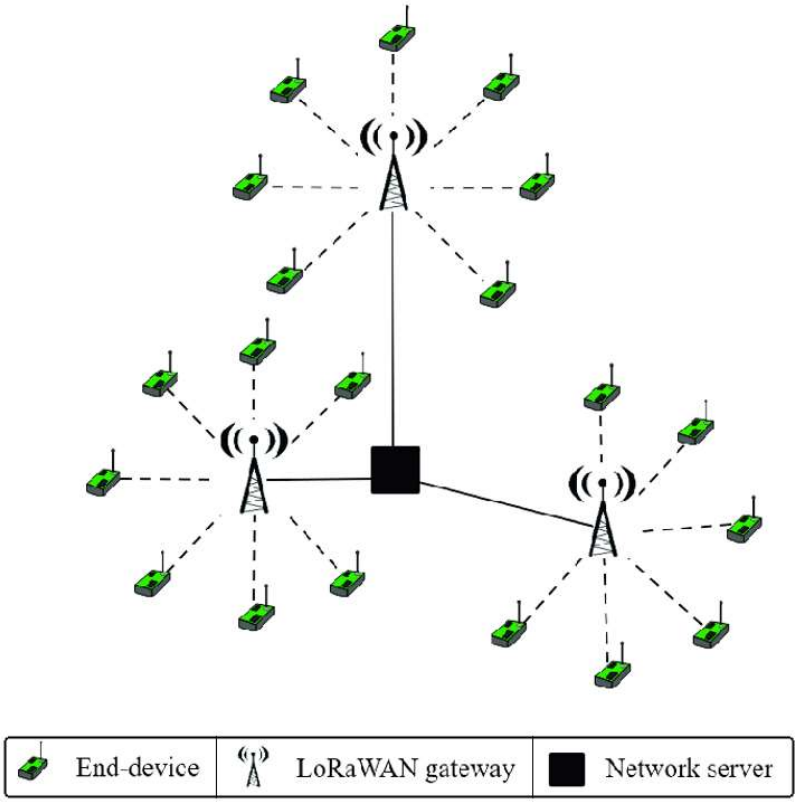

Figure 93 – LoRaWAN Star-of-Stars Topology. Adapted from [72].

Figure 94 shows an architecture of LoRaWAN [73].

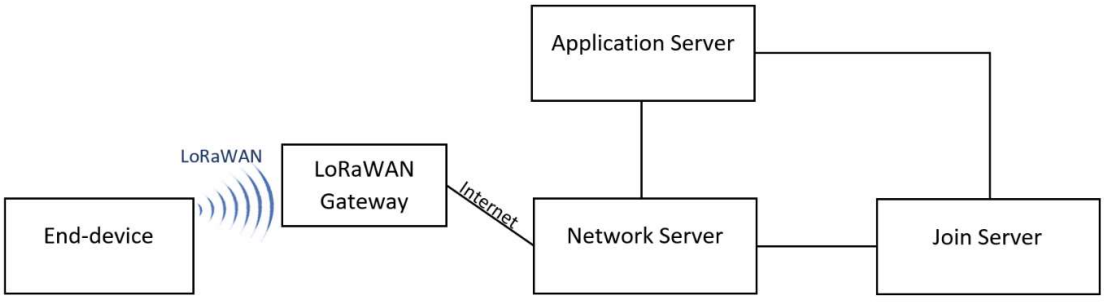

Figure 94 – LoRaWAN Architecture. Adapted from [73].

Network Server, Application Server and Join Server are co-located. The end-device uses the LoRaWAN communication to one or many gateways and the gateways are connected to the Network Server via Internet connection. The communication is generally bidirectional, although uplink communication from the end device to the Network Server is expected to be the predominant traffic [73].

End-Device:

The prototype unit developed in this project is made up of several microcontrollers (the Raspberry, the Arduino and the LoRa32) and sensors (the MA300, the MPU-6050 and the MAX30100), that was named End device. The End-Device is wirelessly connected to the LoRaWAN Gateway and the Gateway is, in turn, connected to the LoRaWAN Backend. The application layer of the End-Device is connected to a specific Application Server. All application layer payloads of this End-Device are routed to its corresponding Application Server.

LoRaWAN Gateway:

A LoRaWAN gateway is an intermediary that allows LoRaWAN end devices to transmit data to the network server. Gateways are often mains-powered and some can have backup batteries or solar panels and they can be indoor or outdoor. The gateways use high bandwidth (Wi-Fi, Ethernet, cellular, satellite) connections to the Internet in order to reach the Network Server. A single gateway is capable of serving thousands of devices. Whenever an end device transmits a message, all gateways in the range will receive that message and forward it to The Things Stack Community Edition. The network server is responsible for deduplicating those messages and selecting the best gateway for downlink communication. Gateways are routers equipped with a LoRa concentrator, which allows them to receive LoRa frames. LoRaWAN operates on unlicensed bands so, in most countries, running a personal gateway is completely legal. There can be, however, country-to-country restrictions for example on where to install it. However, it was not necessary to install a personal gateway, because there are several gateways covering the Coimbra city area. Figure 95 shows the gateways present in Coimbra.

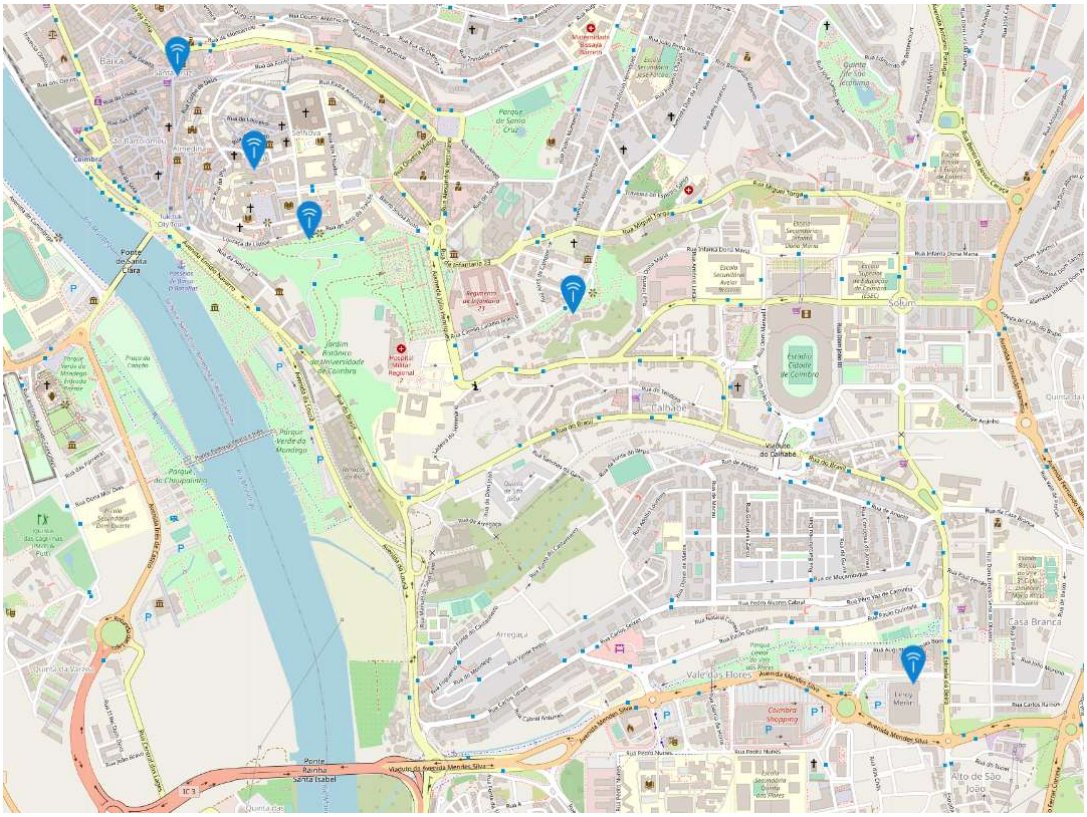

Figure 95 – LoRaWAN Gateways in Coimbra.

Network Server:

The Network Server terminates the LoRaWAN MAC layer for the End-Device connected to the network. It is the centre of the main star topology.

Generic features of Network Server are:

- It filters redundant data coming from more LoRaWAN gateways and chooses the optimal gateway for the downlink communication;
- Frame authentication and frame counter checks;
- It automatically schedules acknowledges through the optimal gateway;
- Adaptively optimises data rates based on device or network conditions;
- Responding to all MAC layer requests coming from the End-Device;
- Forwarding uplink application payloads from the End-Device of the network to the associated Application Server;
- Sending downlink payloads from the Application Server to the End-Device;

- Forwarding Join-request and Join-accept messages between the End-Device and the Join Server.

Join Server:

To secure radio transmissions the LoRaWAN protocol relies on symmetric cryptography using session keys derived from the device's root keys. In the backend the storage of the device's root keys and the associated key derivation operations are ensured by the Join Server. It also communicates the Network Session Key of the End-Device to the Network Server, and the Application Session Key to the corresponding Application Server.

The Join Server manages the Over-The-Air (OTA) End-Device activation process.

Application Server:

The Application Server handles all the application layer payloads of the associated End-Device and provides the application-level service to the end-user. It also generates all the application layer downlink payloads towards the connected end-device.

6.5.1 LoRaWAN Message Format

LoRa physical message has already been analysed in the Subsection 6.4.2 and it is represented in Figure 91 [74].

The physical payload of the LoRa message is the entire data of the upper layer (MAC). The LoRaWAN frame can be one of the types shown in Figure 96.

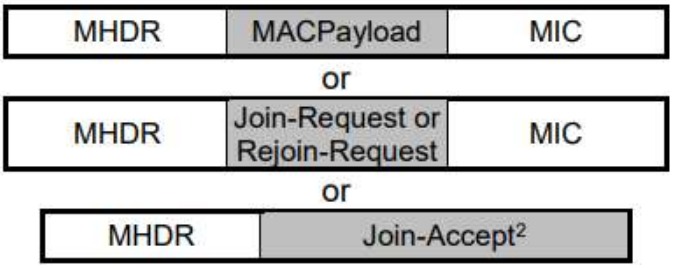

Figure 96 – LoRaWAN Frame [74].

All the MAC messages start with a single-octet MAC header (MHDR), followed by a MAC payload (MACPayload). Then, only the uplink frames end with a 4-octet Message Integrity Code (MIC).

The MAC payload can be:

- Useful data (MACPayload) or
- a Join-request or Rejoin-request, that the End-Device sends to the Network server or
- a Join Accept that the Network server sends to the End-Device.

6.5.2 OTA Activation of the End-Device

The Over-The-Air (OTA) Activation Procedure is used by the End-Device in order to mutually authenticate with the network and get authorised to send uplink data and to receive downlink data [75].

When the End-Device performs a successful Join or Rejoin Procedure, the End-Device is said to have a LoRaWAN session with the backend. Each LoRaWAN session is associated with a set of context parameters handled by the End-Device, and the Network Server, the Join Server and the Application Server. (e.g., session keys, DevAddr, ID of NS, etc.). The LoRaWAN session terminates when the End-Device performs Deactivation (Exit) Procedure or another successful Join/Rejoin Procedure. Figure 97 represents the OTA Activation Procedure of the End-Device.

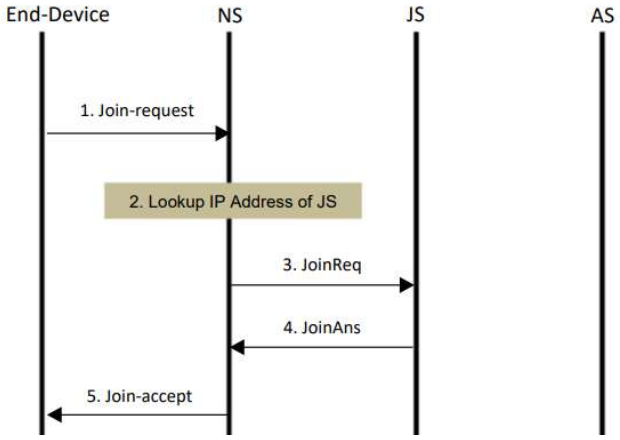

Figure 97 – Over-the-Air (OTA) Activation. Adapted from [75].

1. The End-Device transmits a Join-request message to the Network Server.
2. The Network Server uses the DNS to lookup the IP address of the Join Server based on the Join-request message.
3. The Network Server sends a JoinReq message to the Join Server carrying the LoRaWAN frame of the Join-request message, DevEUI and DevAddr.

4. The Join Server processes the Join-request message and sends JoinAns to the Network Server carrying Result=Success and the LoRaWAN frame with Join-accept payload.
5. The Network Server forwards the received LoRaWAN frame with Join-accept payload to the End-Device.

The following subsections deal with the Application layer, describing The Things Network and Akenza.

6.6 The Things Network

The Things Network is an Internet of Things ecosystem that creates networks, using LoRaWAN and it is based on The Things Stack [76]. The Things Stack, shown in Figure 98, is a LoRaWAN server that includes the Network Server, the Application Server and the Join Server functions mentioned in the LoRaWAN architecture.

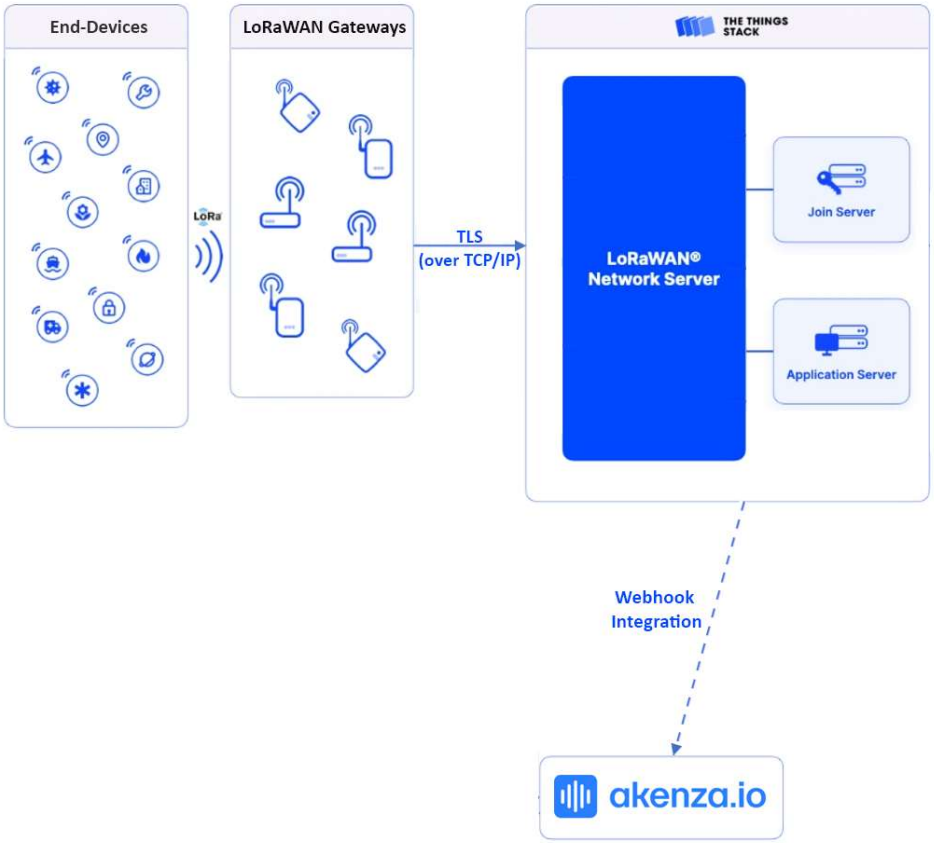

Figure 98 – The Things Stack. Adapted from [76].

The Things Stack is developed and maintained by The Things Industries, it is currently at version 3 and is, therefore, also informally known as V3 [76].

It is possible to create an account on The Things Network to start using The Things Stack Community Edition for free and The Things Stack Console.

The Console is the management application of The Things Stack for LoRaWAN. It is a web application and can be used to register end devices or gateways, to monitor network traffic or to create an integration. The aim is to graphically visualise the data and The Things Stack Console does not provide this feature. For this purpose, it was necessary to add an integration from TTN to Akenza.

6.6.1 DevEUI, DevAddr

There are two addresses associated to a device: DevEUI and DevAddr [75].

DevEUI is a 64-bit unique ID that is commonly assigned to an end-device by the manufacturer. This value is linked to the hardware and cannot be altered. Some more economic devices do not have this address already associated to them.

DevAddr, instead, identifies the end-device within the current network after the device is added to the LoRaWAN network. A DevAddr value is a 32-bit number and consists of NwkID, which identifies the specific network, followed by the NwkAddr, which is the end-device address within the specific network. DevAddr is not unique, in fact multiple devices can have the same DevAddr, but not at the same time.

There are two modes of adding and activating a device in TTN: the Activation By Personalization (ABP) and the Over-The-Air Activation (OTAA). The OTA activation is the one used in this project.

In ABP, a fixed DevAddr is hardcoded into the end-device and it remains the same for that specific network, until it is changed by the user.

In OTAA, an end-device performs a join procedure with the LoRaWAN network, during which a dynamic DevAddr is assigned to the end-device in that specific network.

6.6.2 Transport Layer Security (TLS) Protocol

The communication between the LoRaWAN gateway and the Network Server uses TLS on TCP/IP [77]. The Transport Layer Security (TLS) Protocol (Version 1.3) is on top of the transport layer.

The primary goal of TLS is to provide a secure channel for the end-to-end communication; the only requirement from the underlying transport is a reliable, in-order data stream. Since the communication between the LoRaWAN gateway and

the Network Server is on the Internet, the transport layer is TCP, because this is the one that provides the required feature. Specifically, the secure channel should provide confidentiality, because the data sent over the channel is only visible to the endpoints and integrity, as the data sent over the channel cannot be modified by attackers without being detected.

6.6.3 Webhook Integration between TTN and Akenza

The data that is sent to The Things Stack is, subsequently, sent through an integration, to Akenza's server, in order to be graphically visualised and analysed by the user [78].

Between TTN and Akenza there is a Webhook integration. Webhooks are automated messages sent from TTN when this receives new data. They are sent to a unique URL to a personal Akenza account.

They are called webhooks: web because they work over the web; hooks because they are software functions that run when something happens. Each user of an application gets a unique, random URL in order to receive webhook data.

Knowing that the data sent to TTN has to be, subsequently, sent to Akenza, it was necessary to send the data from the end device to TTN already in the format required by Akenza.

6.7 Akenza

Akenza [79] is an IoT application platform that allows to build IoT services. It controls and manages IoT devices. Devices can be managed in the "Asset" option.

The devices can be connected to Akenza's cloud in two ways:

- through device-to-server communication IoT protocols. In this specific case via MQTT.
- through integrations, which allow to manage devices of already existing IoT networks. In this specific case the integration is made with the already existing TTN created network.

During the registration process, it is necessary to create an Organization and a Workspace. An Organization is an account where a business can collaborate across many workspaces at once. A Workspace is where all the activities (Adding Data Flows, Rules or managing devices) are done.

For each device it is necessary to define a Data Flow [80]. A Data Flow indicates the data processing path by using the parameters Device Connector, Device type and Output connectors.

- The Device Connector specifies the source of the data (MQTT or TTN Integration).
- The Device type specifies how the data has to be decoded. In the specific case, it is necessary to indicate that the data coming from the TTN Integration is in the Cayenne LPP form.
- The Output connector specifies the data destination of the data flow. Storing the data in the AkenzaDB allows to have access to historical data at any time.

When creating a MQTT data flow, the topic, the MQTT username and the MQTT password are obtained and it is necessary to insert them into the code of the specific device (in this case the Raspberry) [81].

When creating the integration from TTN to Akenza, it is necessary to take from TTN the appropriate organization ID and the API key (with the necessary permissions), so that the devices that already exist in TTN can appear also on Akenza [82].

It is possible to set up Rules, which are based on logic operations, in order to trigger actions or alarm notifications [83].

A Rule consists of three components: Input, Logic, and Action [84].

- The Rule Input defines which devices provide the data to be used.
- The Rule Logic defines the conditions to be applied to the input data (for example if a value exceeds a threshold).
- The Rule Action defines which action is triggered if the condition is met (for example emails of warning can be sent from Akenza to certain recipients).

By creating a Dashboard, it is possible to graphically see the plots of the data stored in the AkenzaDB [85]. The data can be represented by using different visual components (KPI, Text, Line & Bar Chart, Table, Map and Floorplan).

Returning to the three-layered IoT architecture, this chapter analysed the second and the third layers. Belonging to the Connectivity Layer, the protocols that allow the entire device to send data to the external servers when in the Wi-Fi range (Wi-Fi and MQTT on TCP/IP) or when out of the Wi-Fi range (LoRa, LoRaWAN and TLS over TCP/IP) were described. Subsequently, the chapter dealt with the Application Layer, describing the TTN and the Akenza external servers. The next chapter is going to explore all the details of the implemented system.

7 IMPLEMENTED SYSTEM

This chapter describes the architecture of the implemented system dedicated to the elderly health monitoring system, based on a BSN. The initial configuration of the Raspberry and Akenza are also detailed and discussed. Lastly, the developed software for the Arduino, the Raspberry and the LoRa32 are analysed.

7.1 Sensors and dual-purpose button connection

This section analyses how the MCP3008 is used to connect the MA300 thermistor probe to the Raspberry, how the MPU-6050 is connected to the Raspberry, how the MAX30100 is connected to the Arduino and, lastly, how the dual-purpose button is connected to the Raspberry.

7.1.1 Connection of the MA300 to the Raspberry using the MCP3008

In order to calculate the value of the temperature, it is firstly necessary to calculate the value of the resistance of the NTC. With this purpose, a NTC thermistor was included in a voltage divider, as shown in Figure 99.

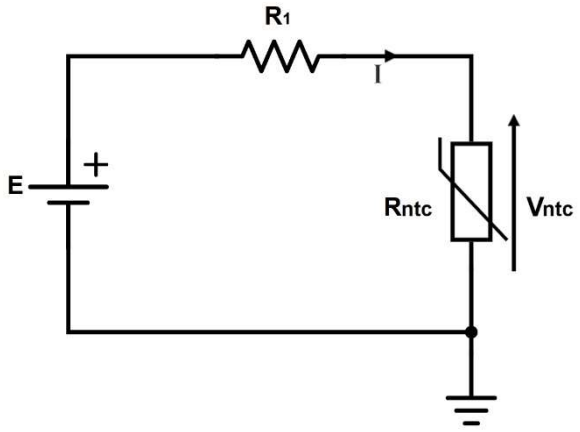

Figure 99 – The Voltage Divider circuit for Rntc.

By applying the formula of the voltage divider,

$$V_{ntc} = E \cdot \frac{R_{ntc}}{R_{ntc} + R_1}$$

it is possible to extract the value of R_{ntc}.

$V_{ntc} \cdot R_1 + V_{ntc} \cdot R_{ntc} = E \cdot R_{ntc}$

$$V_{ntc} \cdot R_1 = E \cdot R_{ntc} - V_{ntc} \cdot R_{ntc}$$

$$V_{ntc} \cdot R_1 = R_{ntc} \cdot (E - V_{ntc})$$

$$R_{ntc} = \frac{V_{ntc} \cdot R_1}{E - V_{ntc}}$$

The values of E and R_1 are already known and it is possible to measure the voltage V_{ntc} using an ADC. An external ADC has been added to the system, given that the Raspberry does not include this functionality.

Since the ADC has as reference the same ground voltage, it is only necessary to connect it as in Figure 100.

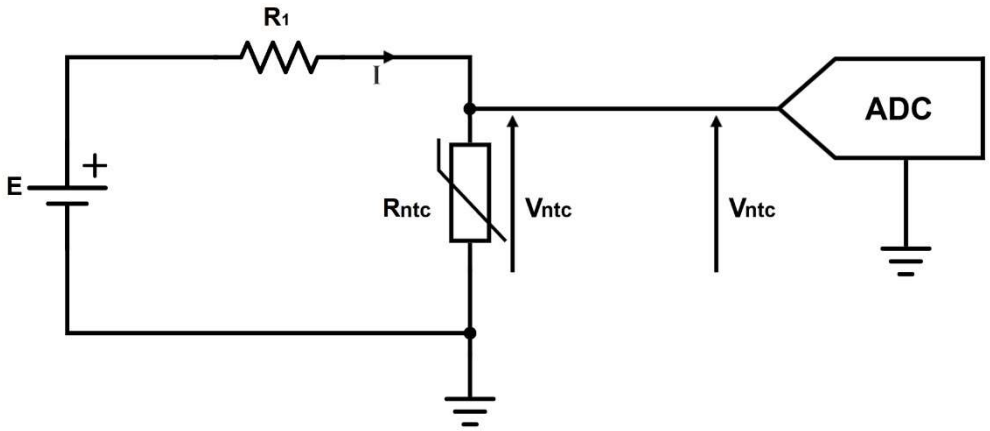

Figure 100 – How the Thermistor is connected to the ADC.

Figure 101 graphically represents how the MCP3008 is connected to the Raspberry.

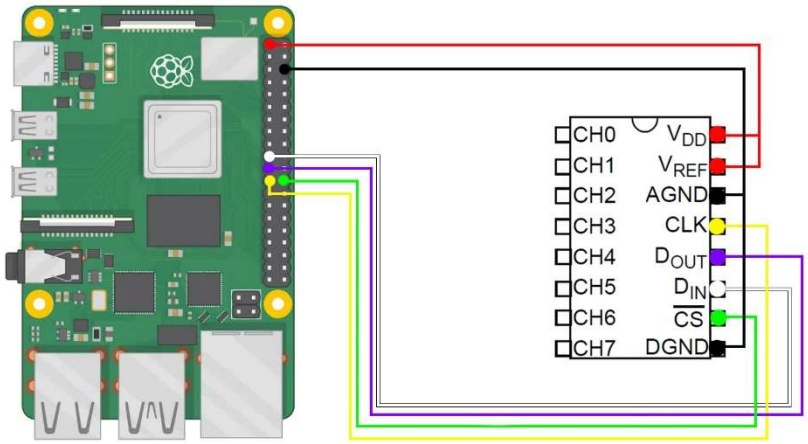

Figure 101 – Connection of the MCP3008 to the Raspberry.

Table 24 shows how the pins of the MCP3008 are connected to the respective pins of the Raspberry.

Table 24 – Connection of the MCP3008 pins to the Raspberry pins.

MCP3008 Pin	Raspberry Pin	Colour of the wire
VDD →	3.3V (Pin 1)	Red
VREF →	3.3V (Pin 1)	Red
AGND →	Ground (Pin 6)	Black
CLK →	SCLK (Pin 23)	Yellow
DOUT →	MISO (Pin 21)	Purple
DIN →	MOSI (Pin 19)	White
CS →	SS (Pin 24)	Green
DGND→	Ground (Pin 6)	Black

7.1.2 Connection of the MPU-6050 to the Raspberry

Figure 102 shows how the MPU-6050 is connected to the Raspberry.

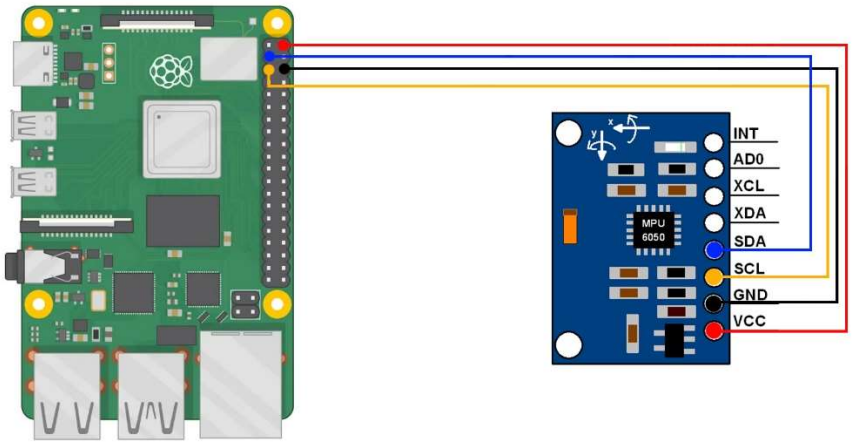

Figure 102 – Connection of the MPU-6050 to the Raspberry.

Table 25 shows how the pins of the MPU-6050 are connected to the pins of the Raspberry.

Table 25 – Connection of the MPU-6050 pins to the Raspberry pins.

MPU-6050 Pin	Raspberry Pin	Colour of the wire
Vcc →	5V (Pin 2)	Red
GND →	Ground (Pin 6)	Black
SCL →	SCL (Pin 5)	Orange
SDA →	SDA (Pin 3)	Blue

7.1.3 Connection of the MAX30100 to the Arduino

Figure 103 illustrates how the MAX30100 is connected to the Arduino.

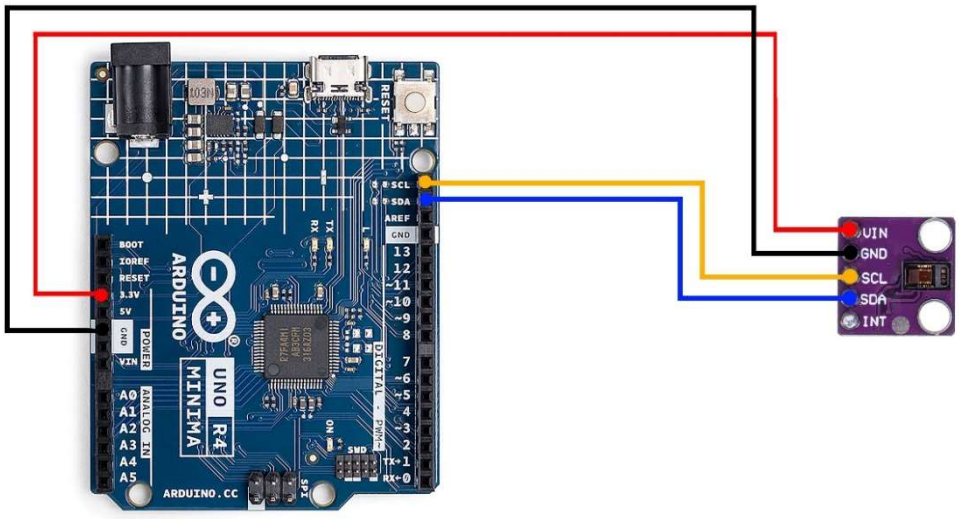

Figure 103 – Connection of the MAX30100 to the Arduino.

Table 26 shows how the pins of the MAX30100 are connected to the pins of the Arduino.

Table 26 – Connection of the MAX30100 pins to the Arduino pins.

MAX30100 Pin	Arduino Pin	Colour of the wire
VIN →	3.3V	Red
GND →	GND	Black
SCL →	SCL	Orange
SDA →	SDA	Blue

7.1.4 Connection of the Button to the Raspberry

Figure 104 illustrates how the button is connected to the Raspberry.

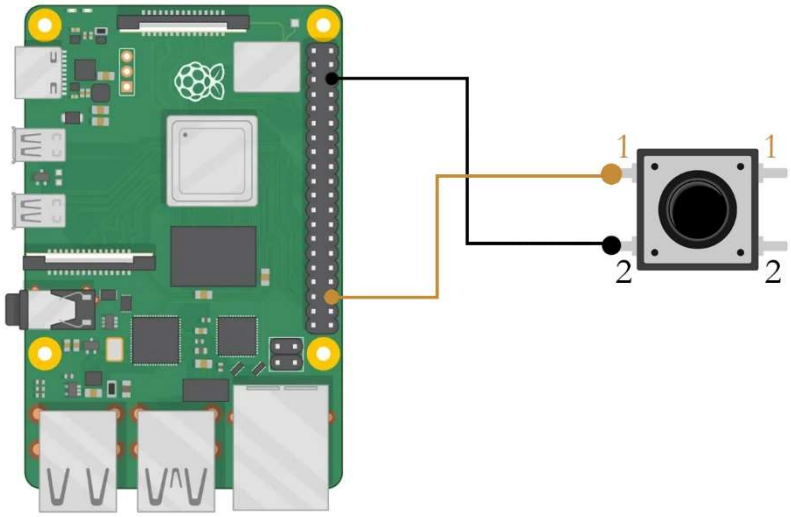

Figure 104 – Connection of the Button to the Raspberry.

Table 27 shows how the button pins are connected to the pins of the Raspberry.

Table 27 –Connection of the Button pins to the Raspberry pins.

Button Pin	Raspberry Pin	Colour of the wire
Pin1 →	GPIO16	Brown
Pin2 →	Ground (Pin 6)	Black

7.2 Initial configuration of the Raspberry

Since the Raspberry used in this project was completely new, the initial configuration was necessary. The first step was to install the operating system on the MicroSD card. In this specific case a 32GB MicroSD card was used. The Raspberry Pi Imager is the quickest and easiest way to install the Raspberry Pi OS on a MicroSD card. The Raspberry Pi OS (previously called Raspbian) is the recommended operating system. This operating system was selected and installed by using the aforementioned Raspberry Pi Imager, as shown in Figure 105.

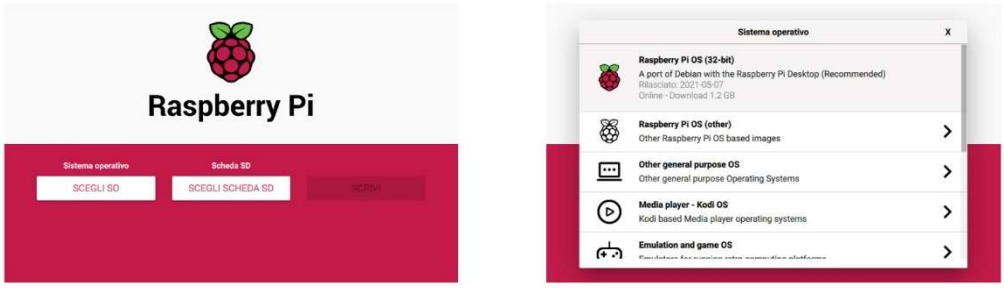

Figure 105 – Choosing the Raspberry Pi OS on the Raspberry Pi Imager.

After having installed the operating system, the next step comprises the insertion of the MicroSD card in the Raspberry; thereafter everything was ready for use. The simplest way to utilise the Raspberry is by connecting a monitor, a keyboard and a mouse to it. However, in this project, it was decided to use it remotely through a MacBook. Therefore, the procedure was as follows: to write an empty text file named "ssh" (no file extension) to the root of the directory of the card. When the Raspberry Pi OS sees the "ssh" on its first boot-up, it automatically enables the SSH (Secure Socket Shell) protocol, which allows a computer connected to the same network as the Raspberry to remotely access the Pi command line interface.

The subsequent step was to connect the Raspberry to the Wi-Fi network which the MacBook is connected to. To proceed this way, it was necessary to create a text file called *wpa_supplicant.conf* and to place it in the root directory of the microSD card. In the file the text shown below was inserted:

```
country=PT
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
scan_ssid=1
ssid="your_wifi_ssid"
psk="your_wifi_password"
}
```

After having inserted the MicroSD card containing the installed operating system, the ssh file and the *wpa_supplicant.conf* file that was previously created, and after powering the Raspberry through its USB-C port, the Raspberry is turned on and is connected to the Wi-Fi network. At this point, it was necessary to establish an SSH connection. First of all, the software PuTTY Configuration was downloaded (since this software is only available on Windows, this operation was performed on another computer). The PuTTY Configuration is illustrated in Figure 106a.

The host name or the IP address of the specific desired device was inserted in PuTTY. After entering its host name *raspberrypi* and clicking on Open, the Raspberry Command Prompt seen from PuTTY appeared (Figure 106b).

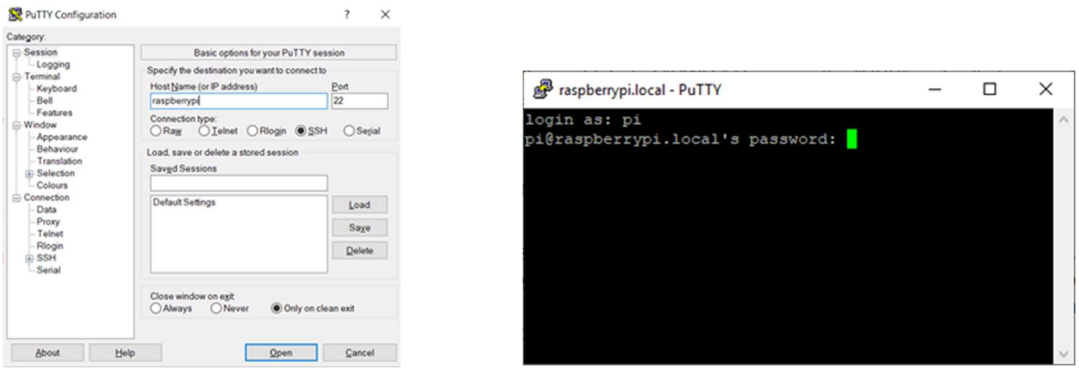

Figure 106 – a) PuTTY Configuration. b) Raspberry Command Prompt seen from PuTTY.

The word *pi* was inserted as the username and *raspberry* as the password to do the first login. All these steps until now were necessary to be able to enter remotely in the command prompt of the Raspberry.

By writing *sudo raspi-config* in the command prompt, it was possible to access the Raspberry Pi Software Configuration Tool. Once having accessed it, in the section interface, the Virtual Network Computing (VNC) interface was enabled.

The VNC interface allows, through the use of the software VNC Viewer, which was installed on the MacBook, to remotely control the Raspberry through its graphical interface.

Figure 107 shows how to access the graphical interface of the Raspberry using VNC Viewer.

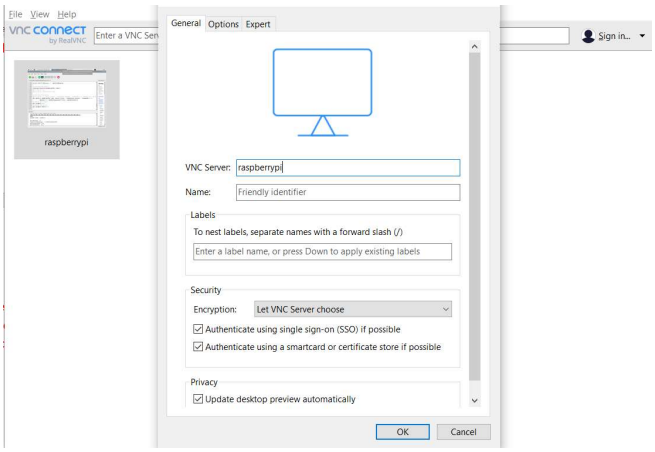

Figure 107 – The VNC Viewer Software.

7.3 Getting started with TTN version 3

This section describes the procedure to configure the version 3 of TTN. The first step was to install the Arduino IDE on the MacBook. The Arduino IDE is the software used to program the TTGO LoRa32.

Installing the Board TTGO LoRa32 in the Arduino IDE

Firstly, it was necessary to add the Board TTGO LoRa32 to the Arduino IDE, because the boards with processor ESP32 were not already present.

On the Arduino IDE, File → Preferences was selected. On the Additional boards manager URLs field, https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json was inserted (Figure 108a).

After selecting the Boards Manager icon in the left-side corner of the Arduino IDE, the ESP32 was chosen to be installed (Figure 108b).

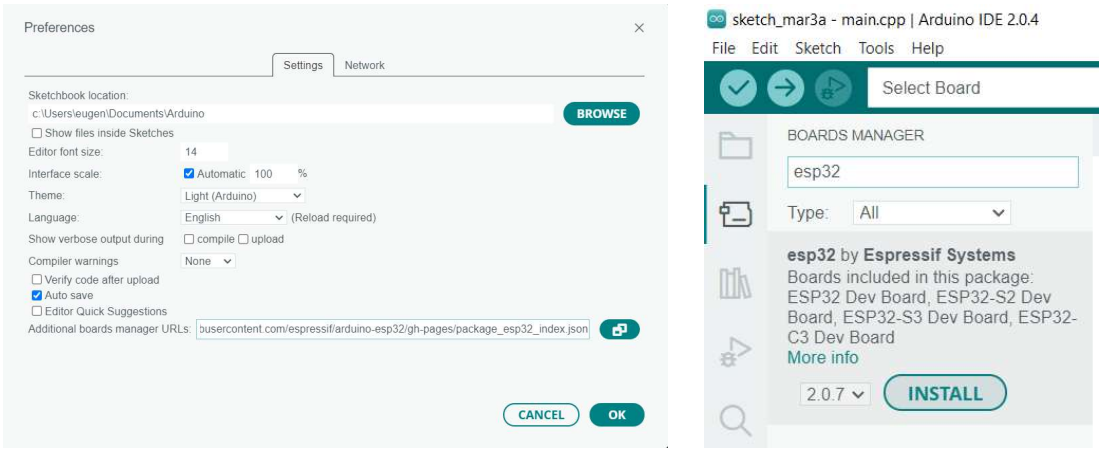

Figure 108 – a) Arduino IDE, File → Preferences. b) Arduino IDE, Boards Manager.

On Select board → Select Other Board and Port, TTGO LoRa32-OLED was chosen (Figure 109).

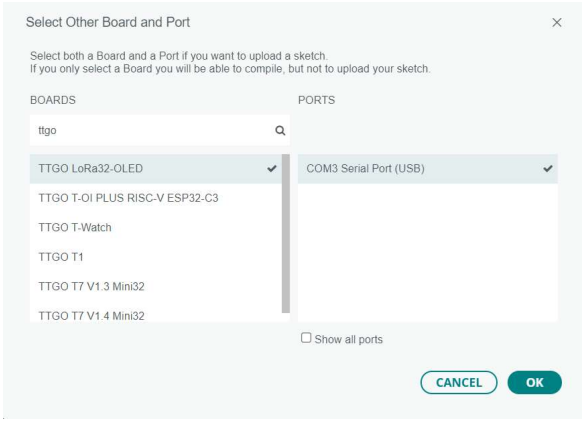

Figure 109 – Arduino IDE, Select board → Select Other Board and Port.

Installing the library mcci lmic

After the installation of the board, it was necessary to search for the correct library mcci lmic and to install it (Figure 110).

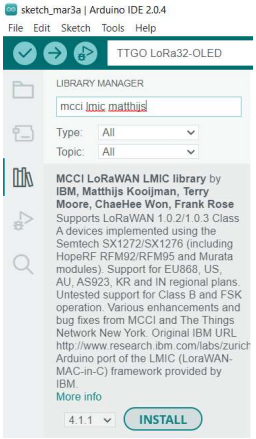

Figure 110 – Arduino IDE, Library Manager.

After installing the library, it was necessary to do some changes on it.

In \Arduino\libraries\MCCI_LoRaWAN_LMIC_library\project_config in the file *lmic_project_config.h*, the European setting was chosen and the line *#define hal_init LMICHAL_init* was added, as shown in Figure 111.

```
// project-specific definitions
#define CFG_eu868 1
//#define CFG_us915 1
//#define CFG_au915 1
//#define CFG_as923 1
// #define LMIC_COUNTRY_CODE LMIC_COUNTRY_CODE_JP      /* for as923-JP; also define CFG_as923 */
//#define CFG_kr920 1
//#define CFG_in866 1
#define CFG_sx1276_radio 1
//#define LMIC_USE_INTERRUPTS

#define hal_init LMICHAL_init
```

Figure 111 – File *lmic_project_config.h*.

Code to send data from the TTGO LoRa32 to TTN

After the Arduino IDE was ready, it was necessary to write the code to send data from the TTGO LoRa32 to TTN. The example code in ZIP format was downloaded from https://github.com/squix78/TTGO-LoRa32-V1.0-TTN-OTAA and then extracted. This example code only contained the instructions to connect the TTGO LoRa32 to TheThingsNetwork over OTAA and to send simple data to TTN.

In the Arduino program, in Sketch → Add File the file \TTGO-LoRa32-V1.0-TTN-OTAA-master\TTGO-LoRa32-V1.0-TTN-OTAA-master\src\main.cpp was opened.

After copying the example code of the *main.cpp* file, and after pasting the code in a new sketch, this new file was saved as "main" (the format of the file is automatically .ino).

In order for the example code to work, firstly it was necessary to create the account on TTN and then to adapt the example code to make it able to send data to the specific TTN account.

Configuring TTN

At this point, on https://console.cloud.thethings.network/ the correct V3 region was selected and an account on TTN was created. Then, it was necessary to create an application on TTN, selecting +Add application (Figure 112).

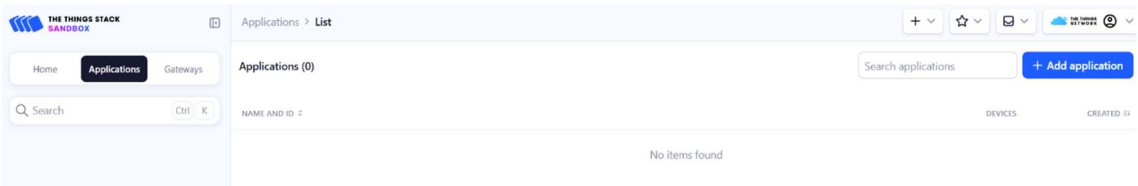

Figure 112 – TTN, Applications.

An Application ID and an Application Name were required before selecting Create application (Figure 113).

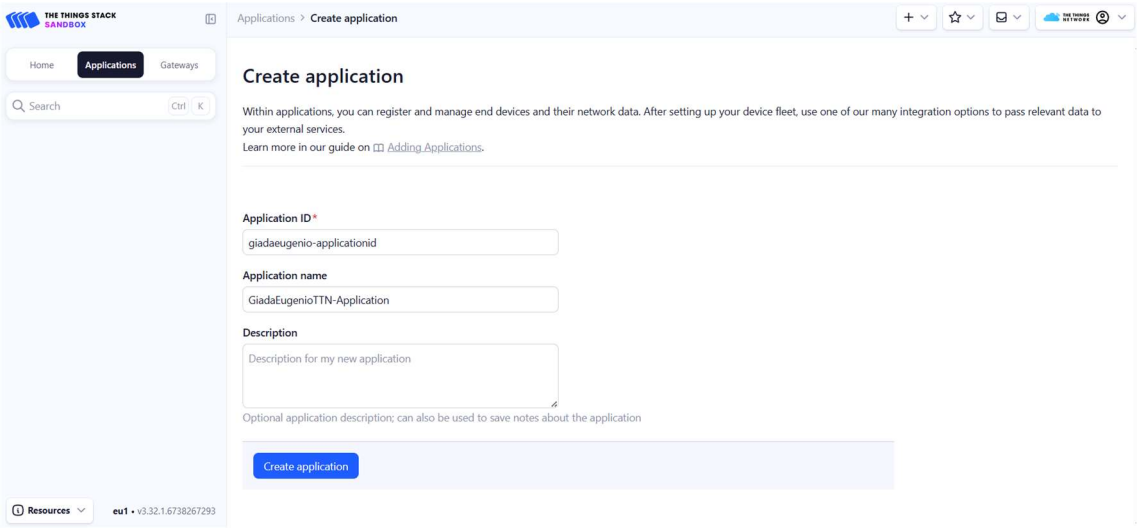

Figure 113 – TTN, Create application.

At this point it was necessary to create the LoRa32 device so, in Application overview, +Register end device was selected (Figure 114).

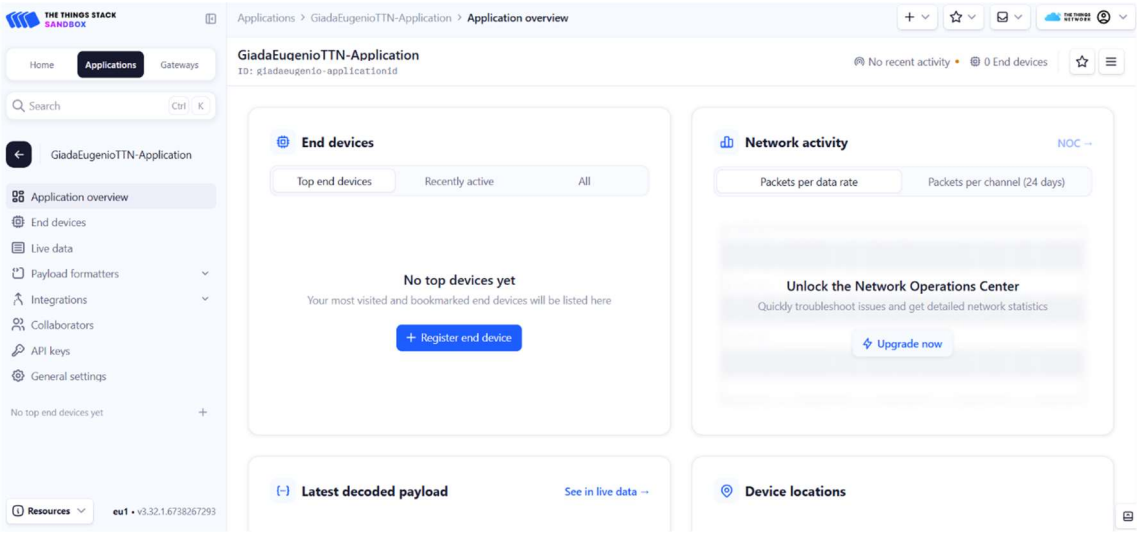

Figure 114 – TTN, Application overview.

DevEUI was already described in Subsection 6.6.1. Since the LoRa32 does not have a DevEUI, it was necessary to generate it on TTN.

The JoinEUI (AppEUI) is a 64-bit globally unique identifier assigned to the LoRaWAN network's Join Server. It identifies the Join Server that will be employed for the join procedure. As it was not already provided by the manufacturer of the LoRa32, it was mandatory to fill it with all zeros.

The AppKey is the encryption key used for messages during every over the air activation. It can be randomly generated by TTN and has to be known by both TTN and the LoRa32.

In order to create the LoRa32 device, it was necessary to choose the European LoRaWAN frequencies (Figure 115), to insert the JoinEUI (Figure 116) and to generate the DevEUI and the AppKey, before selecting Register end device (Figure 117).

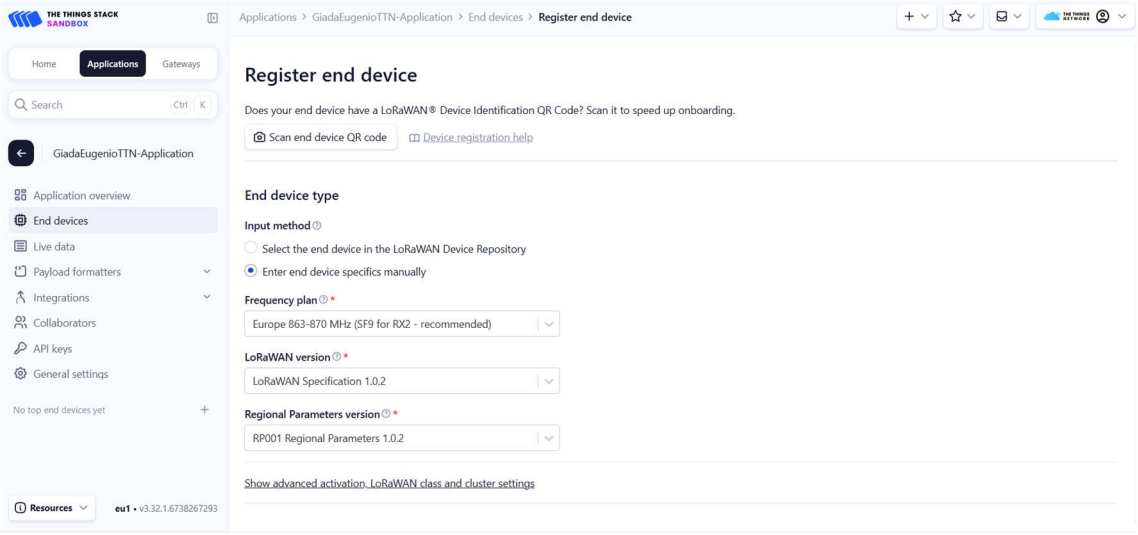

Figure 115 – TTN, Register end device (part 1).

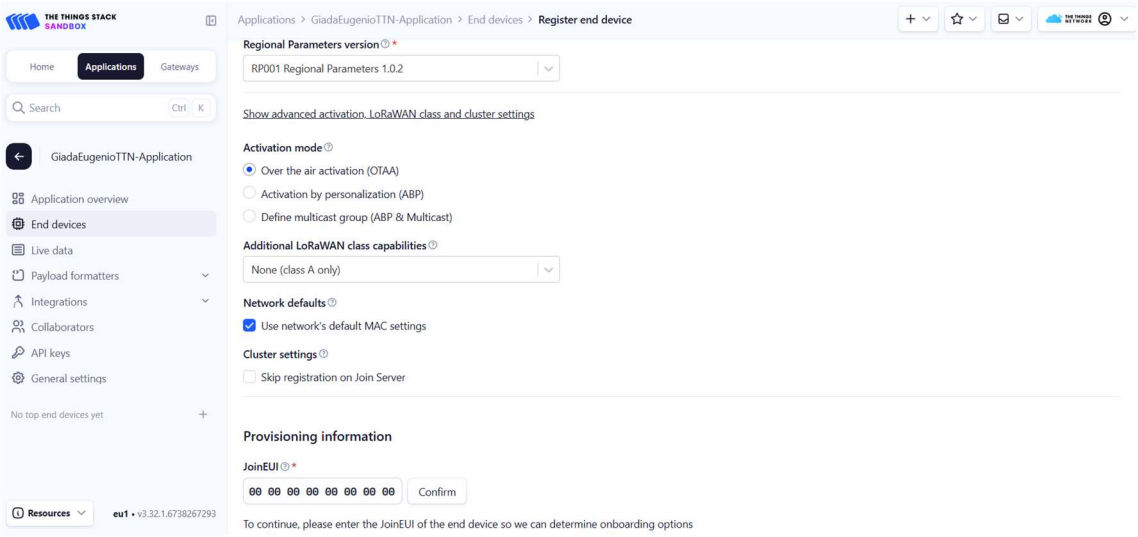

Figure 116 – TTN, Register end device (part 2).

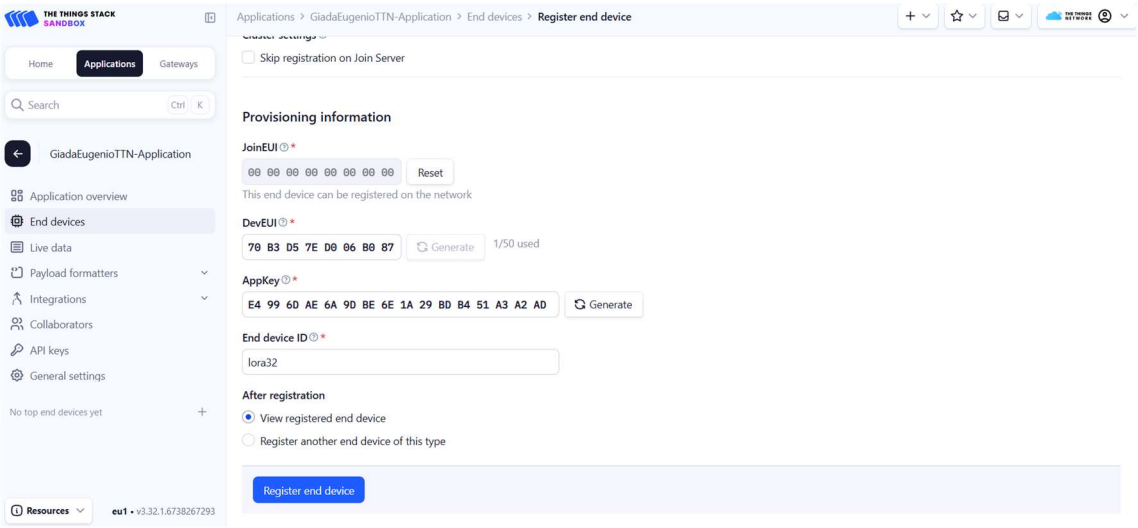

Figure 117 – TTN, Register end device (part 3).

In Device overview, AppEUI and DevEUI were ordered to start with their lsb, while AppKey had to start with its msb, before copying them (Figure 118).

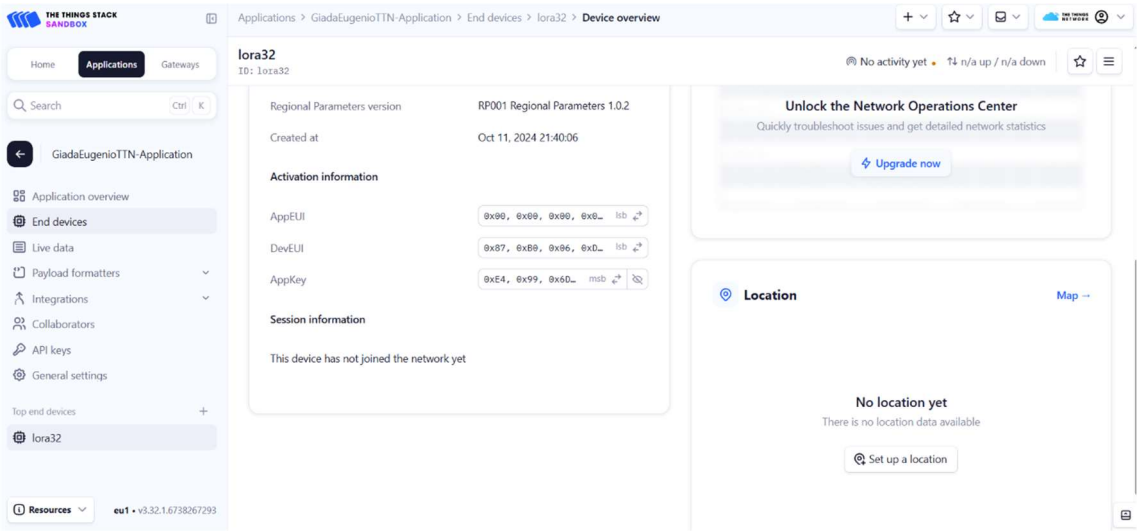

Figure 118 – TTN, LoRa32 Device overview.

Adapting the example code

The personal AppEUI, DevEUI and AppKey obtained on the TTN website were inserted into the example code (Figure 119).

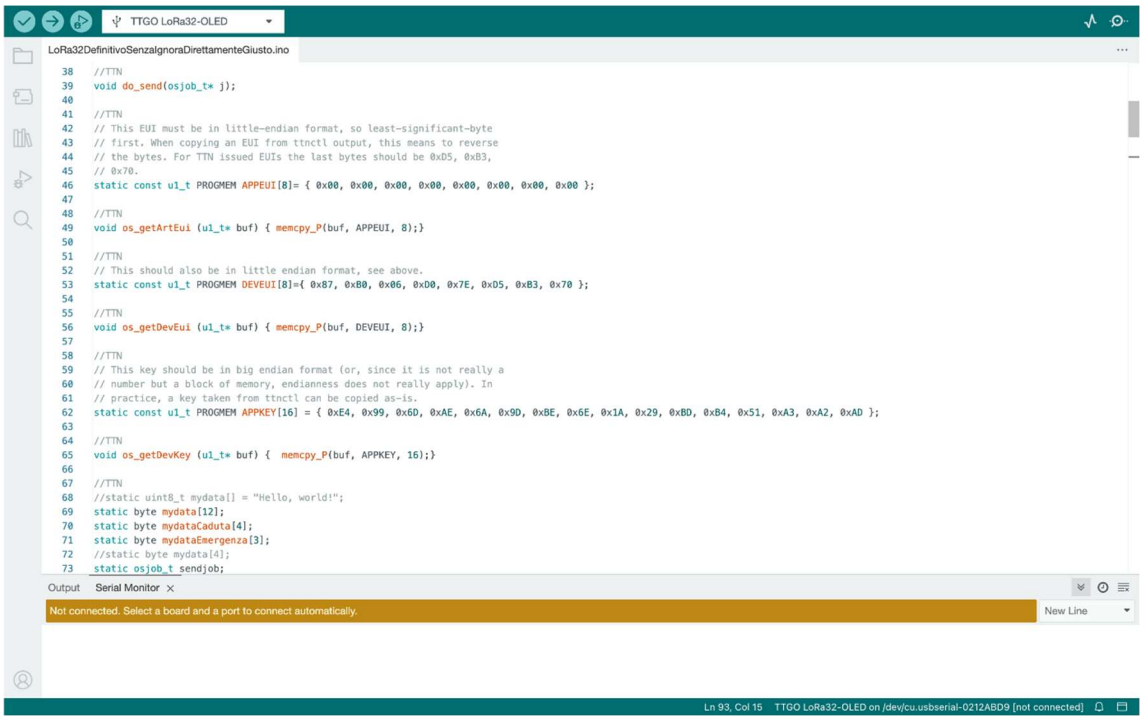

Figure 119 – Arduino IDE, inserting AppEUI, DevEUI and AppKey into the code.

In the adapted code, it was also necessary to write the correct pins of the specific TTGO LoRa32 board (Figure 120).

Figure 120 – Arduino IDE, writing the correct pins of the specific TTGO LoRa32.

Based on the example code, the complete code was written and then compiled and uploaded to the TTGO LoRa32.

7.4 Cayenne LPP Formatter Type

Akenza allows to graphically display the data sent to TTN. Since Akenza accepts the Cayenne Low Power Payload (LPP) standard, it was decided to directly send the data to TTN in this format [86].

The Cayenne Low Power Payload provides a convenient and easy way to send data over LPWAN networks such as LoRaWAN. The Cayenne LPP allows the device to send multiple sensor data at one time. Table 28 illustrates the structure of the Payload.

Table 28 – Structure of the Cayenne Low Power Payload.

1 Byte	**1 Byte**	**N Bytes**	**1 Byte**	**1 Byte**	**M Bytes**	**…**
Data1 Ch.	Data1 Type	Data1	Data2 Ch.	Data2 Type	Data2	…

Data Channel is the channel of the respective widget used on Cayenne. The channels are not perceived by Akenza, so this field is ignored.

Data Type identifies the respective data type that is being sent in the frame, e.g. “temperature”. Table 29 shows all the possible Data Types.

Table 29 – Data Types of the Cayenne LPP.

Type	Hex	Data Size	Data Resolution per bit
Digital Input	0	1	1
Digital Output	1	1	1
Analog Input	2	2	0.01 Signed
Analog Output	3	2	0.01 Signed
Illuminance Sensor	65	2	1 Lux Unsigned MSB
Presence Sensor	66	1	1
Temperature Sensor	67	2	0.1 °C Signed MSB
Humidity Sensor	68	1	0.5 % Unsigned
Accelerometer	71	6	0.001 g Signed MSB per axis
Barometer	73	2	0.1 hPa Unsigned MSB
Gyroscope	86	6	0.01 °/s Signed MSB per axis
GPS Location	88	9	Latitude: 0.0001 ° Signed MSB Longitude: 0.0001 ° Signed MSB Altitude: 0.01 m Signed MSB

It is possible to send data collected by different sensors in the same frame, by specifying the respective channel and type for each of them.

7.4.1 The Cayenne LPP in TTN

It was necessary to indicate on TTN that the type of data that TTN will receive will be in the Cayenne LPP format. Therefore, on Payload formatters → Uplink, Cayenne LPP was chosen as Formatter Type (Figure 121).

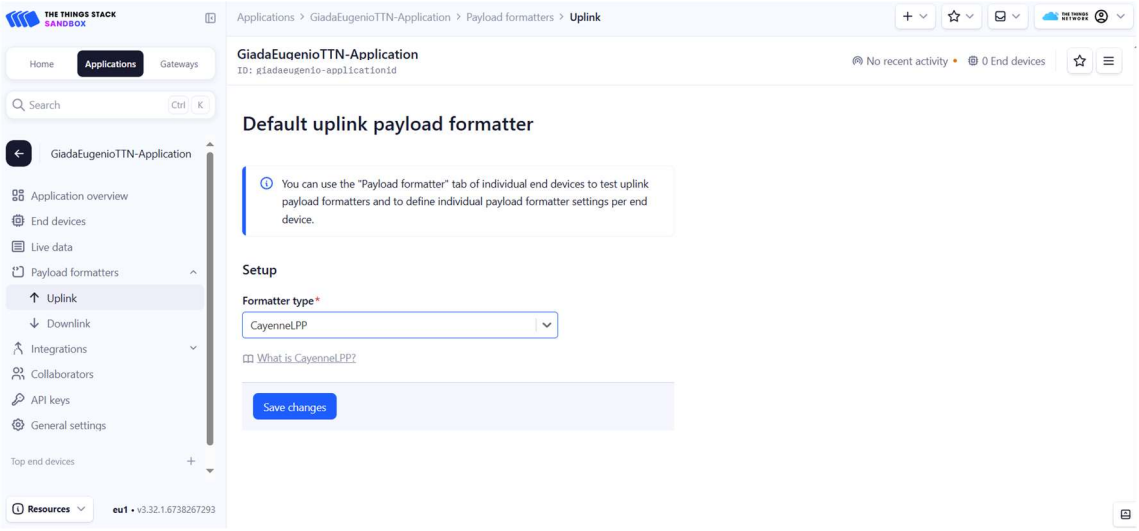

Figure 121 – TTN, setting the Cayenne LPP Formatter type.

This step is needed to visualise also in TTN the data in the correct way. If this step is skipped, for example a temperature value of 27.2 °C sent to channel 3, will be visualized on TTN as "03 67 01 10", because TTN does not understand the real meaning behind those numbers. This example is explained in Table 30.

Table 30 – Example of the Cayenne LPP format.

Payload (Hex)	03 67 01 10	
Data Channel	**Type**	**Value**
03 ⇒ 3	67 ⇒ Temperature	0110=272 ⇒ 27.2 °C

By specifying, instead, to TTN that a payload is being sent in the Cayenne LPP format, it is possible to visualise on TTN the channel, type and the value of the data.

7.5 Getting started with Akenza

This section describes the procedure that was carried out to configure Akenza.

After creating an account on the Akenza website and after doing the login, Start the setup was selected (Figure 122).

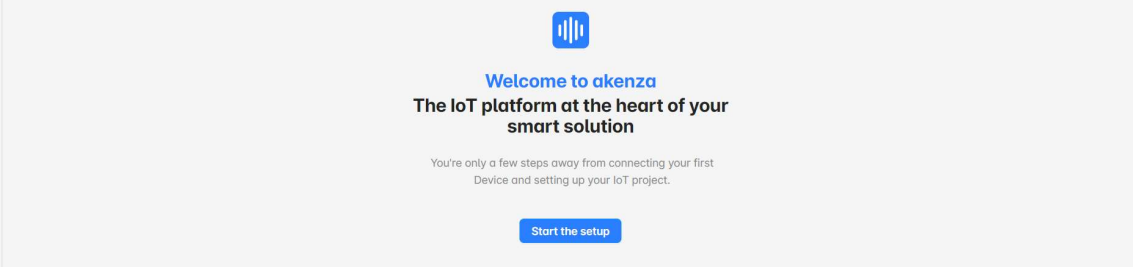

Figure 122 – Akenza, welcome page.

The first step to use Akenza, was to create an Organization, a space where a business can collaborate across many workspaces at once. The name of the organization was inserted before selecting Create Organization (Figure 123a).

The next step was to create a Workspace, which is the space where the actual work happens. All the activities of Adding Data Flows, Rules or managing devices are, in fact, done within a workspace. A Workspace name was inserted before selecting Create Workspace (Figure 123b).

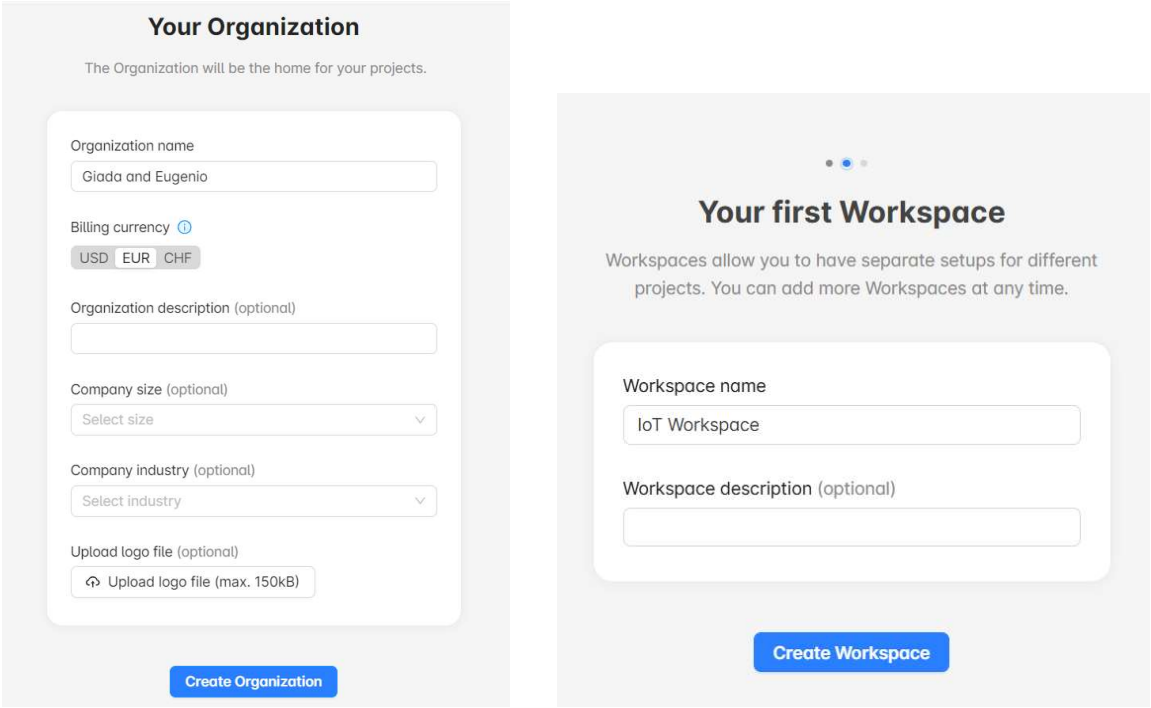

Figure 123 – a) Akenza, creating an Organization. b) Akenza, creating a Workspace.

At this point it was necessary to connect the Raspberry to Akenza, so Connect your first device → Start now was selected (Figure 124a).

Data Flows define which is the data source and how this data is processed and then stored in Akenza. In order to create the first Data Flow, +Create Data Flow was selected on the menu tab Data Flows (Figure 124b).

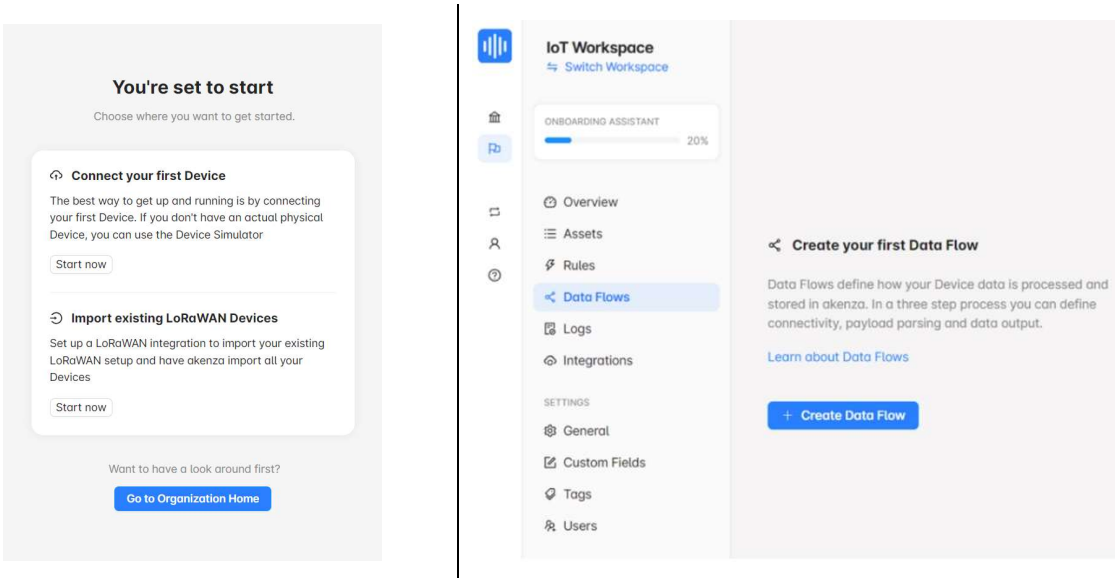

Figure 124 – a) Akenza, getting started. b) Akenza, Data Flows.

The Raspberry has to send the data via MQTT, so Connect a Device over MQTT was chosen (Figure 125).

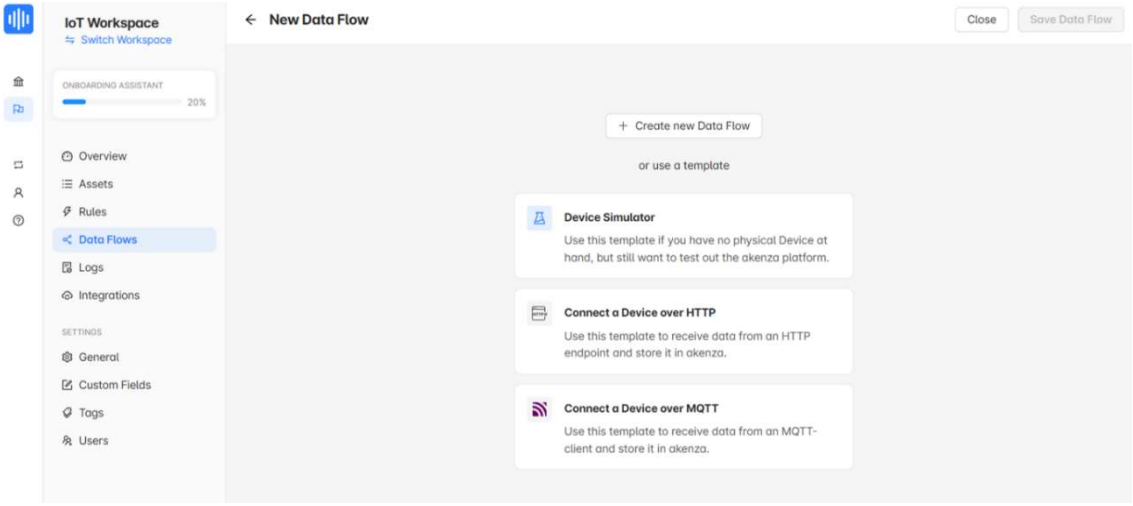

Figure 125 – Akenza, creating a Data Flow.

This specific data flow takes the data from a MQTT source (the Raspberry) and saves it on Akenza's Database without doing any kind of processing on it (Figure 126).

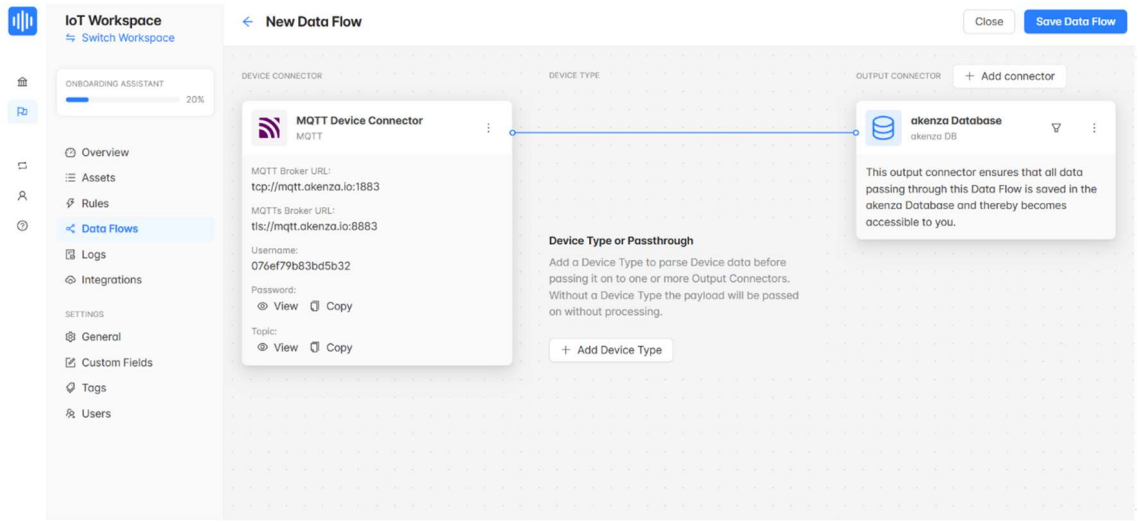

Figure 126 – Akenza, creating the MQTT Data Flow.

After creating the MQTT Data Flow, it was necessary to create the Raspberry device and to assign to it this Data Flow.

While saving the created Data Flow, the option Create Device with this Data Flow was selected (Figure 127).

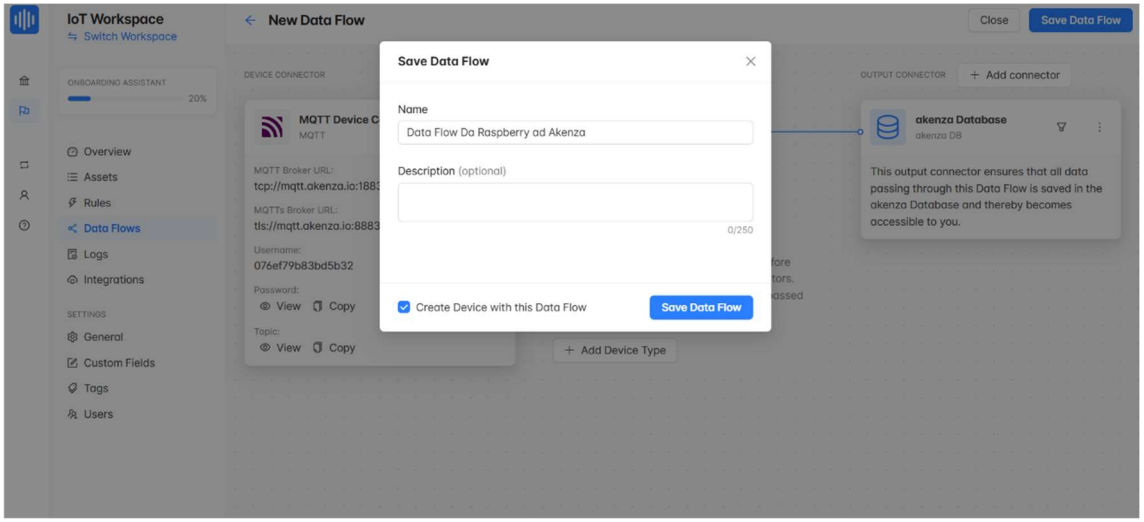

Figure 127 – Akenza, saving the MQTT Data Flow.

"Raspberry" was inserted as the name of the device (Figure 128).

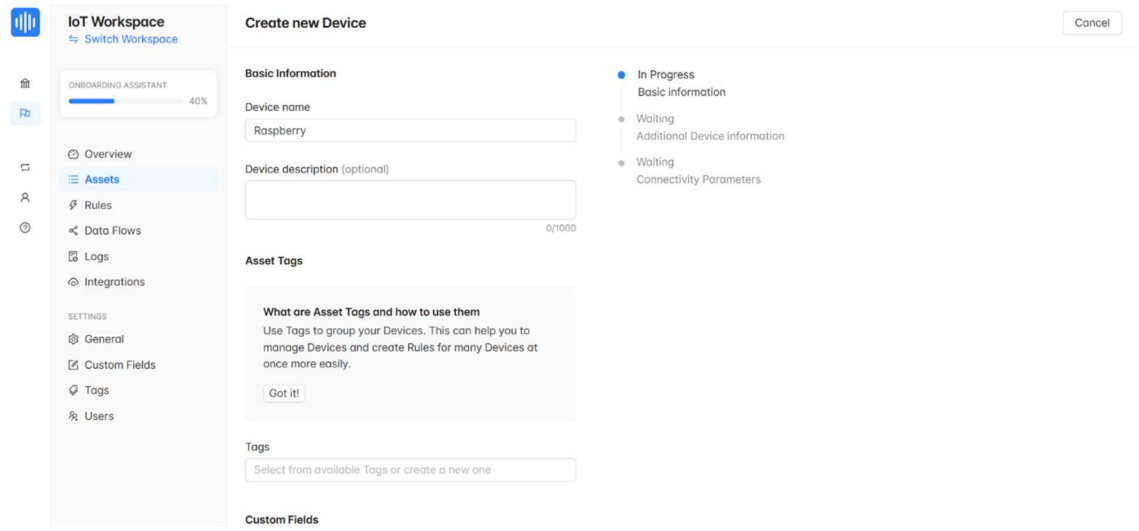

Figure 128 – Akenza, creating the Raspberry Device (part 1).

The created MQTT Data Flow was chosen (Figure 129).

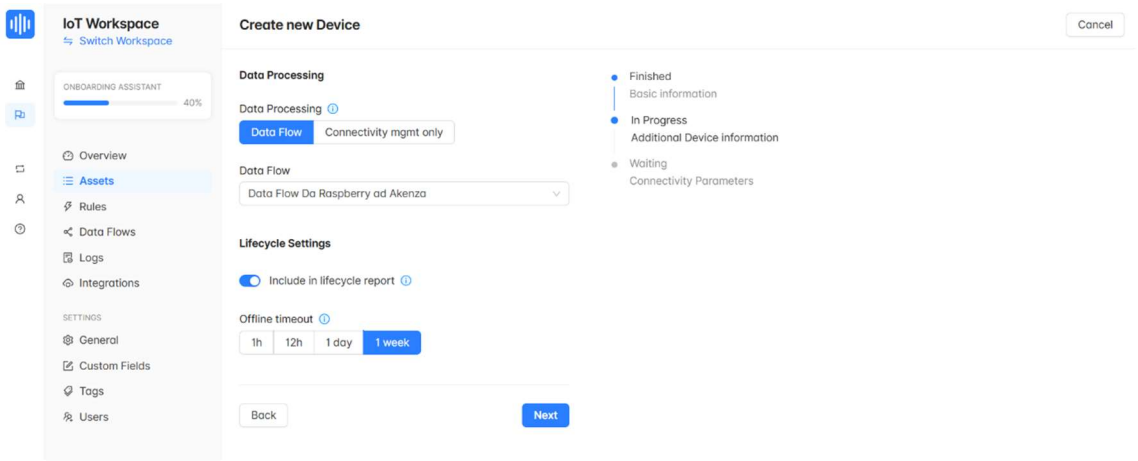

Figure 129 – Akenza, creating the Raspberry Device (part 2).

After selecting Generate ID, the Raspberry device was finally created (Figure 130 and Figure 131).

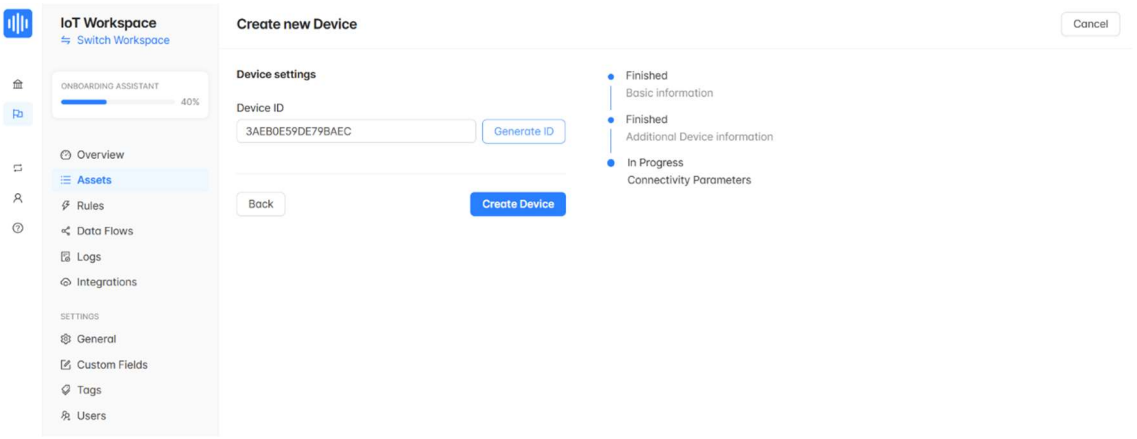

Figure 130 – Akenza, generating the Raspberry Device ID.

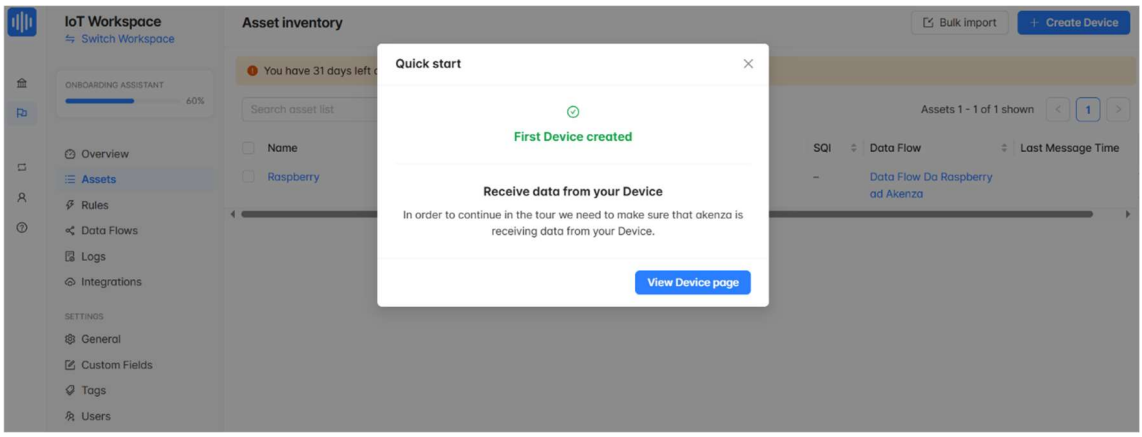

Figure 131 – Akenza, the Raspberry Device created.

Then on the command prompt of the Raspberry the line *pip3 install paho-mqtt* was inserted to install the libraries that allow the communication with Akenza via MQTT. Subsequently, in order for the Raspberry to send data to the specific Akenza account, it was necessary to write in its code the MQTT username, the MQTT password and the Topic that were obtained on Akenza (Figure 132).

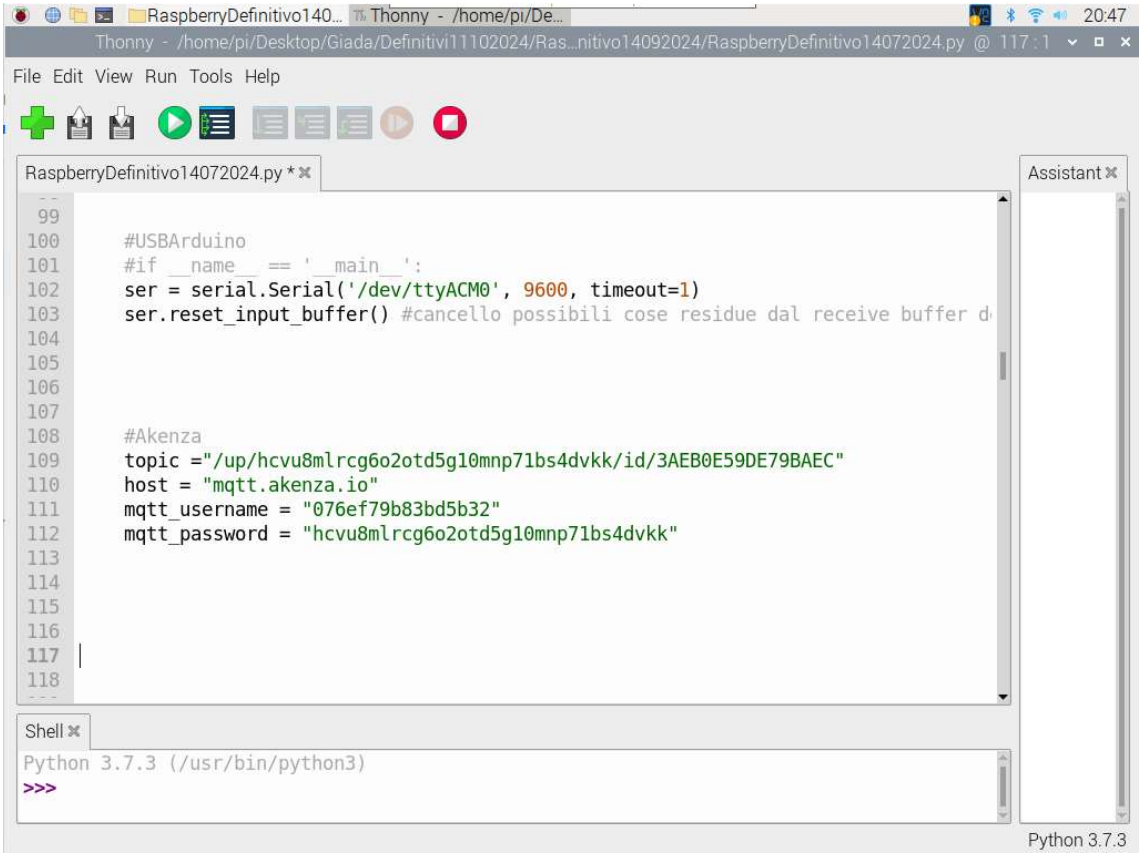

Figure 132 – Inserting the MQTT details into the Raspberry code.

7.5.1 Akenza Dashboard

In the Dashboard Overview, Start new was selected to create the personal Dashboard (Figure 133).

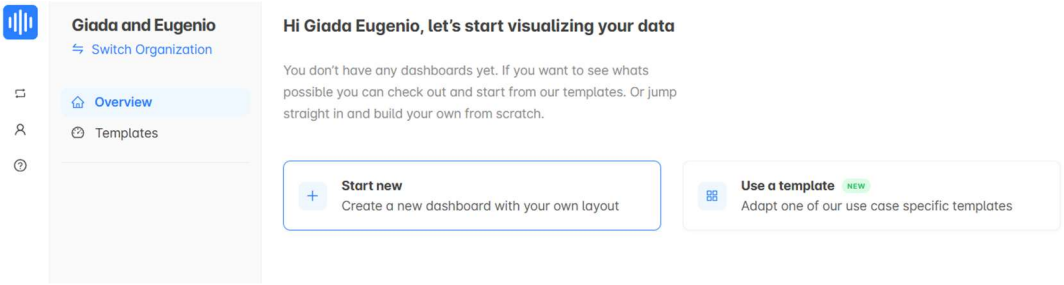

Figure 133 – Akenza, Dashboard homepage.

After inserting the name of the dashboard (Figure 134), it was necessary to define the Dashboard scope, which is the specific Workspace where the data to visualise is taken from (Figure 135).

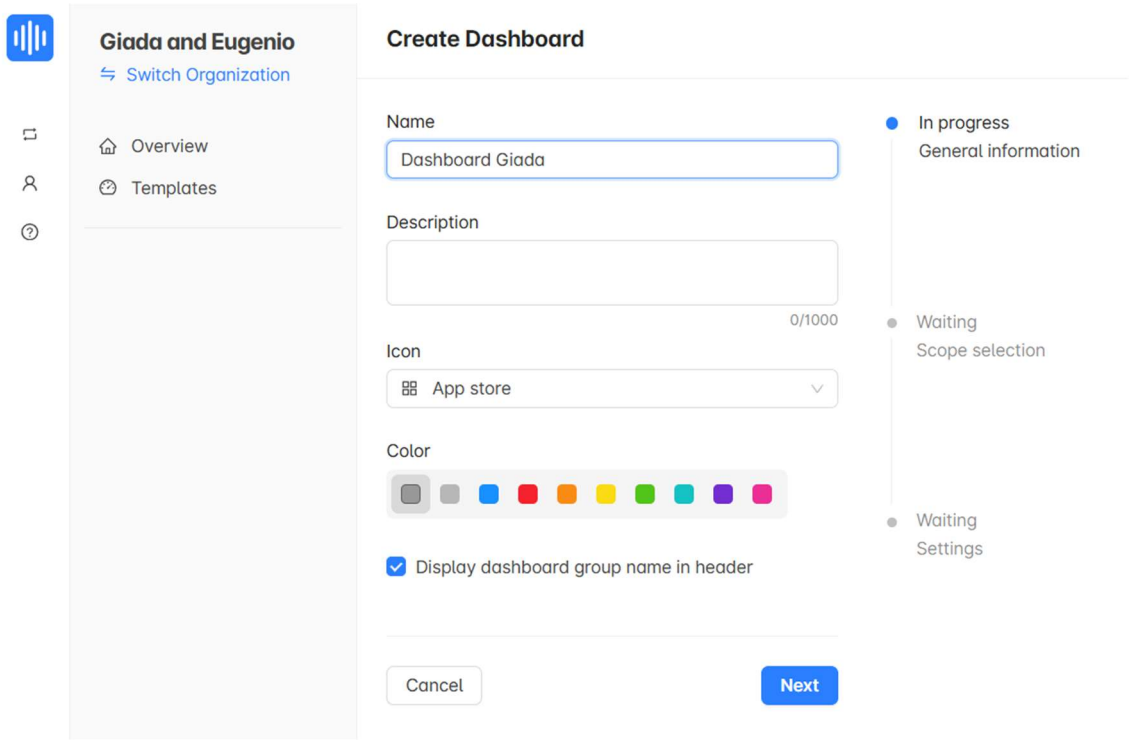

Figure 134 – Akenza, creating the Dashboard (General information).

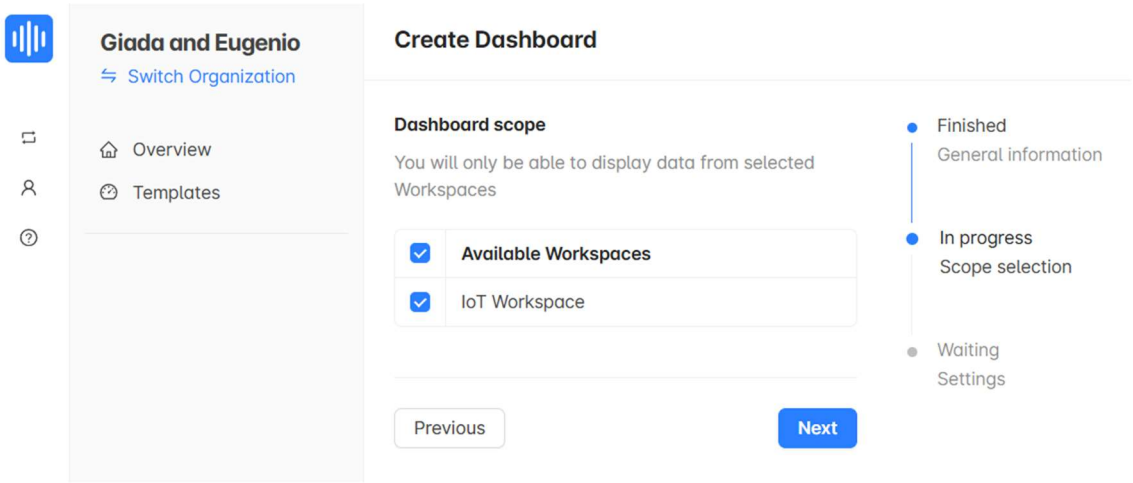

Figure 135 – Akenza, creating the Dashboard (Scope selection).

Finally, the default time range of the Dashboard and its Refresh time were chosen (Figure 136).

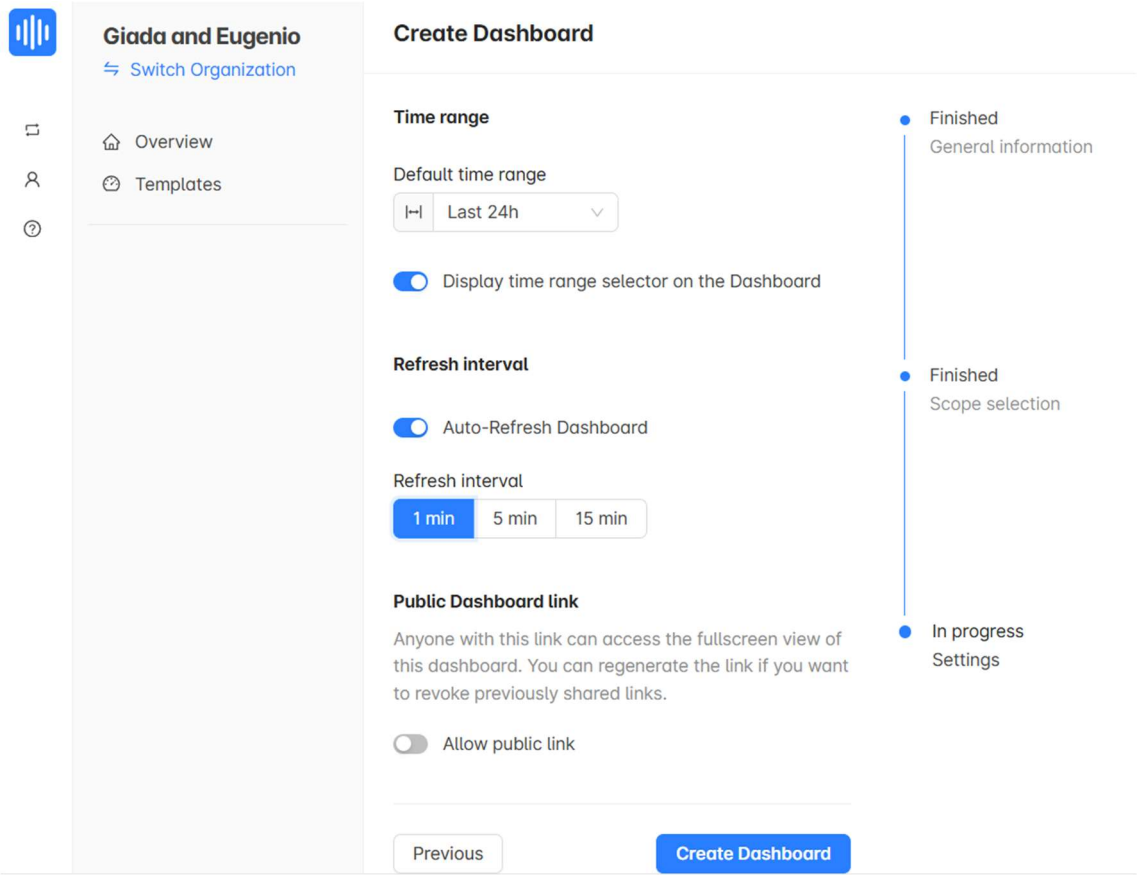

Figure 136 – Akenza, creating the Dashboard (Settings).

7.6 Integration between TTN and Akenza

This section describes how the Integration between TTN and Akenza was created. To create it, it was necessary to take the Application ID and the API key from the personal TTN and to insert them in Akenza.

The TTN Application ID was taken directly from Application → List on TTN (Figure 137).

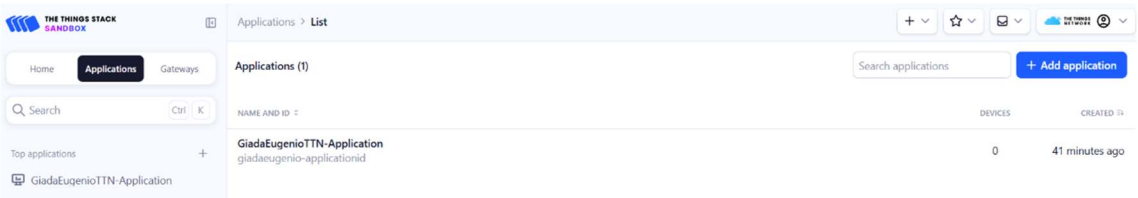

Figure 137 – TTN, Application ID.

The API key had to be created on TTN, by selecting API Keys → +Add API key. After choosing the correct settings (Figure 138), Create API key was selected (Figure 139).

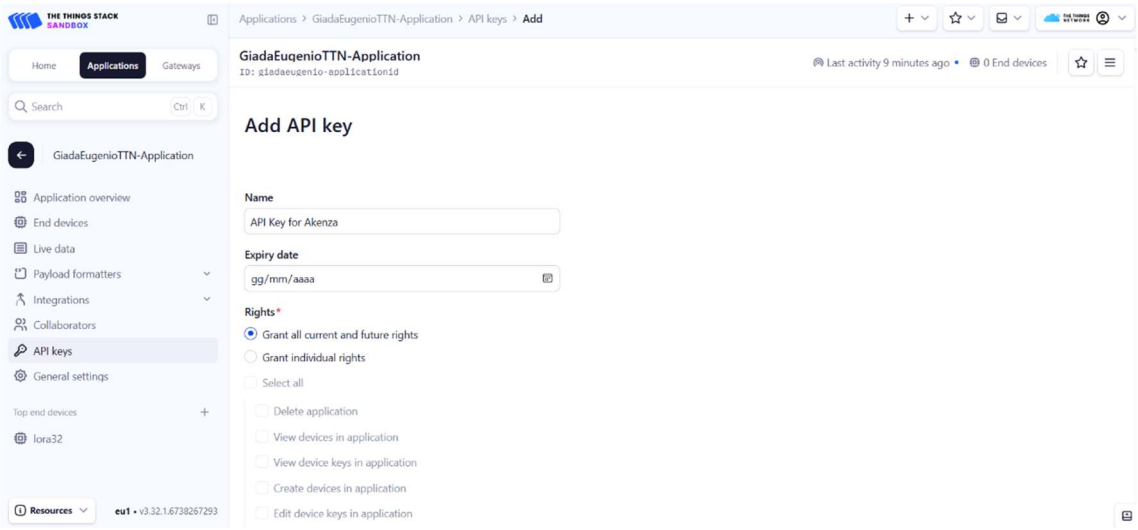

Figure 138 – TTN, creating an API key (part 1).

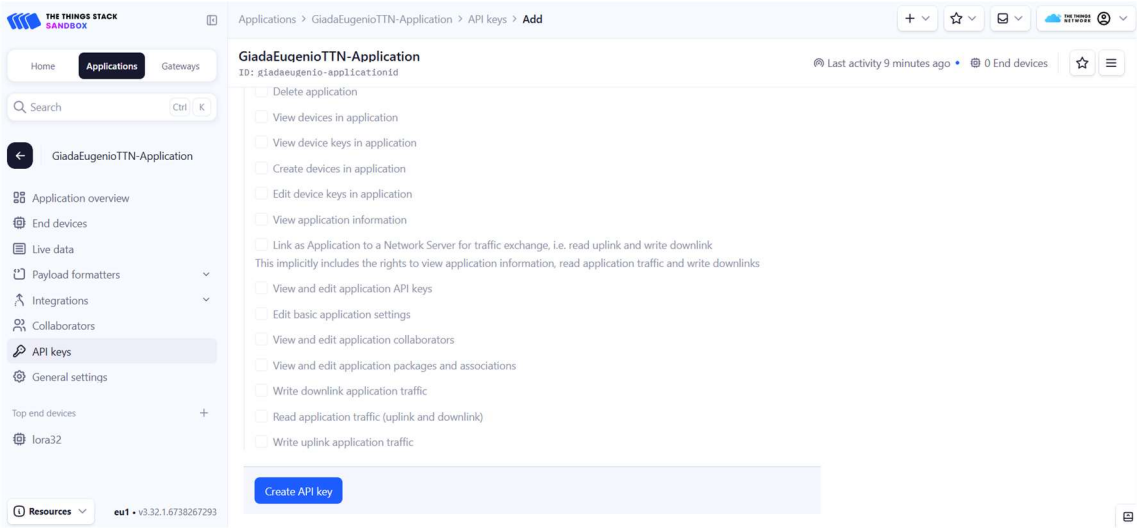

Figure 139 – TTN, creating an API key (part 2).

On the Akenza website Integrations → +Create Integration → TTN LoRaWAN was selected (Figure 140).

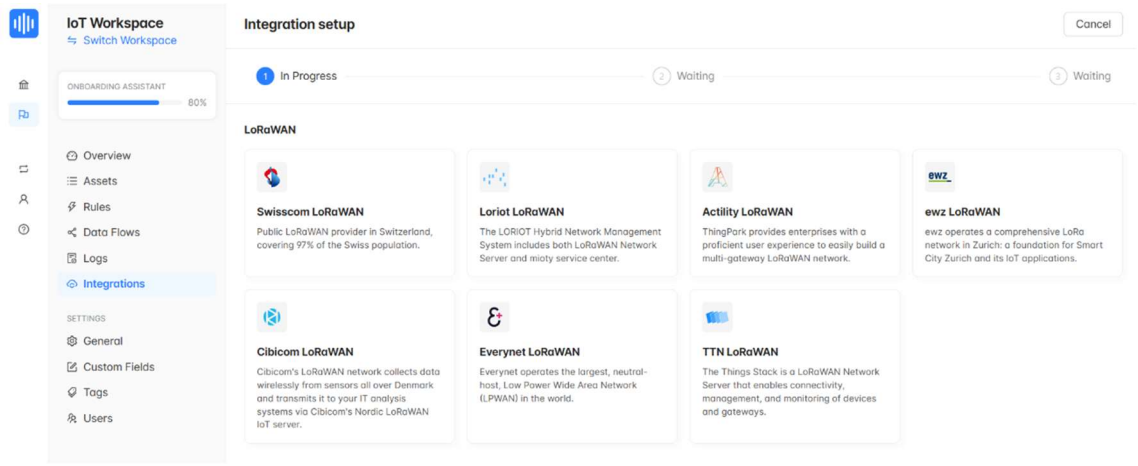

Figure 140 – Akenza, creating the Integration with TTN.

At this point the Application ID and the API key that were obtained from the TTN website, were inserted in Akenza (Figure 141).

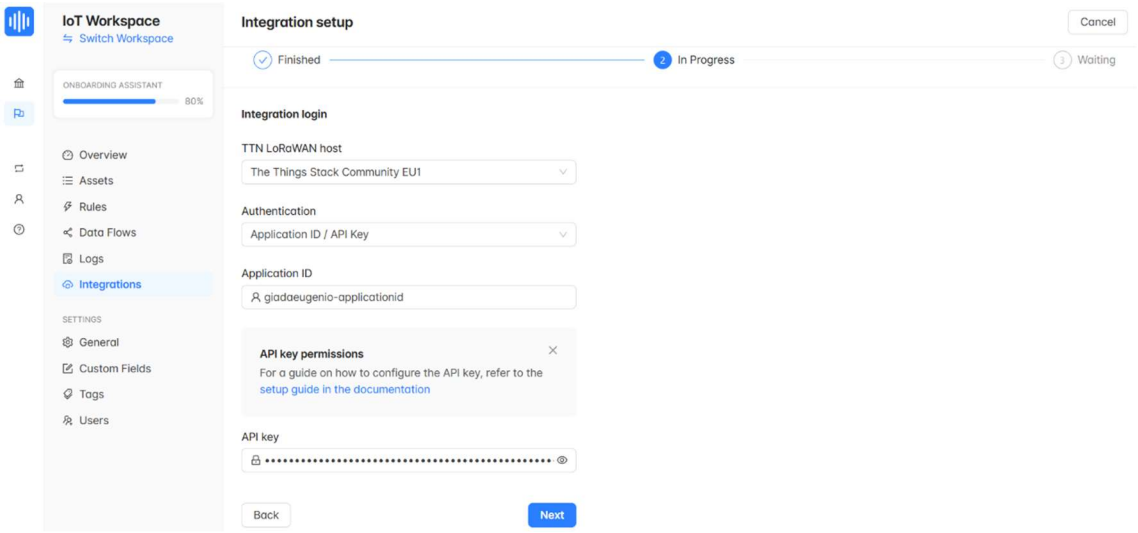

Figure 141 – Akenza, inserting the details from TTN.

Finally, an integration name was inserted, and the option Import Devices was automatically checked. This option allows Akenza to import the LoRa32 device that was created on the specific TTN Application (Figure 142).

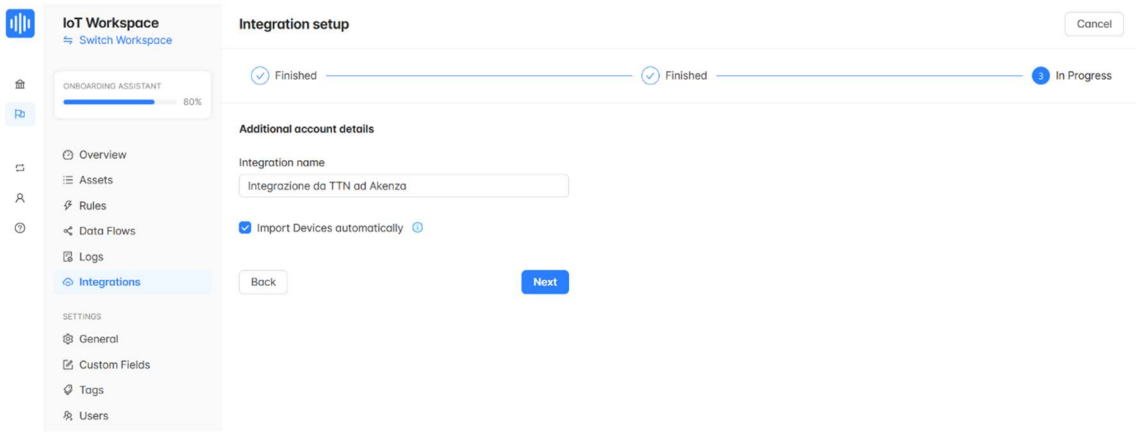

Figure 142 – Akenza, naming the Integration with TTN.

At this point, the integration appeared on both the Akenza (Figure 143) and the TTN (Figure 144) websites.

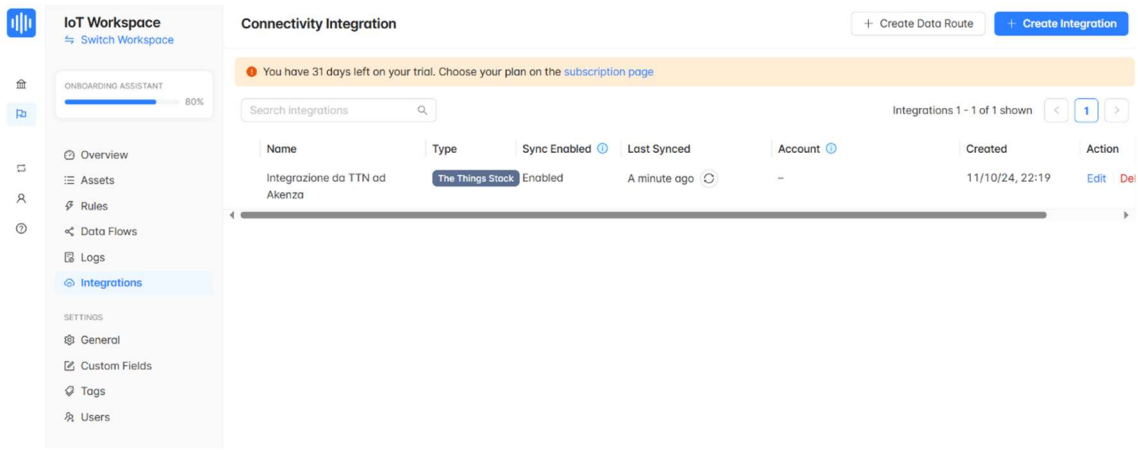

Figure 143 – Akenza, the created Integration.

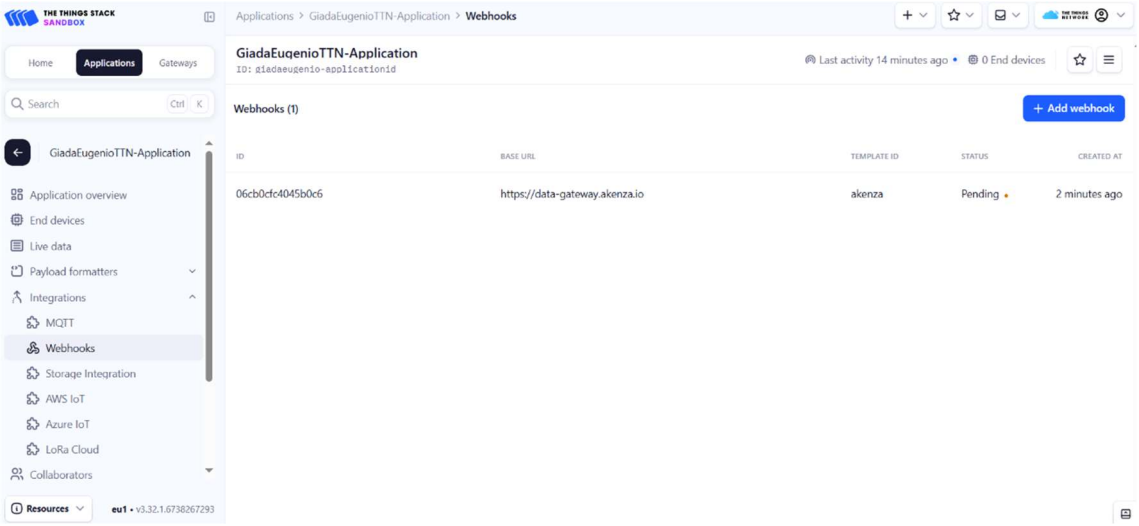

Figure 144 – TTN, the resulting Integration.

In Akenza, in Assets, the LoRa32 device appeared (Figure 145).

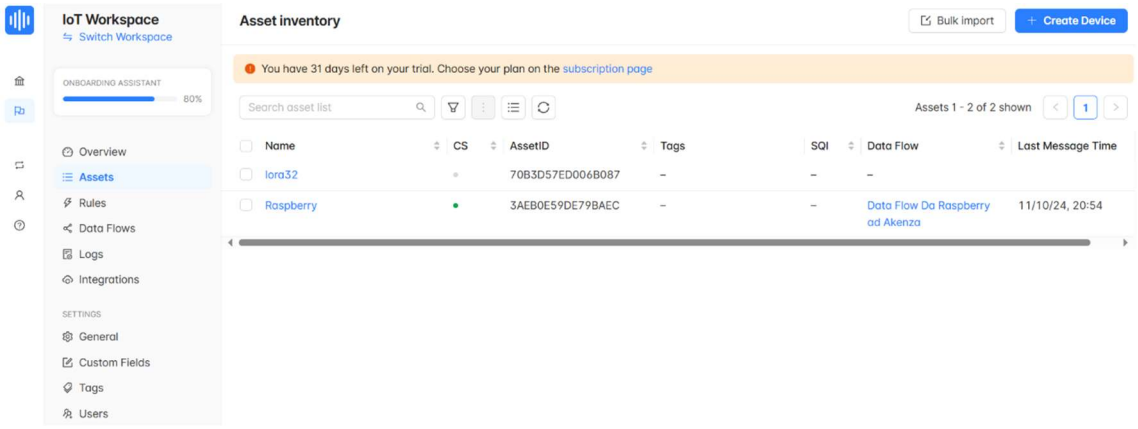

Figure 145 – Akenza, Assets.

The next step was to create the LoRaWAN Data Flow, which will be assigned to the LoRa32 device. "+Create Data Flow" was selected (Figure 146).

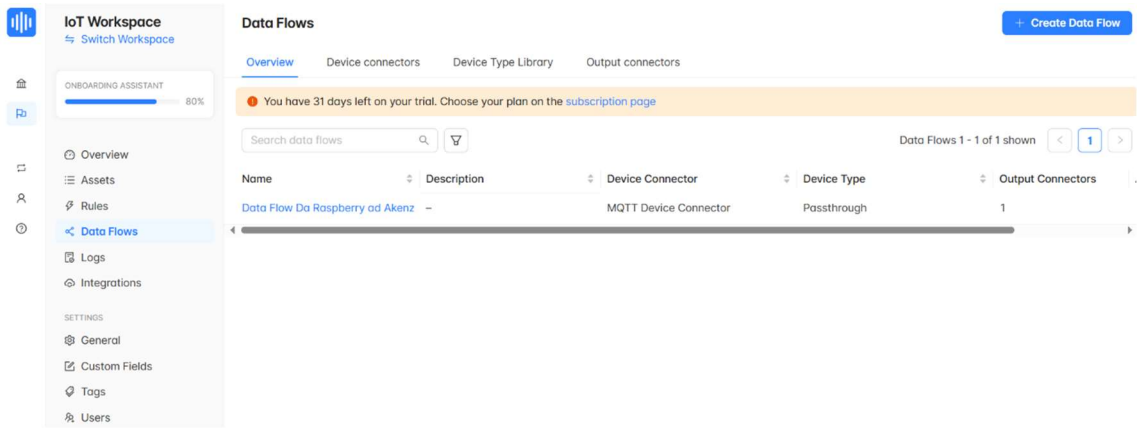

Figure 146 – Akenza, list of Data Flows.

In this second Data Flow, instead of using a template, it was necessary to create it manually, by choosing "+Create new Data flow" (Figure 147).

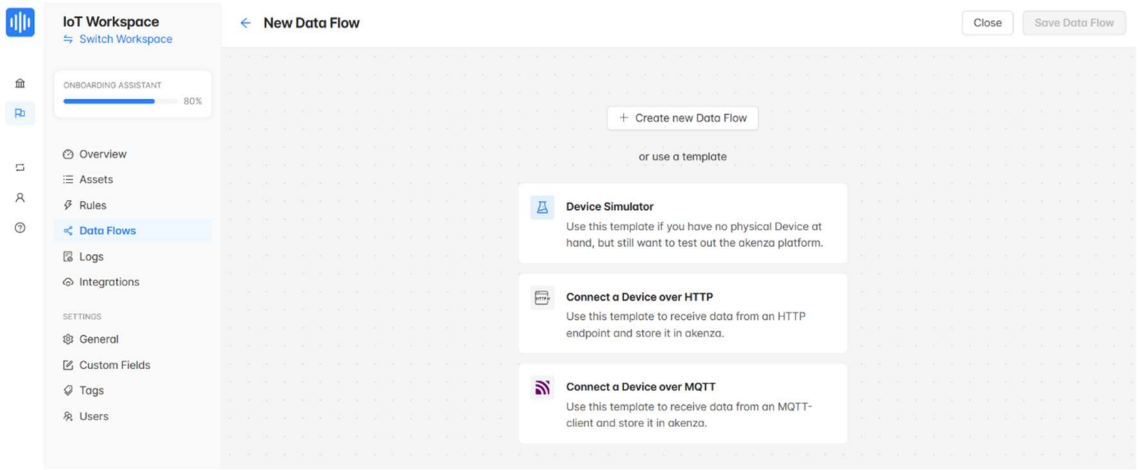

Figure 147 – Akenza, creating the LoRaWAN Data Flow.

In this second Data Flow, LoRa → The Things Stack → the created Integration with TTN was chosen as Device Connector (Figure 148).

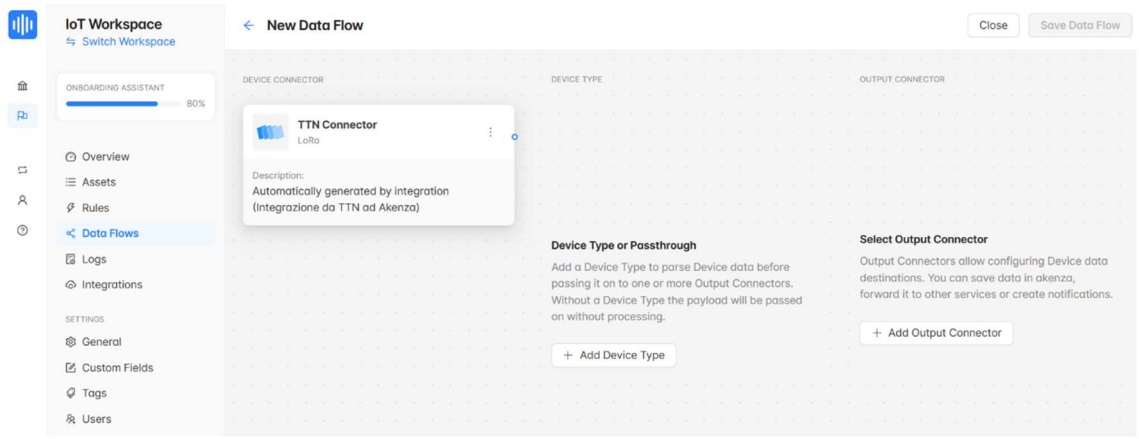

Figure 148 – Akenza, choosing the Device Connector.

Thereafter, Cayenne LPP was selected as Device Type (Figure 149).

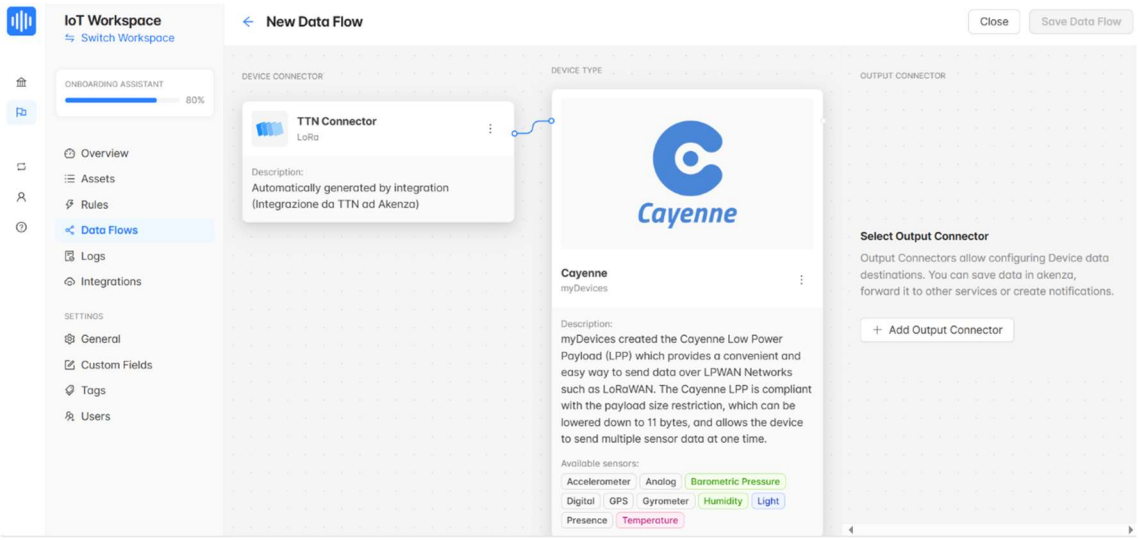

Figure 149 – Akenza, choosing the Device Type.

Akenza Database was chosen as Output Connector (Figure 150).

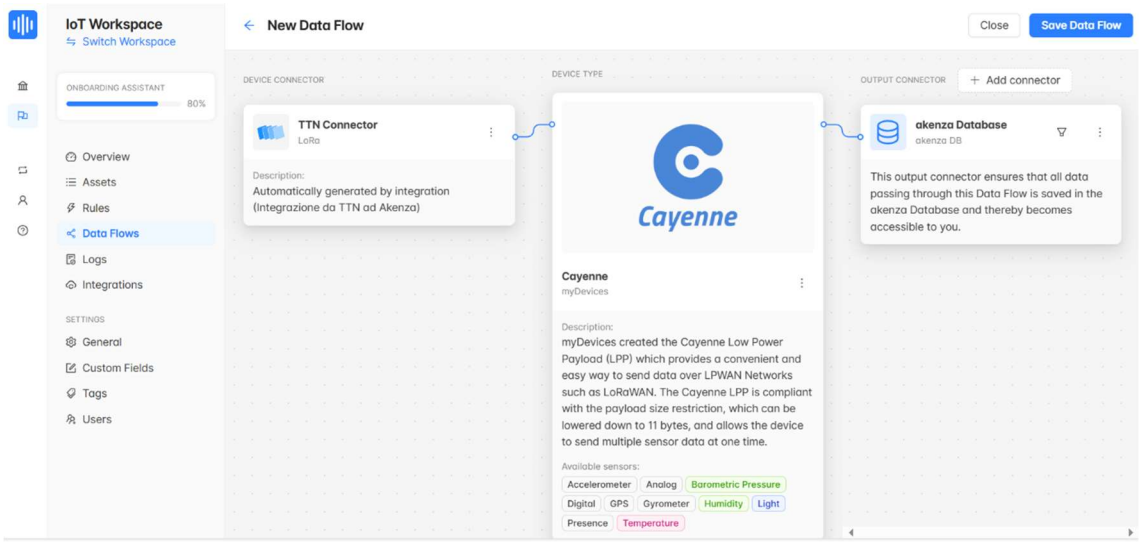

Figure 150 – Akenza, choosing the Output Connector.

For this new Data Flow, the data source is the integration with TTN, the data coming from TTN is in the Cayenne LPP format and has to be processed as such and the data is, then, stored in Akenza's database.

The subsequent step was to associate the LoRaWAN Data Flow created to the LoRa32 device (Figure 151), so in Assets → LoRa32 → Edit Device the LoRaWAN Data Flow was assigned to it (Figure 152).

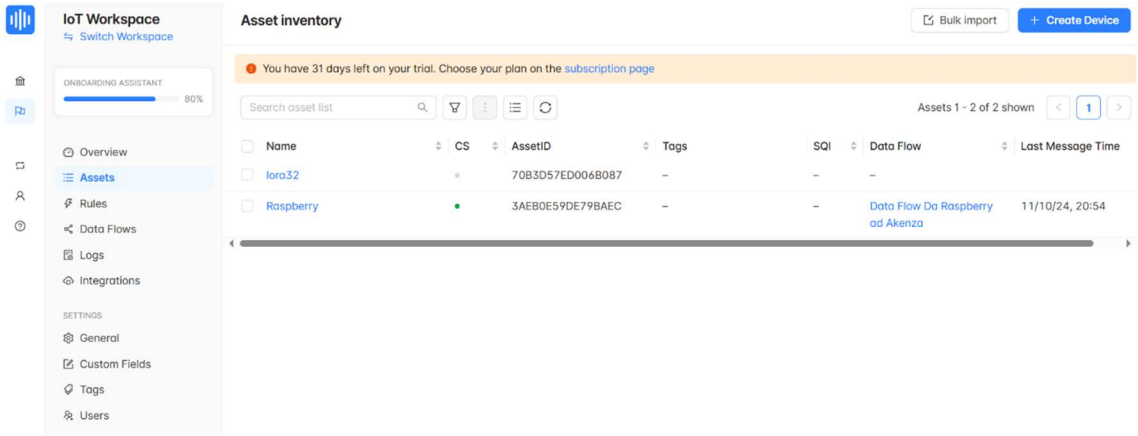

Figure 151 – Akenza, updated list of Devices in Assets.

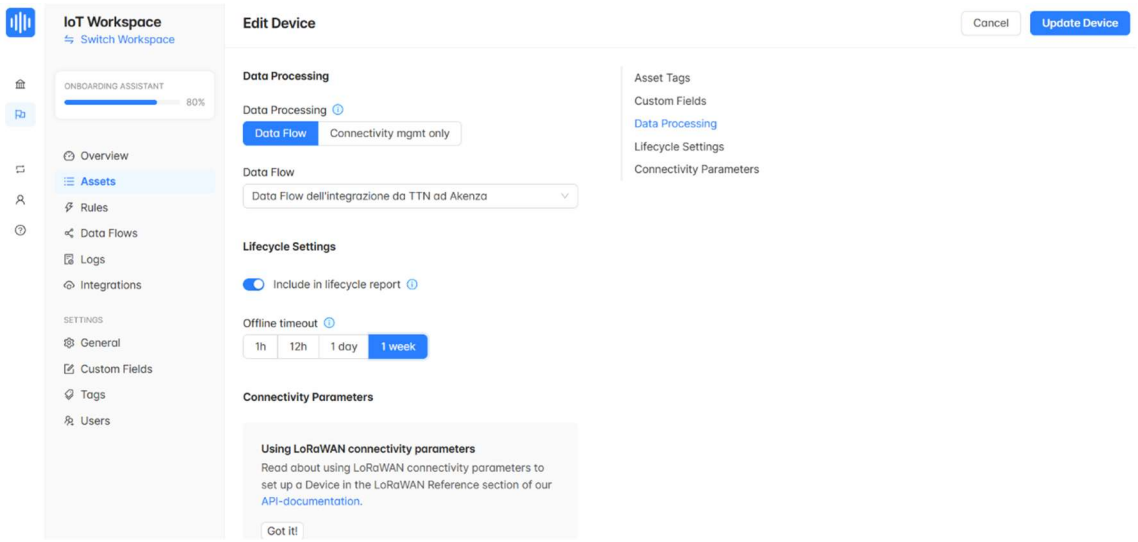

Figure 152 – Akenza, assigning the LoRaWAN Data Flow to the LoRa32 device.

7.7 Akenza Rules

This section describes the procedure to create an Akenza Rule.

To create a Rule, it is necessary to define the input, the logic and the action (Figure 153).

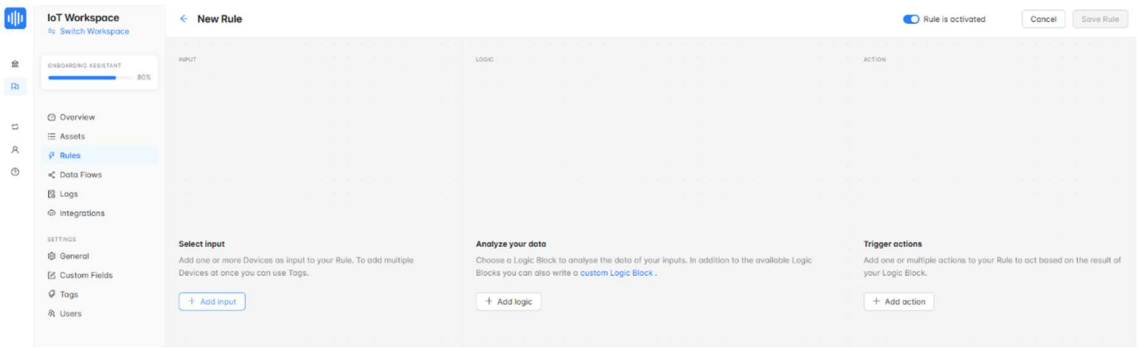

Figure 153 – Akenza, New Rule.

Device→Raspberry was chosen as the input (Figure 154).

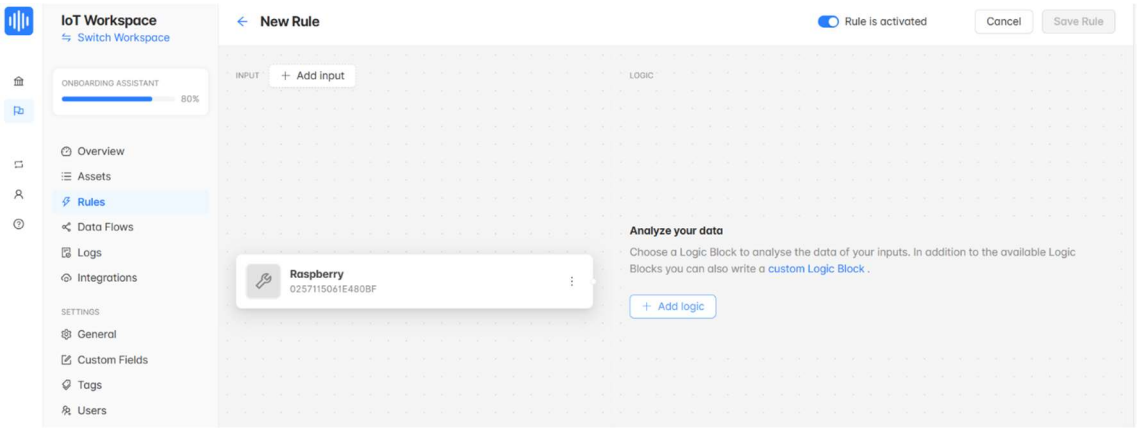

Figure 154 – Akenza, Input Block of the Rule.

Then, the Comparison was chosen as logic and was connected to the Input Block. Figure 155 shows the details of the condition: if the value of the temperature that Akenza receives from the Raspberry is greater than 37.5 °C, then the action is triggered.

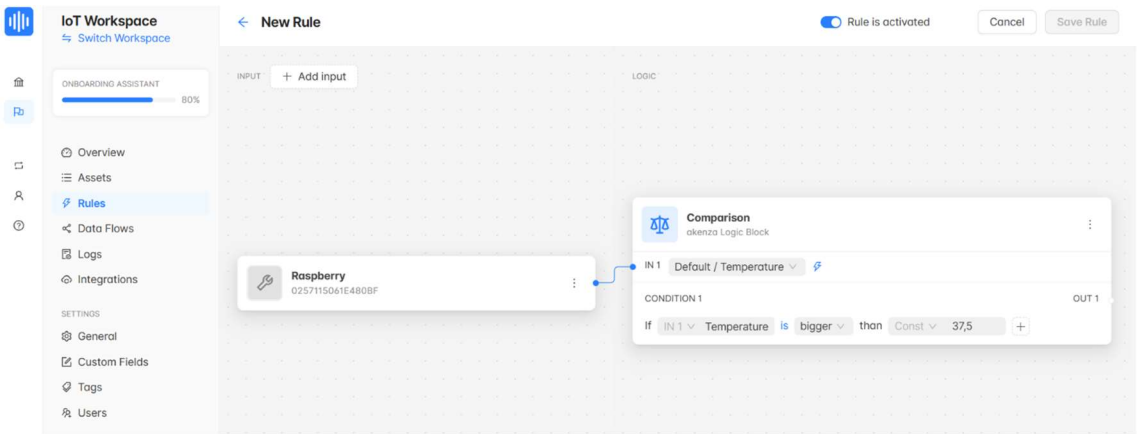

Figure 155 – Akenza, Logic Block of the Rule.

Finally, the Comparison Block and the Action Block were connected, and it was chosen that if the condition is met, then Akenza will send a warning email, whose text and Recipients were inserted (Figure 156).

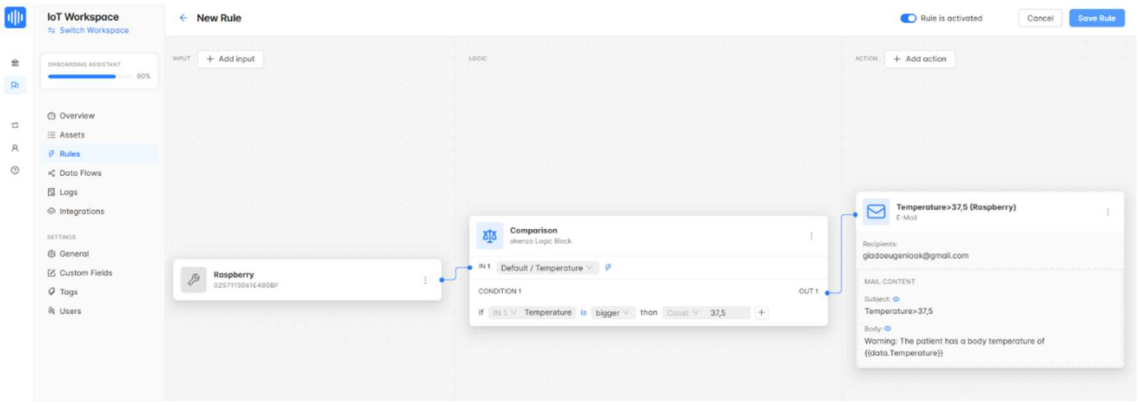

Figure 156 – Akenza, Action Block of the Rule.

After saving the Rule, it appeared in the list of Akenza Rules (Figure 157).

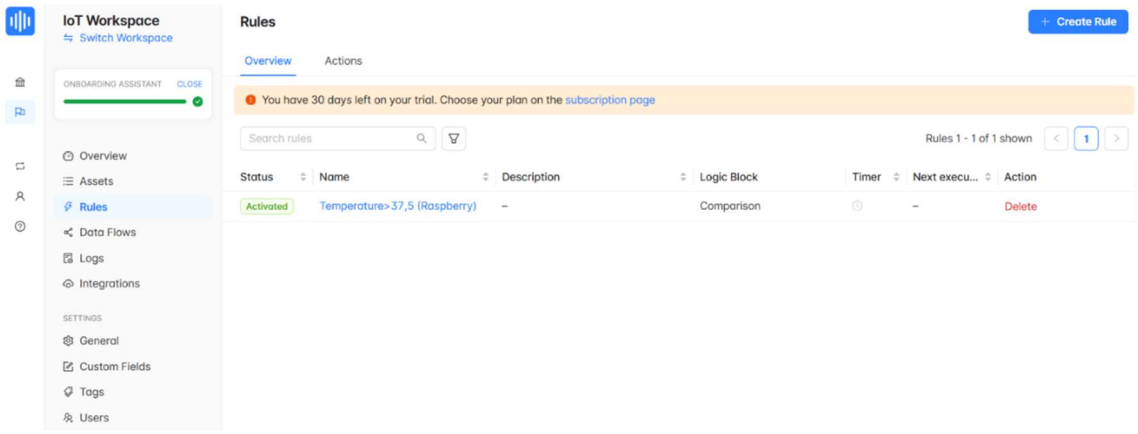

Figure 157 – Akenza, list of the Rules.

Figure 158 shows an example of the email sent by Akenza after the condition was met.

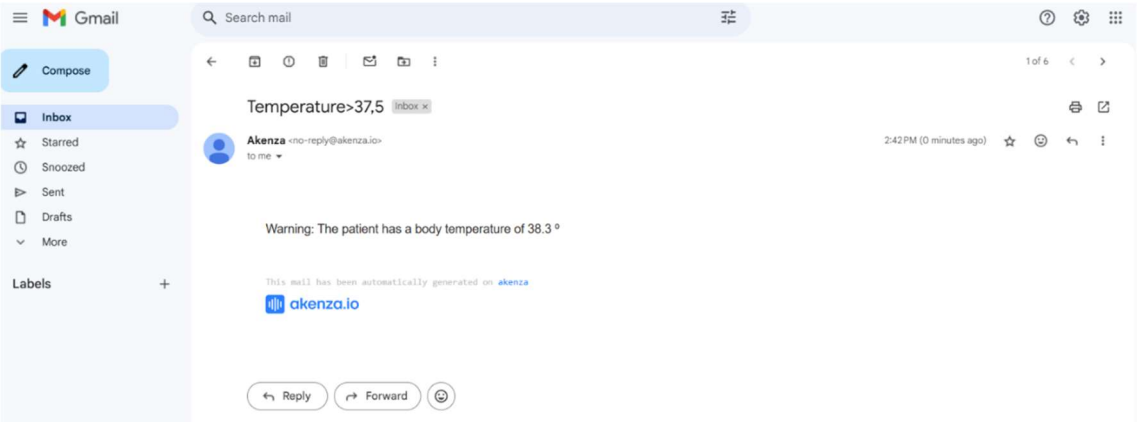

Figure 158 – Akenza's email after the condition is met.

7.8 Description of the Software running on the Arduino

The task of the Arduino is to read from the MAX30100 (through the I^2C communication), the Heart Rate and SpO2 values and to send them to the Raspberry. Figure 159 is the Flowchart of the software that runs on the Arduino.

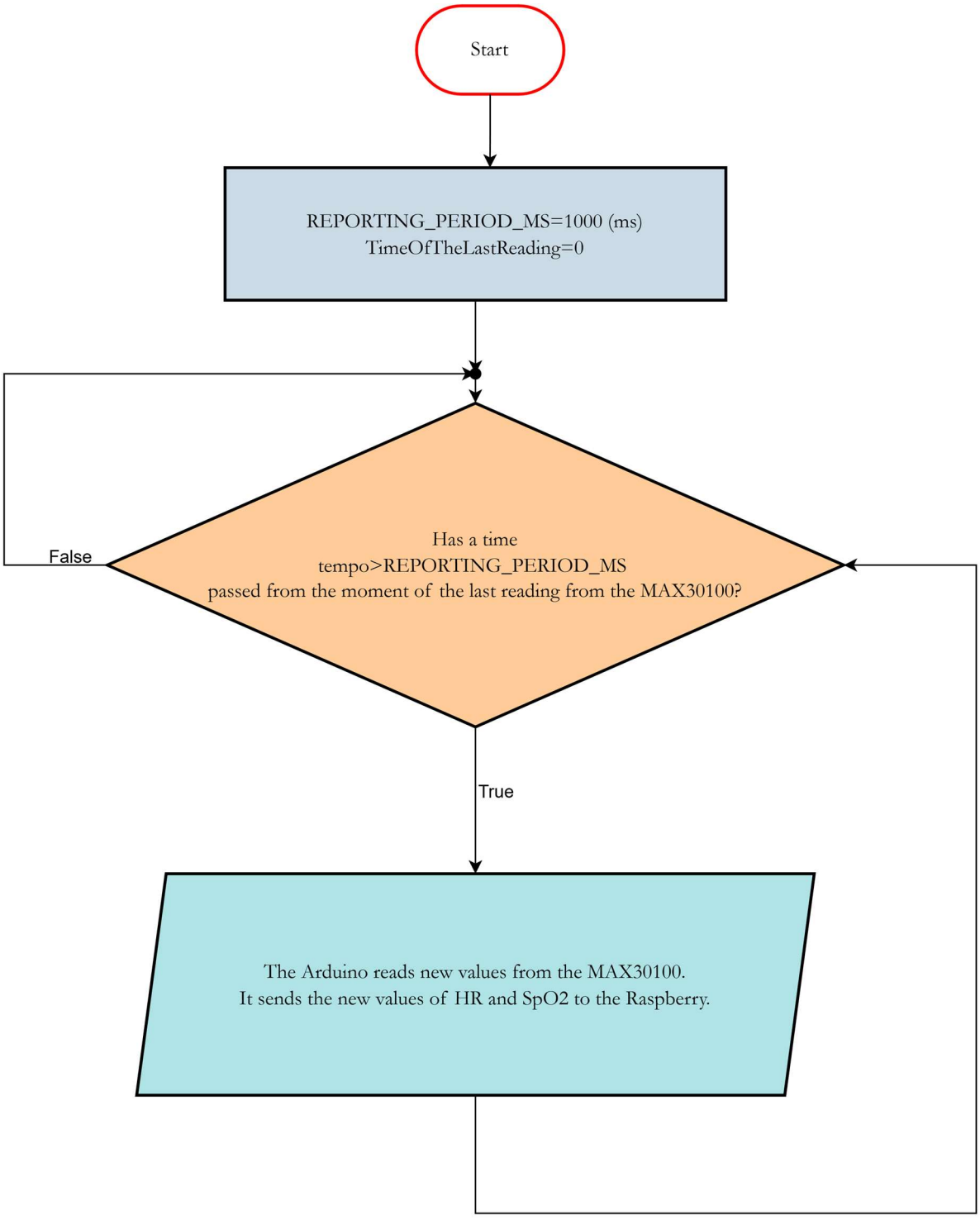

Figure 159 – Flowchart of the Software that runs on the Arduino.

The Arduino takes a new reading from the MAX30100 and sends the new HR and SpO2 values to the Raspberry whenever the time since last reading is greater than *REPORTING_PERIOD_MS*. The Arduino, then, sends the new values of HR and SpO2 to the Raspberry. The sending of the data takes place through the USB connection and the Arduino sends a string in the form: (HR value)**S**(SpO2 value)**F**.

The letter "S" is necessary so that the Raspberry can understand where the HR value ends, while the letter "F" is needed in order to let the Raspberry understand where one entire reading that it received ends. By proceeding this way if, for some reasons, the Raspberry is too slow in reading the strings that the Arduino sends, then all of these accumulate in the receive buffer of the Raspberry, which thanks to the letter "F" is able to understand where the first string ends.

7.9 Description of the Software running on the Raspberry

Two different software modules run on the Raspberry. One is able to measure the desired parameters of the patient (temperature, HR and SpO2), to save them in the local database and to subsequently send them to Akenza. Moreover, this software module manages the messages that are shown to the patient on the display. The software module also manages the dual-purpose button. The other software module on the Raspberry, instead, is able to continuously check, through the use of the accelerometer and gyroscope, if the patient is falling. This monitoring has to run uninterruptedly, because it is possible that, in the event of a program break/stop, the elderly person falls and the system does not detect this fall. This implies that the second software module cannot be incorporated into the first one and has to be independent from it.

7.9.1 Software Module for the Data Collection

In the main() of this software module, the Raspberry opens the local database and, if it does not already exist, the Raspberry creates it. Then the Raspberry creates a table named *ValuesTable* with the columns ID, Date, Time, Temperature, HR and SpO2, if it does not already exist. The Id is the primary key and it increments automatically. Then there are the parameters of the circuit (R1 and E), of the ADC (VREF) and of the NTC (beta, T0 and Rntc0). Then, the serial receive buffer of the Raspberry is cleaned from possible waste data. Subsequently, the software indicates which are the topic, the MQTT username and the MQTT password of the MQTT communication in which the Raspberry is the publisher and Akenza is the subscriber. This information is taken from Akenza, after registering the Raspberry end-device.

At this point, the pin16 of the Raspberry was chosen as an input pin and it was configured in the software, in order to connect it through an internal pull-up resistor to an internal voltage of 3.3 V. In the architecture, the button is between the pin16

and ground and in the software it was established that the event of the button consisting in the pin16 changing from high to low would cause an interrupt with the intervention of the callback function *button_pressed_callback*. More specifically, the *bouncetime* was set so that, if within 1 second the pin16 changes from high to low more than once, the event would be considered as if it only happened one time.

At this point there is an infinite loop and the global variables *isBottoneDiOK* and *ilBottoneIsPressed* are fundamental. The variable *isBottoneDiOK* indicates if the button is an emergency button (when *isBottoneDiOK*=0) or an OK button (when i*sBottoneDiOK*=1). At the beginning and normally, this variable has to be 0 and, only when necessary and for a very short time, it has to be 1.

The variable *ilBottoneIsPressed* indicates if the button has been pressed (independently from it being an emergency button or an OK button). At the beginning this variable is 0 and becomes 1 only after the button has been pressed.

The entire device, between the 8h:00m and the 20h:00m and with intervals of 1 hour, has to show on its display the message "Are you ok? Press the confirmation button".

The Raspberry sees if the current time meets these conditions:

- If these conditions are False, the button is an emergency button and *isBottonediOK*=0. As soon as the button is pressed, it generates an interrupt and the callback function is called. In this callback function the variable *ilBottoneisPressed* becomes 1, because the button has just been pressed. At this point, since *isBottonediOK*=0, then the Raspberry tries to send the request for emergency via Wi-Fi and Internet and, if there is no Wi-Fi or Internet connection, the emergency is sent to the LoRa32 via USB.
- On the contrary, if the conditions are True, the Raspberry sends the message "MAre you ok? Press the confirmation button"* to the LoRa32, so that the LoRa32 can show it on its display, and the variable *isBottonediOK* becomes 1. At this point a timer of 0.5 minutes starts. While this timer has not finished and the variable *ilBottoneIsPressed*=0, the Raspberry waits. As soon as the patient presses the button and *ilBottoneIsPressed* becomes 1, it means that the patient is fine, so *isBottonediOK* and *ilBottoneIsPressed* return to 0. On the contrary, if the button is not pressed within that time, then the Raspberry tries to send the request for emergency via Wi-Fi and Internet and, if there is no Wi-Fi or Internet connection, the emergency is sent to the LoRa32 via USB.

Then, the Raspberry takes the value of V_{ntc} from the ADC and uses it to calculate R_{ntc}, then the body temperature expressed in Kelvin and, finally, the temperature is converted into a body temperature value on the Celsius scale.

Then, the Raspberry waits until in its receive buffer there is no data from the Arduino via USB. As soon as there is data from the Arduino, the Raspberry reads this string.

This string is in the form (HR value)**S**(SpO2 value)**F**. The Raspberry uses the position of the letter "F" to take only one string of values from the Arduino and the position of the letter "S" to distinguish between the HR value and the SpO2 value. The Raspberry obtains the substrings *bpmStringa* and *SpO2Stringa*.

Next, it converts the HR substring to *float* and the SpO2 substring to *int*. Then, it rounds the value of the temperature to 1 decimal digit and the HR to 2 decimal digits.

At this point, the Raspberry obtains the current date and time and inserts the values of *date*, *time*, *temperaturaRound*, *bpmRound* and *SpO2* into the table in the local database.

Then, *temperaturaRound* and *bpmRound* are converted to strings and the Raspberry creates the payload that has to be sent to Akenza via Wi-Fi.

The Raspberry tries to send the payload to Akenza via Wi-Fi and the Internet. If it fails, the Raspberry creates the string *StringaCompleta*, which is in the form **T**(temperature value)**B**(HR value)**S**(SpO2 value)* and sends it to the LoRa32.

Figure 160 illustrates the Flowchart of the Raspberry software module for sensors data collection described above.

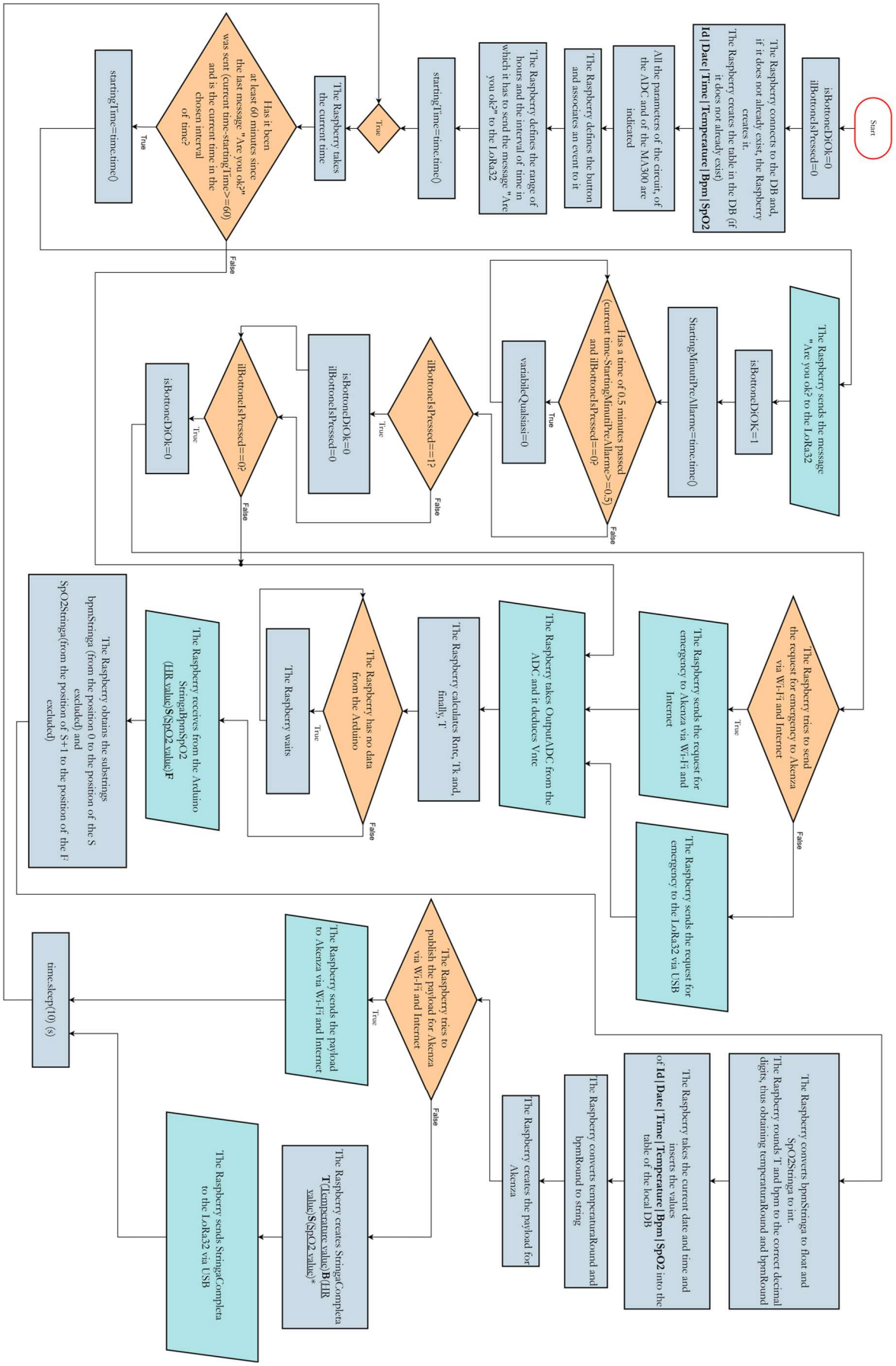

Figure 160 – Flowchart of the Raspberry Software Module for Sensors Data Collection.

Figure 161 illustrates the Flowchart of the *button_pressed_callback()* function, which is the callback function present in the Raspberry software module of sensors data collection, with the task of managing the dual-purpose button.

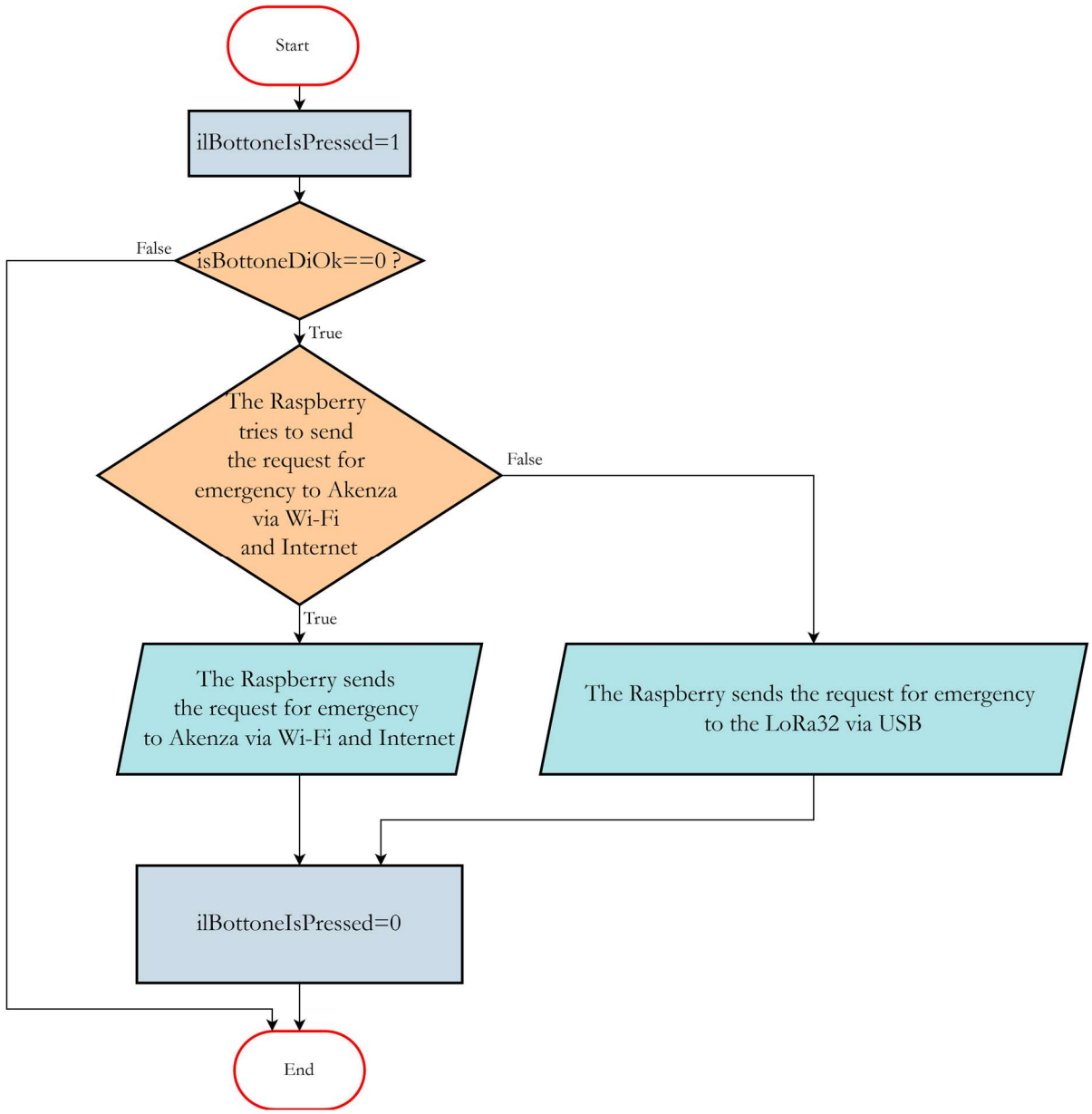

Figure 161 – Flowchart of the *button_pressed_callback()* function of the Raspberry Sensors Data Collection Software Module.

7.9.2 Software Module for the Fall Detection

The fall detection software module uses the MPU-6050 sensor, which must be placed on the patient's torso, for example in a belt bag [87].

Figure 162 represents the behaviour in a fall of the modulo of the acceleration versus time.

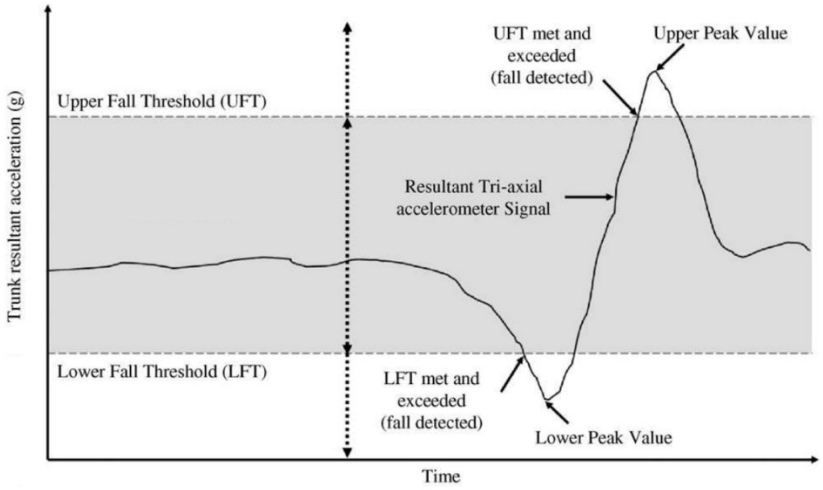

Figure 162 – Modulo of the Acceleration as a function of Time during a Fall [87].

When the patient is stationary the modulo of the acceleration is equal to +1g.

During the fall, the modulo of the acceleration decreases until the impact, when it reaches its lowest value. Due to the impact with the ground, the modulo of the acceleration increases until it reaches its highest value.

Experimentally, it was found that when a fall occurs, firstly the module of the acceleration decreases below a certain Lower Fall Threshold (LFT). Subsequently to the impact, the value of the modulo of the acceleration suddenly exceeds a certain Upper Fall Threshold (UFT).

Figure 163 shows experimental data published in [87]. representing the modulo of the acceleration: in a typical fall (A); in the fall that produced the smallest lower peak value (B); in the fall that produced the smallest upper peak value (C); in a typical sitting on an armchair activity (D); in a getting in and out of a car seat activity (E); and in a walking activity (F).

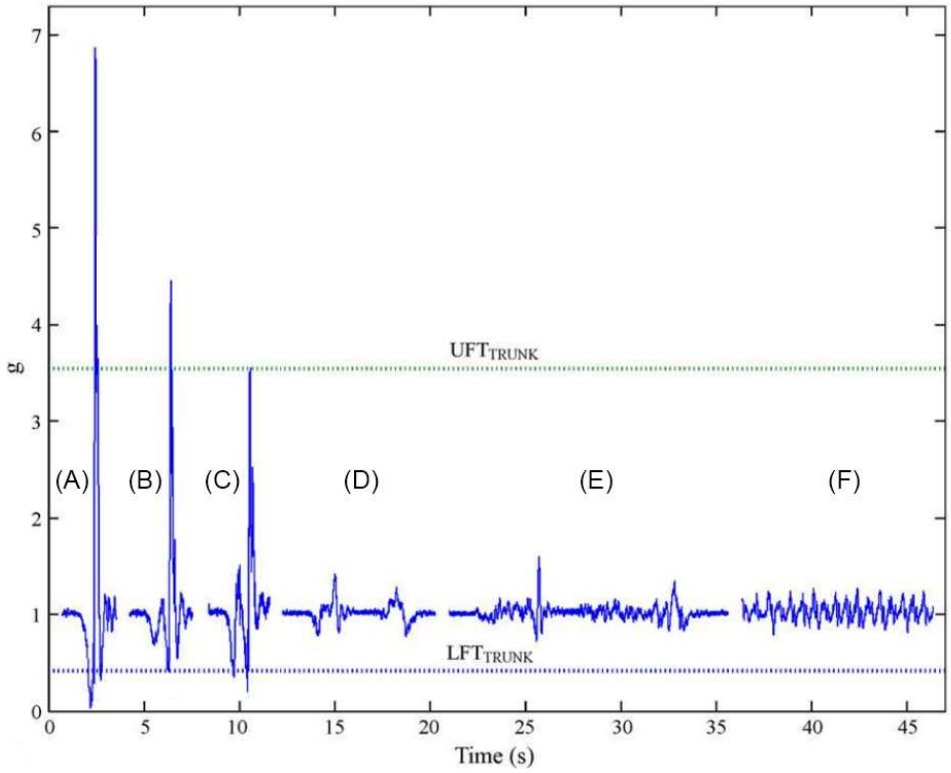

Figure 163 – Behaviour of the Modulo of the Acceleration in different situations.

From the analysis of these signals, it was experimentally found the value of the Lower Fall Threshold, which was set at the level of the smallest magnitude lower fall peak and the value of the Upper Fall Threshold, which was set at the level of the smallest magnitude upper fall peak. The LFT and the UFT are indicated in Table 31.

Table 31 – LFT and UFT threshold values.

Threshold values	Acceleration
LFT	0.41 g
UFT	3.52 g

For the fall detection to be more accurate, it is necessary to calculate the modulo of the angular velocity that the patient had at the moment when the Upper Fall Threshold was surpassed. If this value is between 30 °/s and 400 °/s, it means that the torso of the patient had rotated, thus implying a fall. It is possible that this condition did not occur immediately, and the patient was still upright before rotating. In this situation it is necessary to calculate the new modulo of the angular velocity for the next 50·0.01 s =500 ms, in order to check if the patient is rotating and falling. Lastly, in a fall, the modulo of the angular velocity becomes between 0 °/s and 10 °/s, because at this point the patient lies on the ground.

The variable *trigger1* indicates whether the first condition is happening, which is to say that the modulo of the acceleration is lower than the LFT. In the same way, *trigger2* indicates if the modulo of the acceleration exceeds the UFT and *trigger3* if the modulo of the angular velocity is between 30 °/s and 400 °/s.

If then, after 90·0.01 s= 900 ms the modulo of the angular velocity is between 0 °/s and 10 °/s, then the variable *fall* becomes True.

At the beginning, *trigger1*, *trigger2*, *trigger3* and *fall* are False. The variable *trigger1count* indicates how many times *trigger1* is equal to True. The same happens with the variables *trigger2count* and *trigger3count*. At the beginning *trigger1count*, *trigger2count* and *trigger3count* are equal to 0.

The variable *trigger1count* returns equal to 0 as soon as *trigger1* becomes False.

The same happens for *trigger2count* and *trigger3count*. If the fall occurs, *fall* becomes True and, as soon as the information of emergency is sent to Akenza, the variable becomes False.

If *trigger2count* is greater than 50, it means that a time of 50·0.01 s=500 ms has passed since *trigger2* had become True and only the first and the second conditions have occurred, then the patient has not fallen and *trigger2* becomes False and *trigger2count* returns to 0. In the same way, if *trigger1count* is greater than 50, it means that a time of 50·0.01 s=500 ms has passed since *trigger1* had become True and only the first condition has occurred, then the patient has not fallen and *trigger1* becomes False and *trigger1count* returns equal to 0.

Figure 164 illustrates the Flowchart of the Raspberry software module for the fall detection.

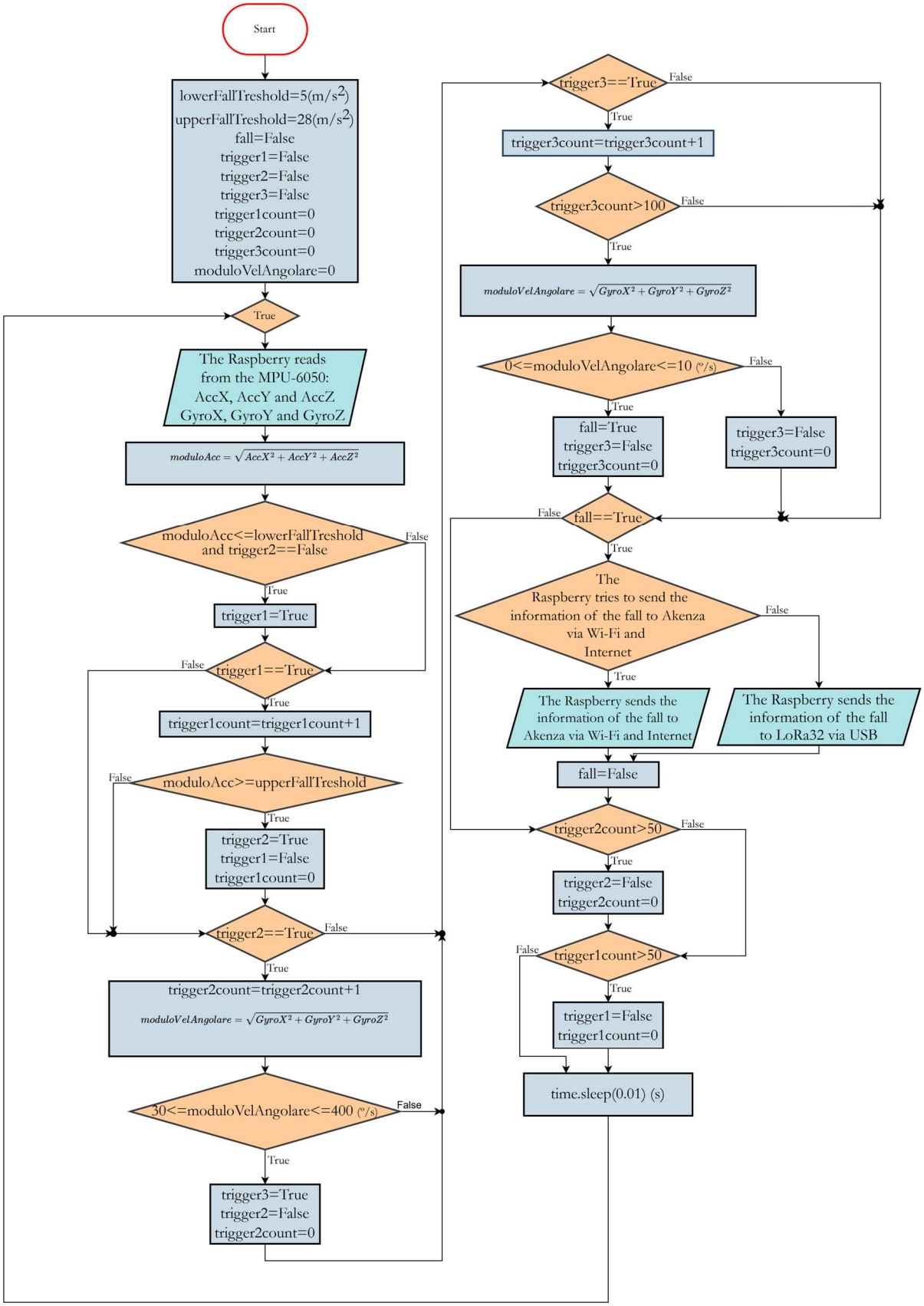

Figure 164 – Flowchart of the Raspberry Software Module for Fall Detection.

7.10 Description of the Software running on the LoRa32

The software in the LoRa32 is composed of the setup and the loop.

At the first power up of the LoRa32, the software starts with the setup, which calls for the first time the function *do_send()*. Figure 165 shows the Flowchart of this function.

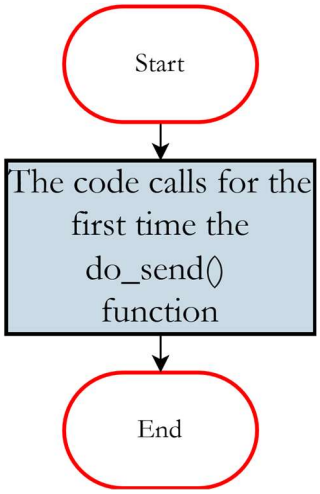

Figure 165 – Flowchart of the *setup()* function of the LoRa32 Software.

In the function *do_send()*, the LoRa32 waits for data from the Raspberry, analyses it and prepares the actual data for the transmission to TTN.

Subsequently, the software continues with the loop, whose function *os_runloop_once()* manages all the events (Figure 166).

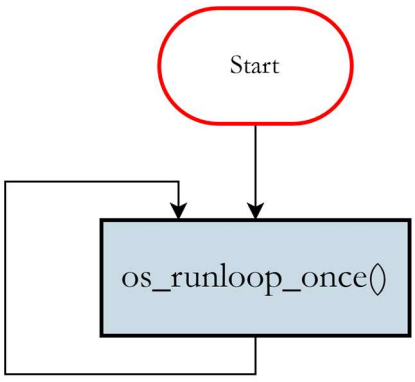

Figure 166 – Flowchart of the *loop()* function of the LoRa32 Software.

Since it is the first time that the LoRa32 has to send data to TTN, then *os_runloop_once()* calls the joining event (EV_JOINING) with the aim of creating the connection between the end-device and the Join Server of TTN. As soon as the

joining is successful, the algorithm continues with the loop, whose function *os_runloop_once*() calls the joined event (EV_JOINED). Subsequently, the transmission event (EV_TXSTART) effectively sends the prepared data to TTN. Thereafter, the next event transmission completed (EV_TXCOMPLETE) calls the *do_send()* function, in order to wait for new data and to prepare it for the next transmission. It is important to observe that the events *joining* and *joined* only happen in the case of the first transmission to TTN.

From this moment onward, the algorithm will proceed in this order:

- In the function *do_send()*, the LoRa32 waits for data from the Raspberry, analyses it and prepares the actual data for the transmission to TTN.
- The data transmission event sends the prepared data to TTN.
- The data transmission complete event calls the *do_send()* function.

These steps will continue to happen in this succession.

The function do_send()

At the beginning of the function *do_send()* the variable *devoUscireDalMessaggio*=1.

This means that the code can enter in the while. Immediately after entering, this variable becomes equal to 0. At this point, the LoRa32 shows on its display "I'm waiting for data from Raspberry" and waits until there is data. As soon as this happens, the LoRa32 creates the string *PREinString* and reads the received data one char at a time and adds them to *PREinString*. This means that the obtained *PREinString* contains all the data received from the Raspberry. Then the LoRa32 shows this data on its display. Subsequently, if the string *PREinString* contains the substring "C*", then it means that the data received from the Raspberry contains the information of a fall of the patient and this kind of data has the priority over every other kind of received data. In this situation *mydataCaduta* is sent to TTN.

The array *mydataCaduta* is made of 4 bytes:

- *mydataCaduta[0]* contains the number of the channel of the Cayenne LPP payload. This number, however, is not used because Akenza does not use the Cayenne LPP channel.
- *mydataCaduta[1]* contains the type of data in the Cayenne LPP payload.
- *mydataCaduta[2]* contains the MSBs, the first 8 bits of *intFallDetected*. This variable is an integer and, since an *int* is made of 2 bytes, only the most significant byte is put into *mydataCaduta[2]*.
- *mydataCaduta[3]* contains the LSBs, the last 8 bits of *intFallDetected*.

If, instead, the string *PREinString* contains the substring "E*", then it means that the data received from the Raspberry contains the information of an emergency, so *myDataEmergenza* has to be sent to TTN.

The array *myDataEmergenza* is made of 3 bytes:

- *myDataEmergenza[0]* contains the number of the channel of the Cayenne LPP payload. This number, however, is not used because Akenza does not use the Cayenne LPP channel.
- *myDataEmergenza[1]* contains the type of data in the Cayenne LPP payload.
- *myDataEmergenza[2]* contains the LSBs, the last 8 bits of *intEmergencyDetected*. In this particular case the data type of the Cayenne LPP payload is only 1 byte. This means that, since the variable *intEmergencyDetected* is an integer, only the less significant byte is put into *mydataEmergenza[2*]. This is the reason why *myDataEmergenza* is composed of only 3 cells.

Else, if the string *PREinString* contains "M", then it means that the data received from the Raspberry contains a message that the LoRa32 has to simply show on its display, without sending anything to TTN. In this case, *PREinString* is in the form Mmessage*. At this point, the LoRa32 identifies the position of the letter "M" and uses this position to make the string *PREinString* start from it. This means that now the letter "M" is in position 0.

Now, the LoRa32 identifies the position of the first asterisk "*" from this position 0 and uses this information to take the substring *inString*, which contains the message that the LoRa32 has to show on its display.

If, instead, the string *PREinString* contains "T", then it means that the data received from the Raspberry contains the collected data that the LoRa32 has to send to TTN. In this case *PREinString* is in the form **T**(Temperature value)**B**(HR value)**S**(SpO2 value)*. At this point, the LoRa32 identifies the position of the letter "T" and uses this position to make the string *PREinString* start from it. This means that now the letter "T" is in position 0.

Now, the LoRa32 identifies the position of the first asterisk "*" from this position 0 and uses this information to take the substring *inString*, which contains all the sensor values.

In this case, using the positions of the letters "B" and "S", the LoRa32 distinguishes the temperature substring, the HR substring and the SpO2 substring, respectively and converts the Temperature string and the HR string to *float* and the SpO2 string to *int*. Subsequently, the LoRa32 makes the correct multiplications on them, according to their data type of Cayenne LPP payload and prepares *mydata*, which will be later sent to TTN.

More specifically, *mydata* is an array of bytes that has 12 cells. The array *mydata* is made of 12 bytes:

- *mydata[0]* contains the number of the channel of the Cayenne LPP payload. This number, however, is not used because Akenza does not use the Cayenne LPP channel.
- *mydata[1]* contains the type of data in the Cayenne LPP payload.
- *mydata[2]* contains the MSBs, the first 8 bits of *intTemperatura*. This variable is an integer and, since an *int* is made of 2 bytes, only the MSBs are put into *mydata[2]*.
- *mydata[3]* contains the LSBs, the last 8 bits of *intTemperatura*.

The remaining cells also follow this pattern, with four cells dedicated to the HR and the last four cells to the SpO2.

It is necessary to notice the importance of the first *while*. In fact, without that *while*, if the data received from the Raspberry is a message that the LoRa32 has to show on its display, after doing this the function *do_send()* would finish and the code would return in the loop. However, if the LoRa32 has no data to send to TTN, the software would be stuck. In order to avoid this situation, this *while* is fundamental. Another important aspect is that this *while* has to be True only at the start of the function and after the situation in which the data received from the Raspberry is a message that the LoRa32 has to show on its display. In all the other three cases the variable *devoUscireDalMessaggio* has to be equal to 0, otherwise the code would not finish the *do_send()* function and would not return to the loop to manage the events.

Figure 167 illustrates the Flowchart of the function *do_send()* of the LoRa32 software.

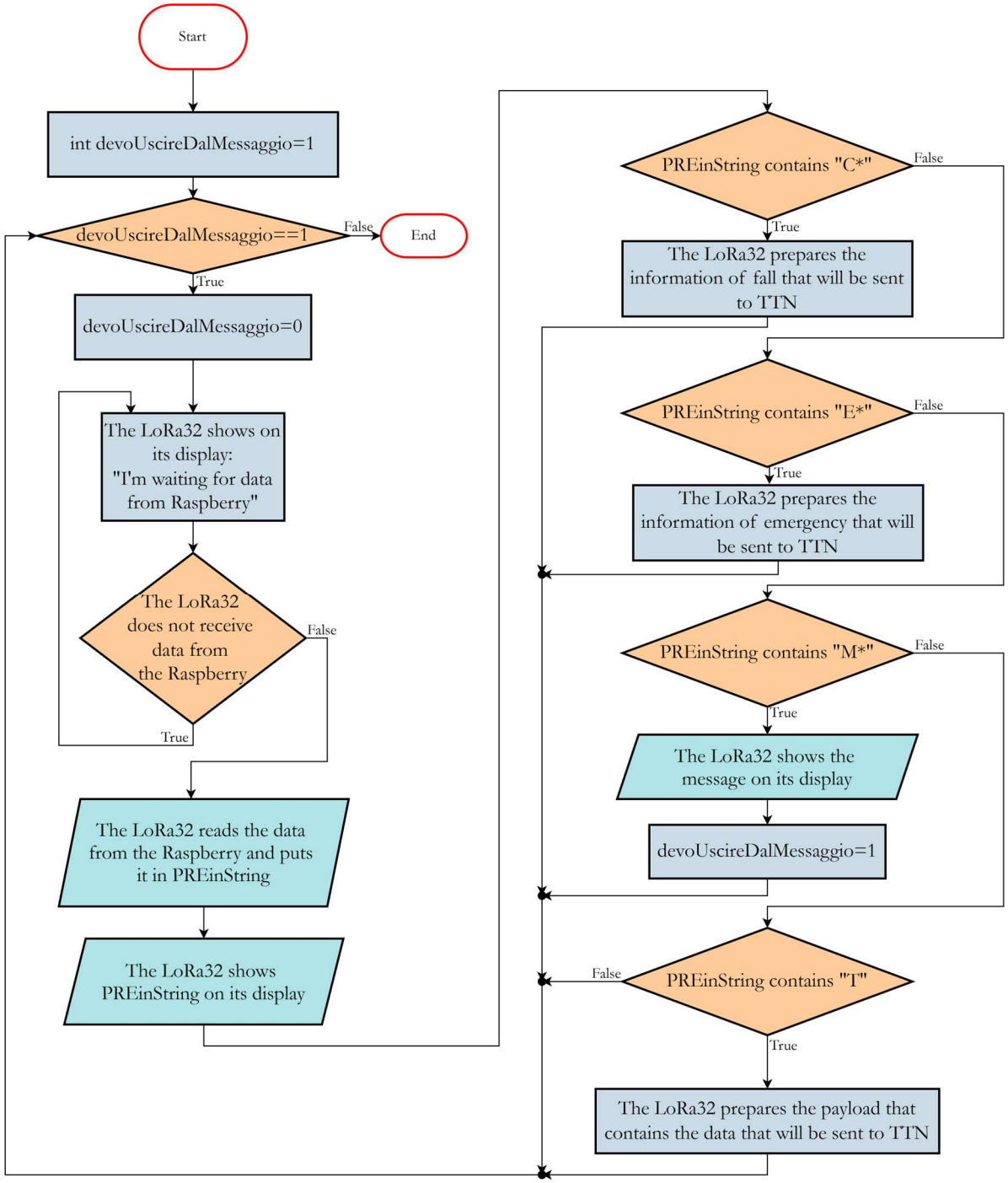

Figure 167 – Flowchart of the *do_send()* function of the LoRa32 Software.

This chapter began with a detailed description of how the sensors and the button are connected into the implemented device. Then, it showed the procedure to configure the Raspberry and the two external servers (Akenza and TTN). Subsequently, it explored the software modules that were specifically written respectively for the Arduino, the Raspberry and the LoRa32 and which allow them to perform their tasks. The next chapter is going to illustrate the various individual tests that were performed on them to demonstrate their individual functioning.

8 EXPERIMENTAL RESULTS FROM THE DEVELOPED PROTOTYPE

This chapter deals with the experiments made to demonstrate the correct functioning of each component of the system. More specifically, the software modules developed for the Arduino, for the LoRa32 and the two software modules for the Raspberry were tested.

8.1 Test of the Software running on the Arduino

By connecting the Arduino to a MacBook via USB and by using the Serial Monitor of the Arduino IDE, it was possible to look at what the Arduino normally sends through the USB connection to the Raspberry (in this specific case to the Serial Monitor). Figure 168 is a screenshot of the Serial Monitor on the MacBook. In this screenshot it is possible to see that the various readings are separated by the letter "F". Within each reading, the letter "S" separates the values of HR and SpO2, expressed respectively in bpm and percentage %.

Figure 168 – Various Readings of HR and SpO2 seen on the Serial Monitor of the Arduino IDE.

Table 32 contains ten readings. The first measurement was made without a finger placed on the sensor, therefore both the values of the HR and of the SpO2 are equal to 0. The second measurement corresponds to the moment immediately after the finger has been placed on the sensor, so this reading was altered. All the other readings are valid.

Table 32 – Ten Readings of HR and SpO2.

Reading	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
HR (bpm)	0.00	73.31	78.95	77.64	78.74	78.26	77.03	78.63	82.55	83.50
SpO2 (%)	0	95	97	97	97	97	97	96	97	95

The first test was carried out to validate the implemented solution by analysing the data obtained from the Arduino and sent to the Raspberry.

Subsequently, a second test was performed with an Arduino software written specifically with the purpose of plotting the values of the Heart Rate and of the SpO2, versus the time, respectively.

Figure 169 is the graph obtained by plotting the values of the Heart Rate that the Arduino collected from the MAX30100 in 207 s of use of the sensor. With a frequency of 1 data per second, the Arduino collects the data from the MAX30100 and sends it to the Raspberry.

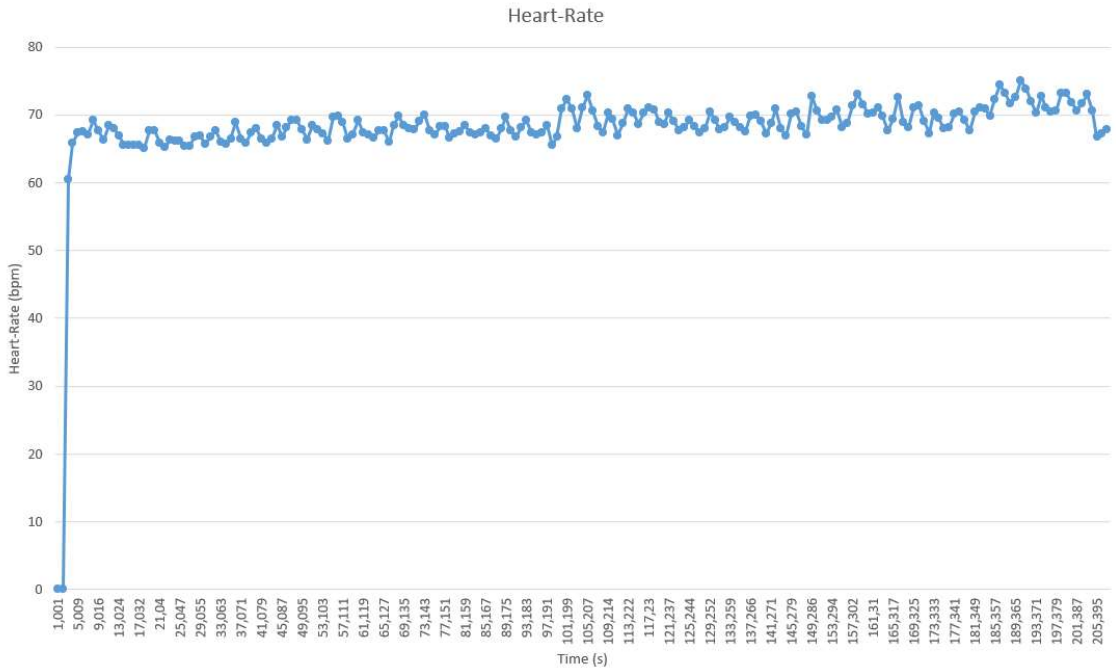

Figure 169 – Experimental Test to plot the Heart Rate versus Time.

All the values of the Heart Rate are between 60.51 bpm and 75.02 bpm.

Figure 170 represents the graph obtained by plotting the values of the SpO2 that the Arduino collected from the MAX30100 together with the values of the Heart Rate in that period of 207 s.

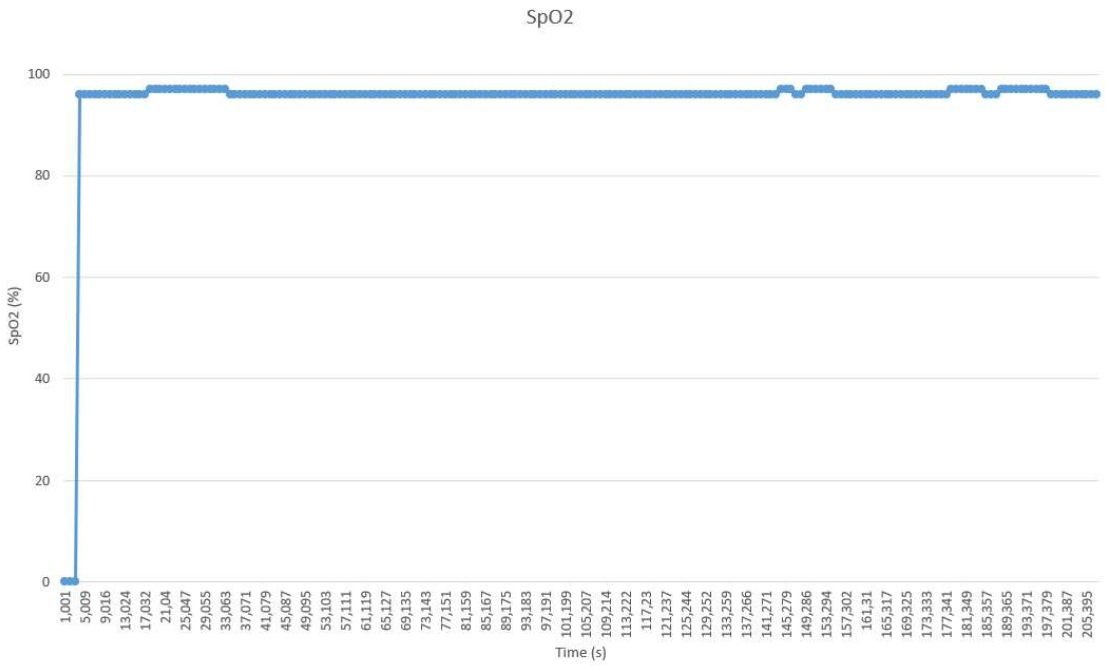

Figure 170 – Experimental Test to plot the SpO2 versus Time.

From the results in Figure 170 it can be observed that all the SpO2 values are between 96% and 97%.

8.2 Test of the Software running on the LoRa32

When the LoRa32 is not busy, it waits for data from the Raspberry, as shown in Figure 171.

Figure 171 –LoRa32 waiting for data from the Raspberry.

When the LoRa32 has to send data to TTN for the first time, it has to firstly do the joining procedure, presented in Figure 172.

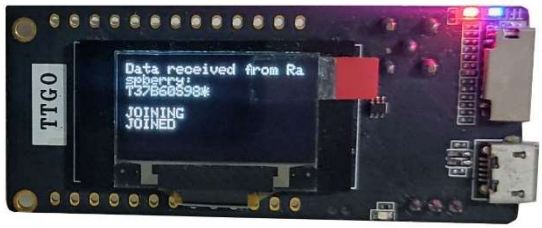

Figure 172 – Joining and Joined events.

The LoRa32 receives the data from the Raspberry via USB. If the LoRa32 is busy doing something else, for example the joining procedure or sending data to TTN, every data from the Raspberry accumulates into the receive buffer of the LoRa32. When the LoRa32 is free and reads what it has received from the Raspberry, in the receive buffer of the LoRa32 there can be more than just one data segment. In this situation the LoRa32 has to read and analyse everything, because among the accumulated data there can be information that has a higher priority. When analysing what is in the receive buffer, the LoRa32 has to respect specific priorities:

- Fall detection. This data has the highest priority, because the fall of the patient can be fatal or can impede the patient from sending the emergency request;
- Emergency;
- Message;
- Sensors data.

Assuming that in the receive buffer of the LoRa32 there is: T37B60S98*E*C*Mare you ok?*, Figure 173 shows the behaviour of the LoRa32 in this case.

In order to simulate this situation the serial monitor of the Arduino IDE was used to send this "accumulated" data to the LoRa32.

```
PREinstring is:
T37B60S98*E*C*MAre you ok?*

instring is:
C
intFallDetected is :
1
3711607: EV_TXSTART
```

Figure 173 – Priority to Fall Detection.

The LoRa32 gave the priority to the fall detection data, so it sent the information of fall to TTN immediately.

If, instead, in the receive buffer of the LoRa32 there is: T37B60S98*E*MAre you ok?*, Figure 174 shows the behaviour of the LoRa32 in this situation.

```
PREinstring is:
T37B60S98*E*MAre you ok?*

instring is:
E
intEmergencyDetected is :
1
7587920: EV_TXSTART
```

Figure 174 – Priority to Emergency.

The LoRa32 gave the priority to the emergency data.

If, instead, in the buffer there is T37B60S98*MAre you ok?*, Figure 175 shows the message that appeared on the display of the LoRa32.

Figure 175 – Priority to Message.

The LoRa32 gave the priority to the message.

Lastly, if instead, in the receive buffer there is: T37B60S98* (Figure 176).

```
PREinstring is:
T37B60S98*

instring is:
T37B60S98
The temperature substring is:
37
The bpm substring is:
60
The SpO2 substring is:
98
floatTemperatura is :
37.0
floatBPM is :
60.00
intTemperatura:
370
intBPM:
6000
intSpO2:
980
10788734: EV_TXSTART
```

Figure 176 – Sensors Data processing on the LoRa32.

In this situation, from the received string, the LoRa32 obtains the Temperature, the HR and the SpO2 substrings, in sequence. Thereafter, the Temperature substring and the HR substring are converted to *float*, while the SpO2 substring is converted to *int*. Then, after the correct multiplications according to their type of data of the Cayenne LPP payload, the data is sent to TTN.

8.3 Test of the Software running on the Raspberry

There are two software modules running on the Raspberry: one module is responsible for the fall detection and the other one for the collection of both the data received from the Arduino (Heart Rate and SpO2) and the data from the ADC (body Temperature).

8.3.1 Test of the Software Module for the Fall Detection

For the purpose of testing the software of the fall detection, the entire device was placed in a belt bag (Figure 177), around the torso of the author, and a fall on a mattress was simulated.

Figure 177 – Device placed in a belt bag.

In order to make the device become mobile, it was powered (more specifically the Raspberry) with a power bank, as shown in Figure 178.

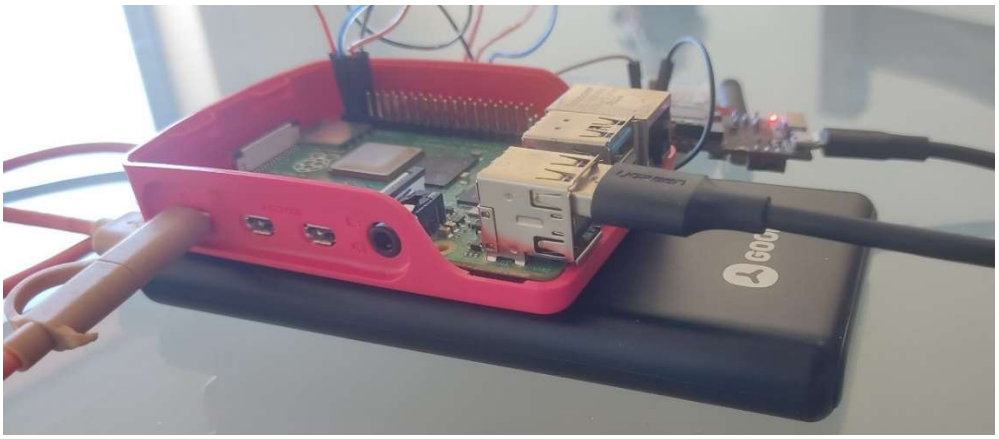

Figure 178 – Device powered by a power bank.

Figure 179 was obtained by plotting the values of the modulo of the acceleration versus time, that were taken from the MPU-6050 accelerometer.

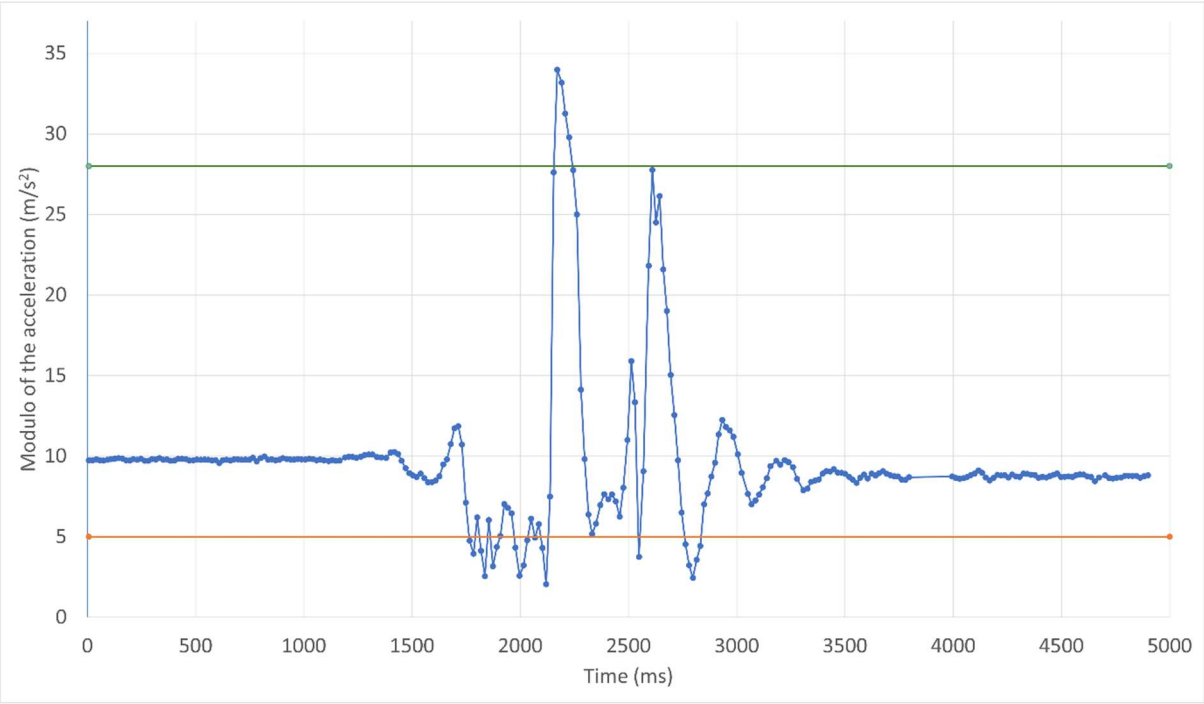

Figure 179 – Experimental Plot of the Modulo of the Acceleration in a Fall.

At the beginning, the modulo of the acceleration is around 9.8 m/s^2, because the person whose movement was under analysis was just staying standing. When the fall started, at around 1435 ms, the modulo of the acceleration began to decrease and dropped below the Lower Fall Threshold, reaching the value 4.75 m/s^2 at 1765.486 ms. Since the first condition for the fall occurred, this made the variable *trigger1* become True. Subsequently, the modulo of the acceleration rapidly increased and exceeded the Upper Fall Threshold, reaching a maximum value of 33.970 m/s^2 at 2171.118 ms. The second condition occurred, so *trigger2* became True. This happened in a time within 500 ms from the first time that *trigger1* had become True. It was, then, necessary to calculate the modulo of the angular velocity taken at 2171.118 ms (when the Upper Fall Threshold was surpassed), which was 258.593 °/s. Since this value was between 30 °/s and 400 °/s, also the third condition was met and *trigger3* immediately became True (so within 500 ms from the first time that *trigger2* had become True). Then at 3796.105 ms, after 1624.987 ms from when *trigger3* had become True, the modulo of the angular velocity obtained was 2.3671 °/s. Since this value is between 0 °/s and 10 °/s, it implies that the person under analysis was lying on the mattress, thus meaning that also the fourth and last condition occurred. This made the variable *fall* become True.

It is important to observe that, from the moment when *trigger3* became True until the moment when the fourth condition was analysed, 90·0.01 s=900 ms should have passed. However, it passed a time of 1624.987 ms, because the various instructions take time to be executed, so the time of one cycle is a little more than 0.01 s.

Between 3796.105 ms and 3994.966 ms there is no data, because in that short time the Raspberry was busy sending the information of the fall to Akenza via MQTT.

The fluctuation of the modulo of the acceleration after the maximum peak is caused by the bounces on the mattress. If, instead, the fall occurs on a hard surface, these fluctuations are not present or are very small.

Subsequently, the developed prototype was used to monitor different daily activities and to verify that the software module of the fall detection did not interpret them as falls.

Walking

Figure 180 shows the experimental behaviour of the modulo of the acceleration in a walking activity.

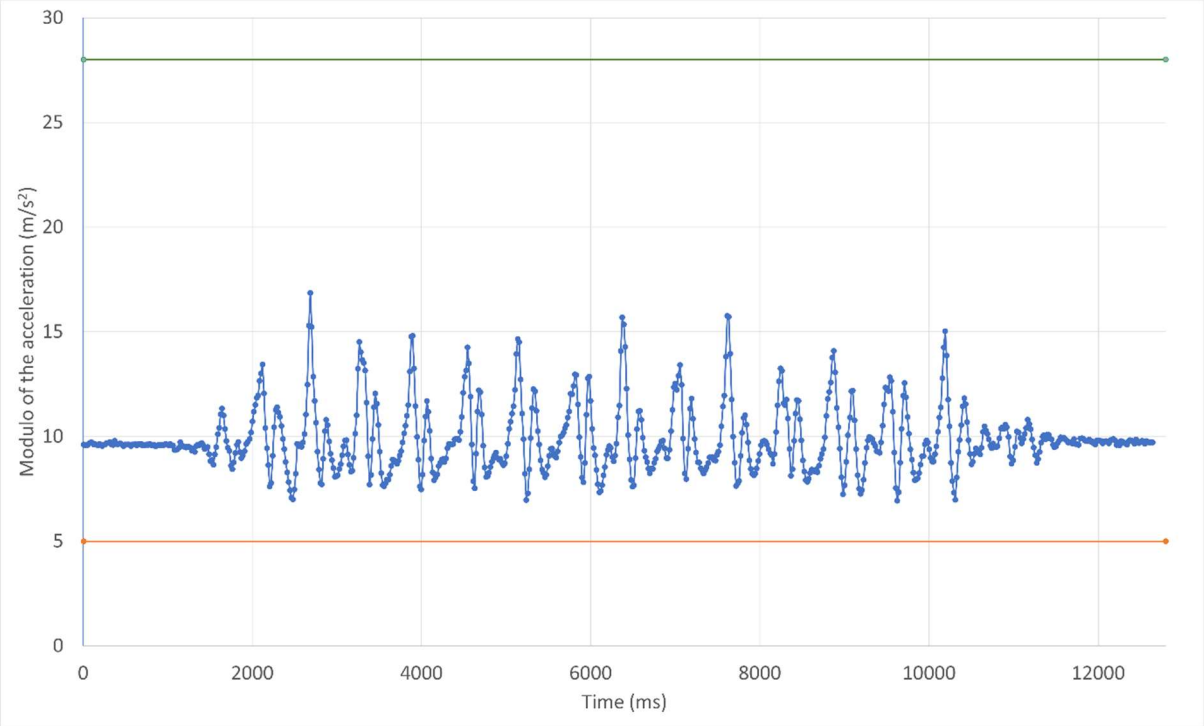

Figure 180 – Experimental Plot of the Modulo of the Acceleration in a Walking activity.

The fluctuations follow the pace of the various steps. The lower peaks do not reach the Lower Fall Threshold and, even when they are near to it, the upper peaks are extremely low compared to the Upper Fall Threshold. This means that even if the first condition had occurred, the second condition would have never occurred.

Running

Figure 181 illustrates the experimental behaviour of the modulo of the acceleration in a running activity.

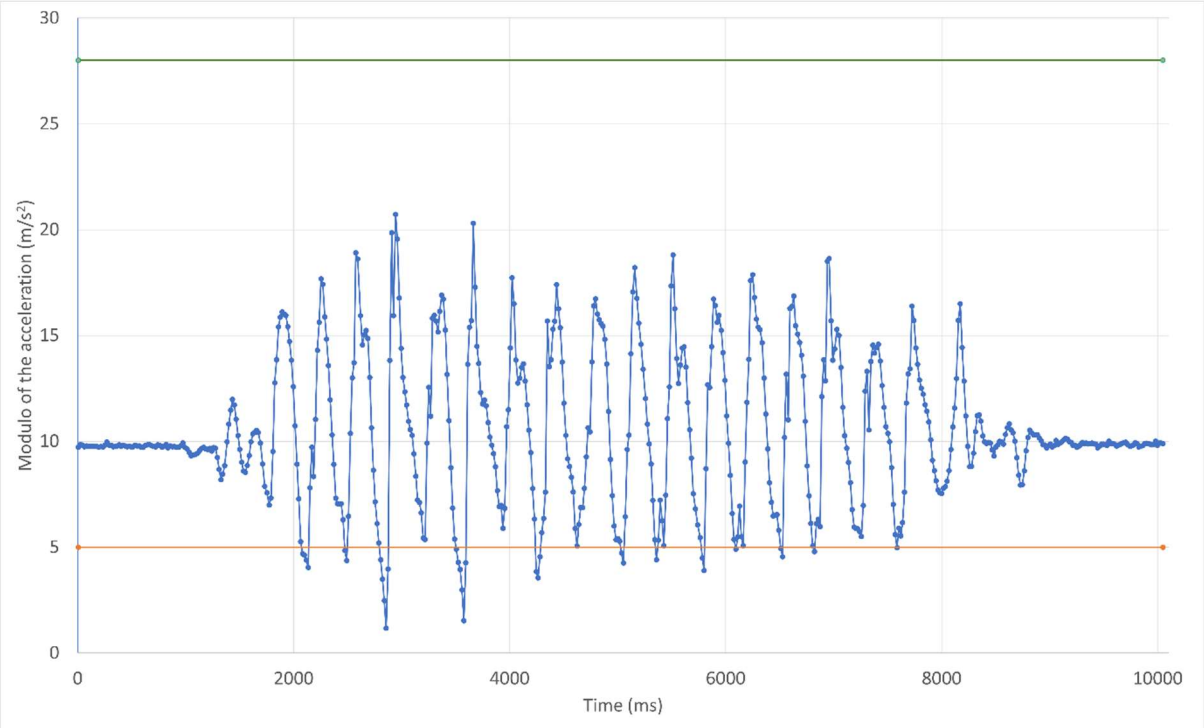

Figure 181 – Experimental Plot of the Modulo of the Acceleration in a Running activity.

In the case of a running activity, the behaviour is similar to that of the walking activity, but in this case various lower peaks exceed the Lower Fall Threshold. However, only the first condition of the fall occurred, while the upper peaks are far when compared to the Upper Fall Threshold and the second condition never occurred.

Jump

Figure 182 shows the experimental behaviour of the modulo of the acceleration in a jump activity.

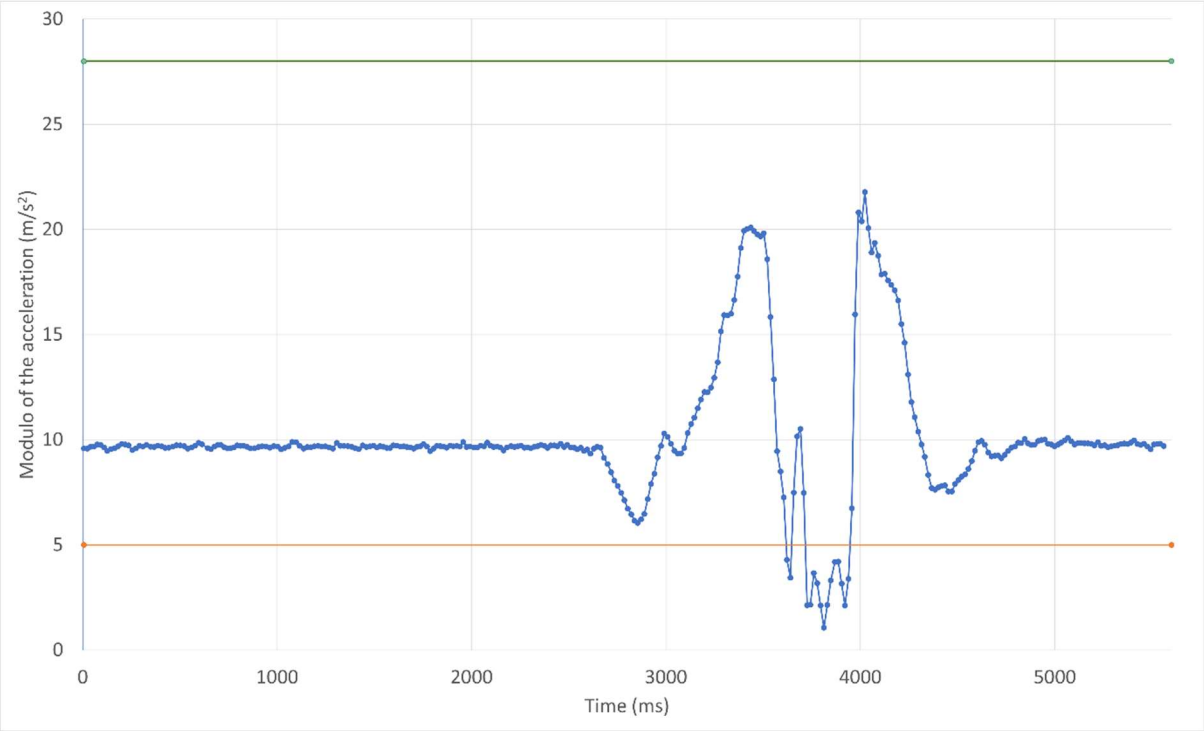

Figure 182 – Experimental Plot of the Modulo of the Acceleration in a Jump activity.

In this case a jump was made starting from a still upright position. In the first phase, while loading the jump, the person whose movement was under analysis descended by bending the knees, so the modulo of the acceleration decreased. During the jump, the modulo of the acceleration increased until it reached a peak in correspondence of reaching the highest position in the jump. From this point until the impact of the toes with the ground, the modulo of the acceleration decreased because the person was descending. After the impact of the toes with the ground, the modulo of the acceleration increased slightly. Immediately after the contact of the toes with the ground, the body continued to descend, so the modulo of the acceleration decreased until the moment of the impact of the heels with the ground. At this point the modulo of the acceleration had a huge peak.

In this experiment, only the Lower Fall Threshold was surpassed and, even if also the Upper Fall Threshold was surpassed, this jump activity would have not been mistakenly detected as fall, because the body did not rotate.

Sitting down and Standing up

Figure 183 illustrates the experimental behaviour of the modulo of the acceleration in a sitting and standing up activity.

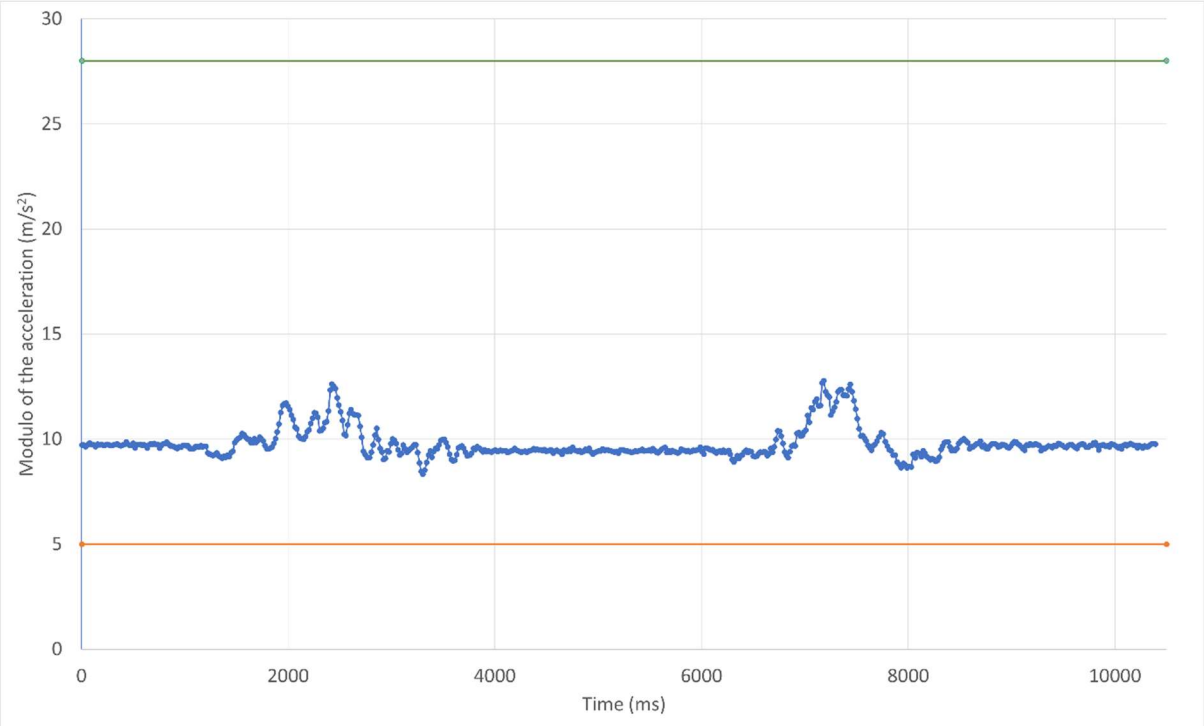

Figure 183 – Experimental Plot of the Modulo of the Acceleration in a Sitting and Standing up activity.

In this experiment the person being analysed was sitting down on a chair and stood up from it. From around 1200 ms until around 4000 ms it is the seating activity, while from 6200 ms it is the standing up activity.

In this experiment none of the thresholds was surpassed, so this activity can never be seen as a fall.

Lying down and Standing up

Figure 184 shows the experimental behaviour of the modulo of the acceleration in a lying down and standing up activity.

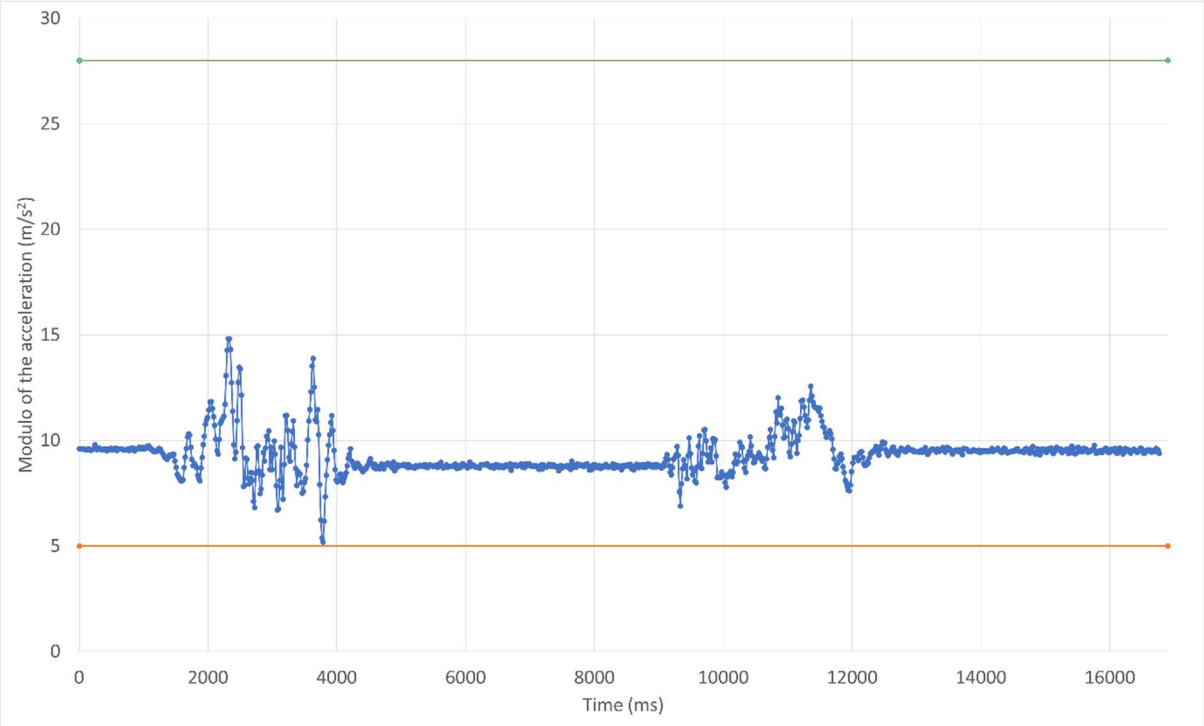

Figure 184 – Experimental Plot of the Modulo of the Acceleration in a Lying down and Standing up activity.

In this case in the first phase, from 1000 ms to 4300 ms, the person under analysis lied down and, in the second phase, from 8400 ms to 12000 ms, stood up. Also in this case the values of the modulo of the acceleration never surpass the two thresholds, so this activity can never been mistakenly detected as a fall.

8.3.2 Test of the Software Module for the Data Collection

Every 10 seconds the Raspberry takes the data from the Arduino and reads the data from the ADC. The data coming from the Arduino was analysed in the dedicated paragraph. Figure 185, instead, represents the data that the Raspberry read from the ADC in a test of 160 s. In this test a Raspberry software module was specifically written with the purpose of plotting the values of the temperature versus time. Just before running the software, the thermistor probe (MA300 NTC) was placed under the armpit. The MA300 is a NTC probe thermistor specifically built to measure the body temperature and its surface perfectly adheres to the skin.

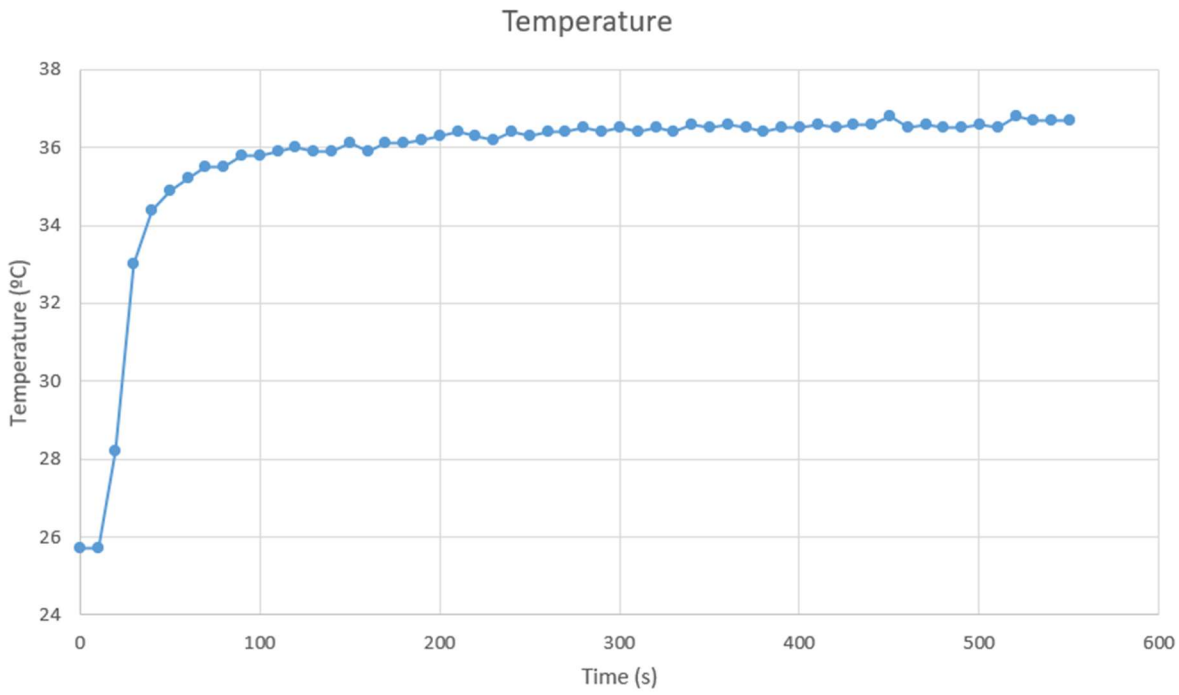

Figure 185 – Experimental Test to plot the Temperature versus Time.

When the software was executed, the NTC was at room temperature because it still had not been placed under the armpit. Then it was positioned and, from 10 s up to 200 s, the temperature raised until reaching the thermal equilibrium. Then, from 200 s onward, the temperature stayed around an average of 36.5 ºC.

This chapter described all the tests that were performed on each component of the system to demonstrate their correct functioning. All the test produced the expected results and the next chapter is going to deal with the use of the device in its entirety.

9 FIELD TEST RESULTS ANALYSIS FROM PLATFORM DASHBOARD

This section presents and analyses the field test results retrieved from the developed device during the day 13/10/2024 in the period ranging from 12h:15m to 14h:20m.

During the use of the electronic BSN system, its parts, namely: the Raspberry, the LoRa32, the power bank, the accelerometer, the button and the ADC, were inside of a belt bag. The Arduino, instead, as can be seen in Figure 186, was tied to the person's hand with a hair tie, while the MAX30100 was held steady on the finger with a plaster.

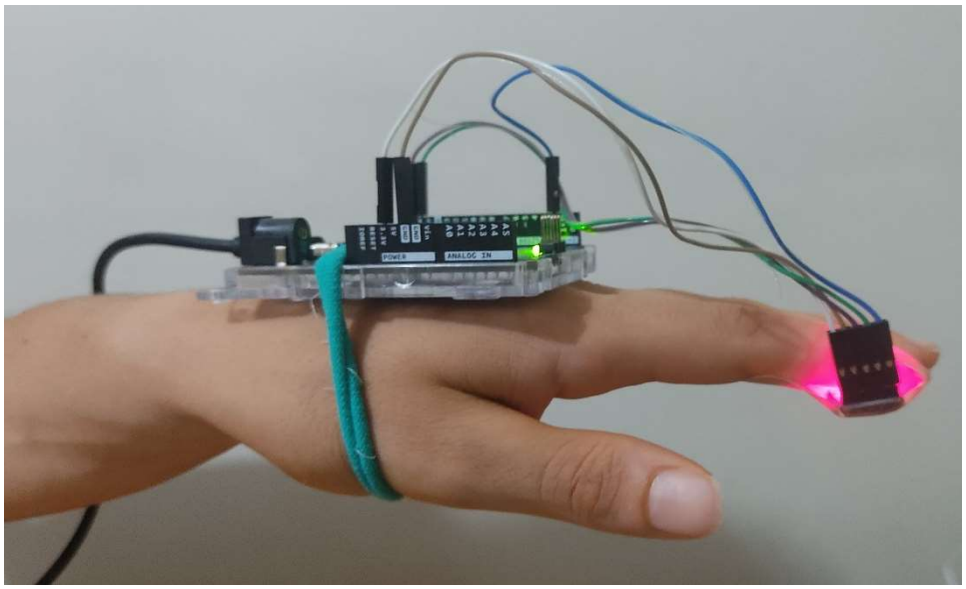

Figure 186 – Arduino and MAX30100 tied on the hand.

The MA300 NTC was sticked down by a plaster under the armpit.

Figure 187 shows the graphs displayed on the Akenza Dashboard at the end of the use of the device. These graphs show the actual data that the developed prototype sent to Akenza during the field tests, namely: body Temperature, HR, SpO2 and Emergency button occurrences.

Figure 187 – Akenza Dashboard after the Experience.

Through the use of a smartwatch, the heartbeat measured by the system was validated at 12h19m. Figure 188 shows the values of the data received by Akenza.

Temperature: 36.6 HeartRate: 99.15 SpO2: 97 13/10/2024, 12:19:59

Figure 188 – Sensors Data at 12:19 on Akenza.

The value of the Heart Rate was equal to 99.15 bpm. Figure 189 shows the simultaneous use of both the developed prototype and the smartwatch.

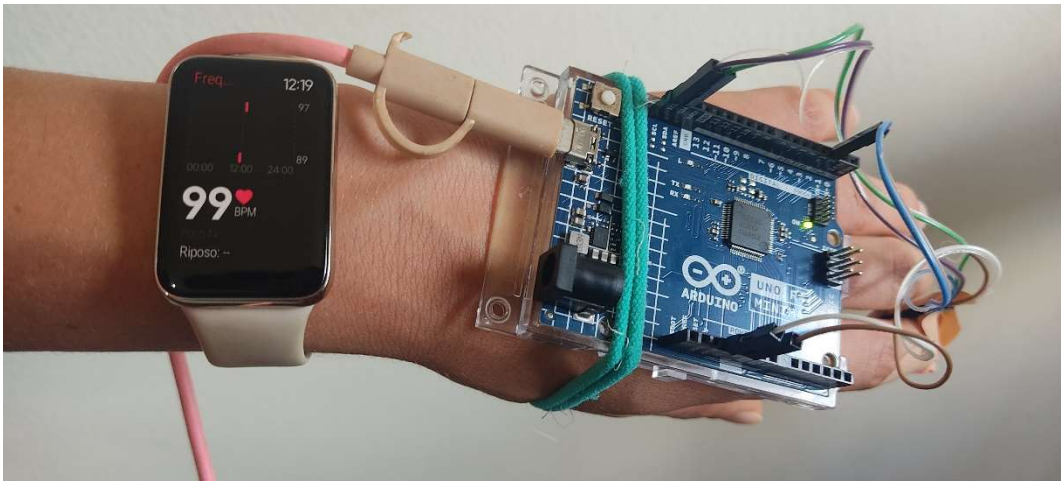

Figure 189 – Use of the smartwatch to confirm the data collected from the MAX30100.

The smartwatch displays the same Heart Rate value (99bpm). Up until 12h:37m, the person carrying the device was at home performing normal daily activities, such as staying standing upright, walking and sitting on the sofa. As the person was in the Wi-Fi range, all this data was sent to Akenza through the Wi-Fi and Internet data flow. On the developed Akenza Dashboard this data is graphically plotted in red. While sitting on the sofa, at 12h:28m, the emergency button was pressed to simulate an Emergency event.

Figure 190 demonstrates that the request for emergency was immediately sent and received in Akenza via TTN.

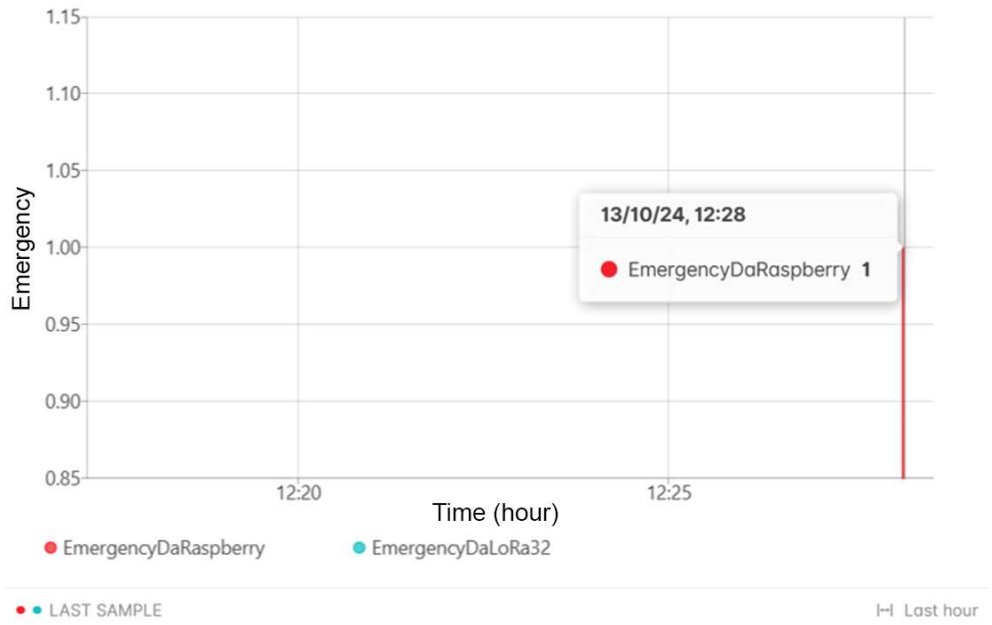

Figure 190 – Emergency sent when in the Wi-Fi range.

Also in this case, the Emergency was sent via the Wi-Fi and Internet path. As soon as Akenza received the request for Emergency, it sent an email, as shown in Figure 191.

Figure 191 – Akenza's email of warning.

Subsequently, from 12h:37m until 12h:44m the person went outside in the garden, out of the Wi-Fi range. Figure 192 shows the body Temperature data that was received by Akenza in this period of time.

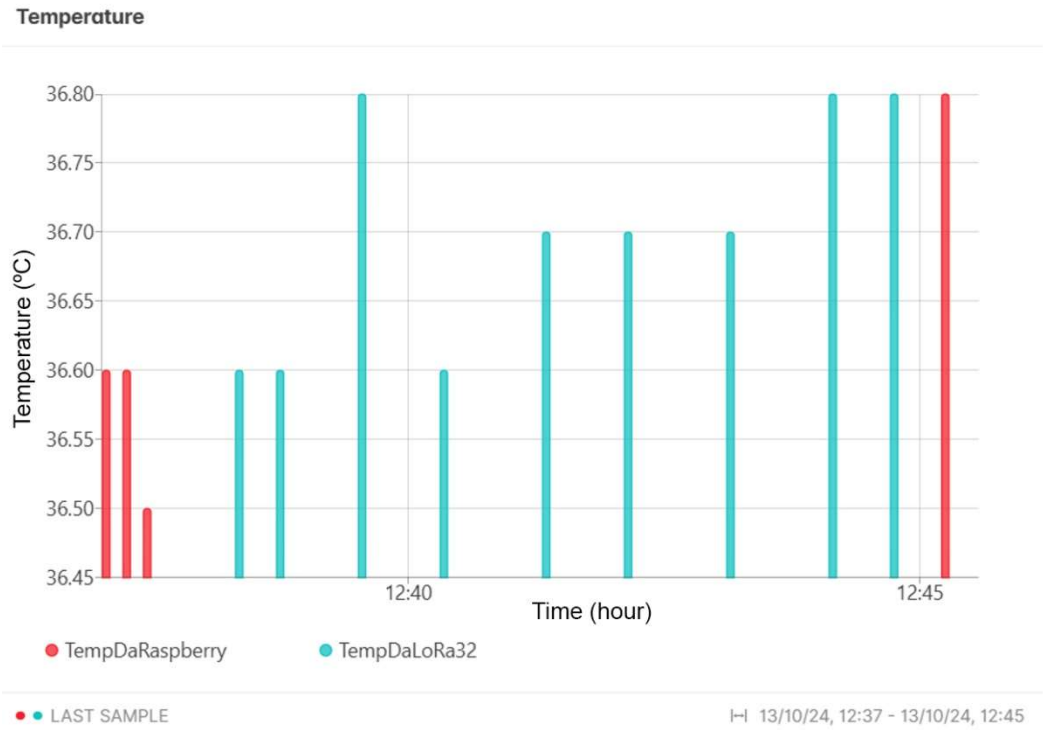

Figure 192 – Body Temperature data sent via LoRaWAN.

This data is plotted in a different colour (green), because it was sent to Akenza through the LoRaWAN path.

While in the garden, at 12h:40m, the person pressed the button two times to send two requests of Emergency. Figure 193 demonstrates that Akenza immediately received both the requests through the LoRaWAN path.

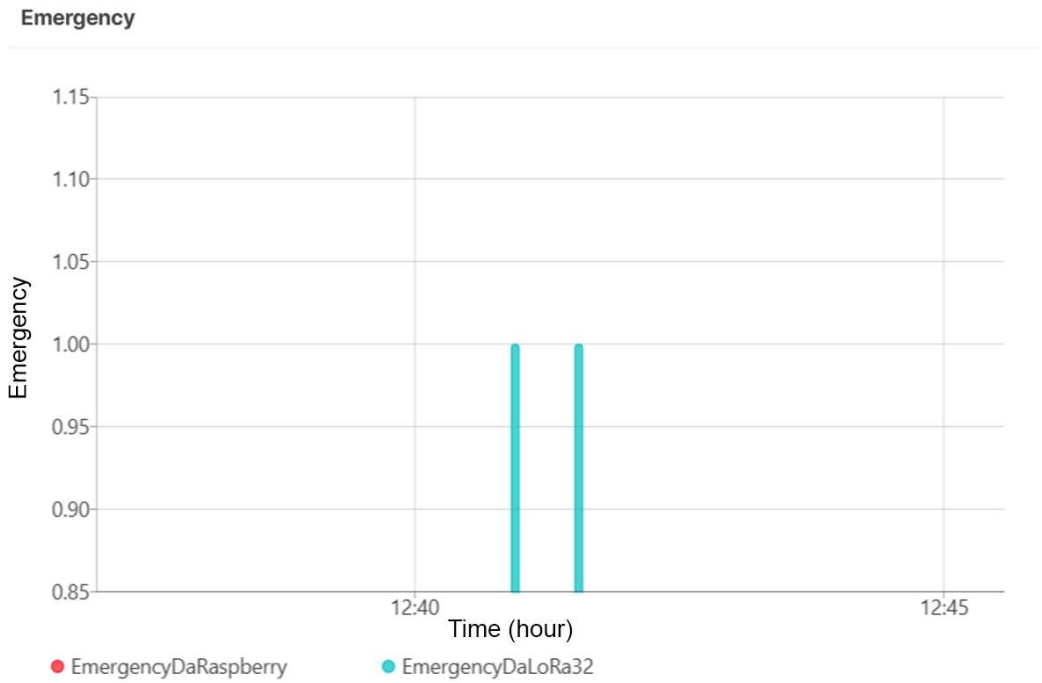

Figure 193 – Emergencies sent via LoRaWAN.

As soon as the requests reached Akenza, it instantly sent two emails, as shown in Figure 194.

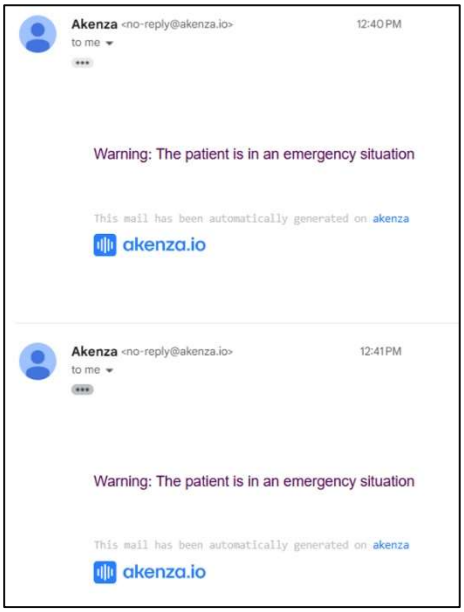

Figure 194 – Akenza's emails of warning.

At 13h:17m an hour had passed from the starting of the experience and the current time was between the 08h:00m and the 20h:00m, so the display of the system showed the message "Are you ok? Press the button". In this situation the button was purposely not pressed, in order to simulate that the person was not feeling ok. Figure 195 shows that in this occasion, a request for Emergency was sent to Akenza.

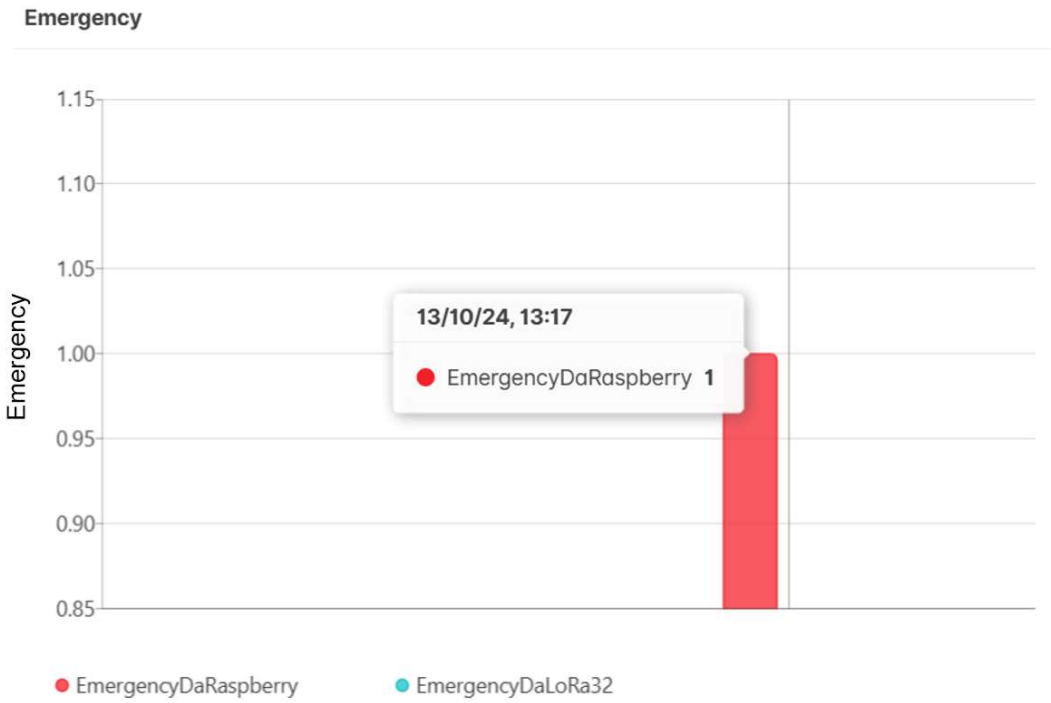

Figure 195 – Emergency automatically sent when the patient is not ok.

As shown in Figure 196, Akenza immediately sent an email of warning.

Figure 196 – Akenza's email of warning.

At this point the person under analysis continued to perform daily activities and, from 13h:59m to 14h:06m, fell asleep. How it is possible to notice in the highlighted zone in Figure 197, while the person was sleeping, the cardiac activity slowed down.

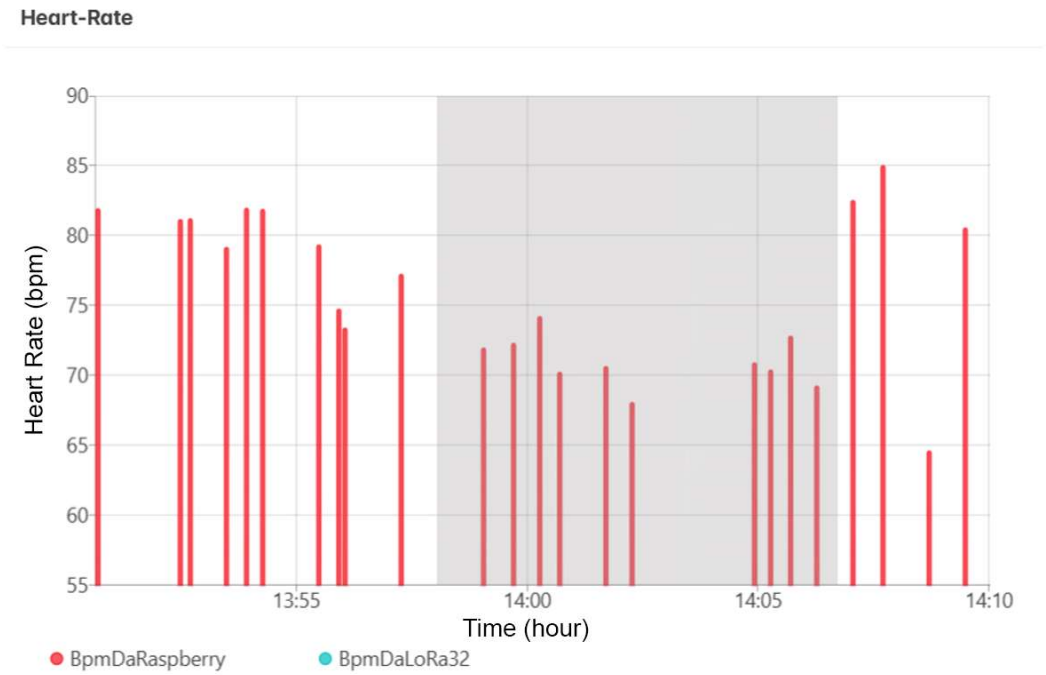

Figure 197 – Heart Rate during Sleep phase.

At 14h:18m the system displayed again the message "Are you ok? Press the button", but this time the button was pressed, so no Emergency was sent to Akenza. At this point the experience was stopped.

In the graphs there are few Heart Rate and SpO2 anomalous values, due to the fact that the MAX30100 sensor was weakly held steady by a plaster and sometimes it loosened and it was necessary to adjust it. In order to avoid these irregularities, it is important to secure the sensor to the finger in a more professional way.

At 14h:40m a new experience was initiated with the purpose of simulating a fall, while using the entire system. Figure 198 illustrates the graphs on the Akenza Dashboard derived from this second experience.

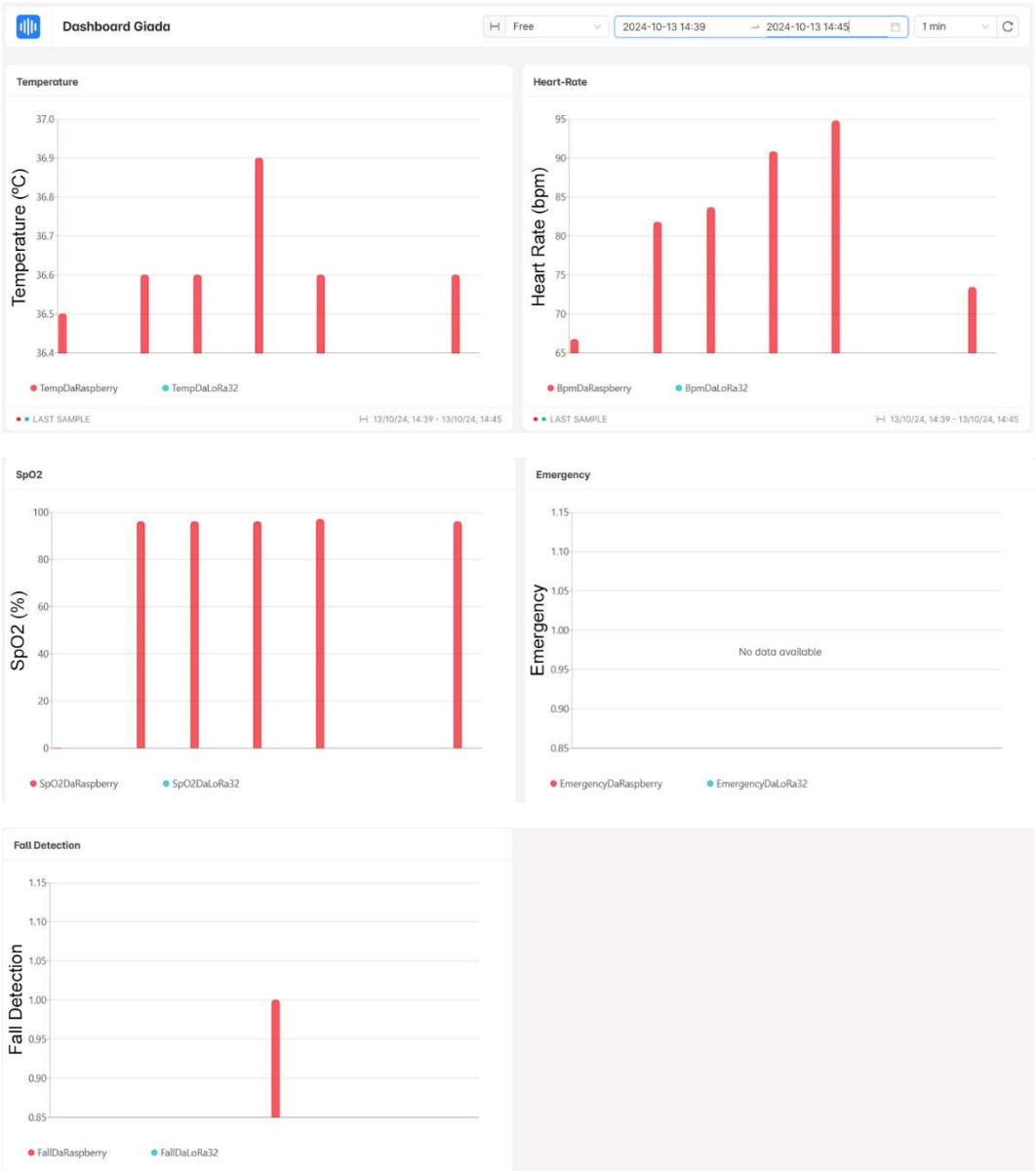

Figure 198 – Akenza Dashboard after the Fall Experience.

As can be seen in Figure 199, at 14h:41m the device detected a fall and sent this information to Akenza.

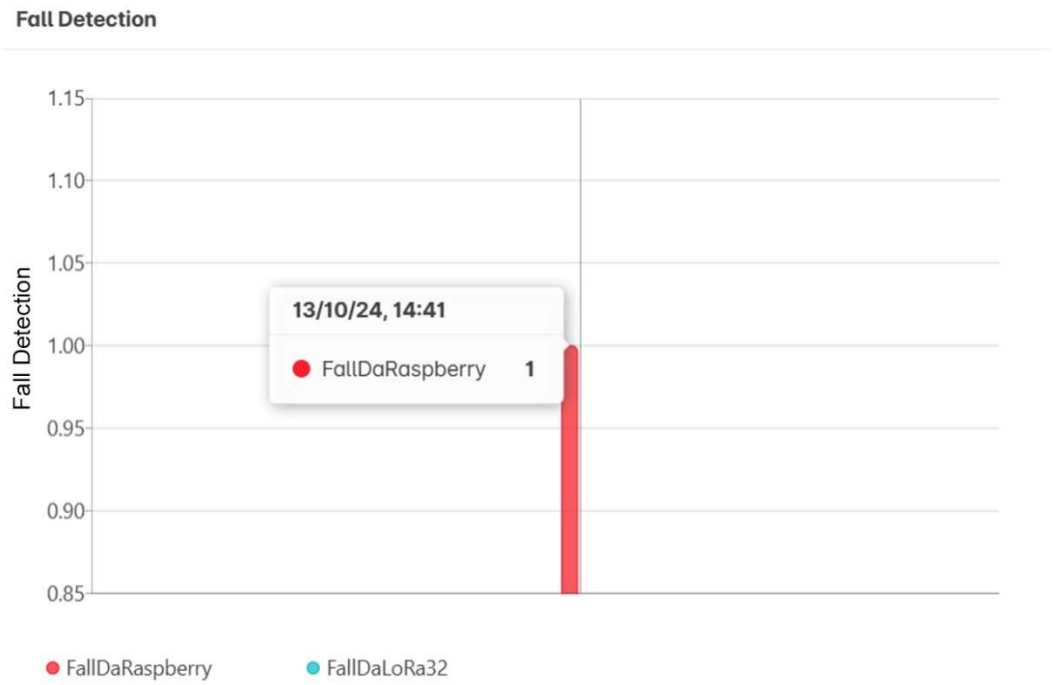

Figure 199 – Fall detected sent to Akenza.

Figure 200 shows that Akenza, as a consequence, sent an email of warning.

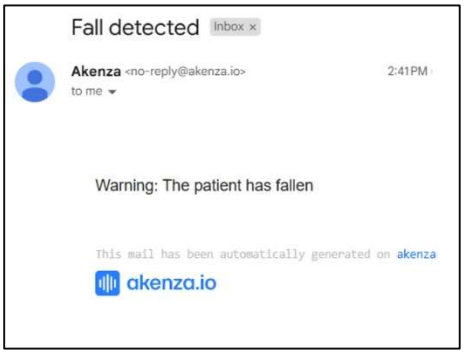

Figure 200 – Akenza's email of warning after the Fall.

This chapter demonstrated that the device of the implemented system can truly be used for its purposes and that the collected data are perfectly compliant with the expectations.

10 CONCLUSIONS AND FUTURE WORK

The objective of this project was to develop, implement and test a Health Monitoring System for Elderly People, based on a BSN, and with a wearable system architecture, which is suited for patients that are not in critical condition but need a continuous or a periodical monitoring.

The developed system collects the data of the body temperature, the heartbeat, the SpO2 and checks if the patient has fallen, and stores this data in both a local database and the Akenza platform. The doctor can remotely access the Akenza Dashboard to analyse the patient's health condition, without the need for the patient to visit the doctor or the hospital.

The MA300 NTC thermistor was selected to monitor the body temperature, being connected to the skin under the armpit. The MCP3008 ADC was used to connect, via SPI, the NTC with a Raspberry. The MPU-6050 Six-Axis (Accelerometer/ Gyroscope) was used to detect if the patient has fallen. The communication between this sensor and the Raspberry is of the I^2C type. The MAX30100 Heart Rate Sensor and Pulse Oximeter was used to measure the heartbeat per minute and to analyse the level of oxygen in the blood of the patient. The use of an Arduino R4 Minima was necessary to read the data via I^2C from the MAX30100, and to send it to the Raspberry via USB.

A dual-purpose button was used as an emergency button and, only on some occasions, as a confirmation button to interact with the device.

It was possible to reach Akenza through one out of two possible data paths:

- Whenever the device was within the Wi-Fi range, the Raspberry used the Wi-Fi connection to send the collected data to the home router. Then, the home router sent the data directly to Akenza through the MQTT protocol over TCP/IP.
- When the device was out of the Wi-Fi network range, the Raspberry sent, through the USB connection, the collected data to the LoRa32, which used the LoRaWAN technology to send the data to a LoRaWAN gateway. Subsequently, this LoRaWAN Gateway sent the data to TTN through the TLS protocol over TCP/IP. Then, TTN sent the data to Akenza, through a webhook integration.

Various tests were executed on each sensor, with the purpose of showing its individual correct functioning. In particular, the MPU-6050 was tested not only during a fall, but also during several daily activities.

Thereafter, two long tests were executed to show the correct functioning of the entire system. During those tests, the system worked exactly as expected and was able to send all the data to Akenza, using the first path or the second one, according to the situation. Moreover, the system was able to correctly detect all the falls and the daily activities were never misinterpreted as falls.

The Akenza Dashboard graphically illustrated the plots of the body temperature, the heart rate and the SpO2, versus time. It also showed the emergency situations and the detected falls. Moreover, for each emergency and fall detected, Akenza sent a warning to the chosen email address.

Future developments involve the design of a PCB, in order to have more stability in the connections of the various components of the system and to, therefore, avoid the possible disconnection of some wires. Moreover, the MAX30100 sensor needs a more stable support that can secure it to the finger of the patient.

The entire system can be customised according to the needs of different patients, so new types of messages can be introduced and new Akenza Rules can be set.

BIBLIOGRAPHY

[1] A. Rashed, "Integrated IOT medical platform for remote healthcare and Assisted Living," *Japan-Africa Conference on Electronics, Communications and Computers (JAC-ECC),* Dec. 2017.

[2] J. Ausubel, "Older people are more likely to live alone in the U.S. than elsewhere in the world," Pew Research Center, [Online]. Available: https://www.pewresearch.org/short-reads/2020/03/10/older-people-are-more-likely-to-live-alone-in-the-u-s-than-elsewhere-in-the-world .

[3] J. Molinsky, "The number of people living alone in their 80s and 90s is set to soar," Joint Center for Housing Studies, [Online]. Available: https://www.jchs.harvard.edu/blog/the-number-of-people-living-alone-in-their-80s-and-90s-is-set-to-soar.

[4] "A survey on elderly loneliness in the wake of covid-19," Elder, [Online]. Available: https://www.elder.org/the-elder/survey-on-elderly-loneliness/.

[5] M. Schade, "Body temperature: Normal ranges & how to measure," cosinuss°, [Online]. Available: https://www.cosinuss.com/en/measured-data/vital-signs/body-temperature/ .

[6] T. N. Gia, "IOT-based fall detection system with energy efficient sensor nodes," *2016 IEEE Nordic Circuits and Systems Conference (NORCAS),* vol. 13, p. 1–6, Nov. 2016.

[7] M. H. Alsulami, A. S. Atkins and R. J. Campion, "The use of smart watches to monitor heart rates in elderly people: A complementary approach," *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA),* vol. 8, p. 1–6, Nov. 2016.

[8] S. Natarasan and P. Sekar, "Design and implementation of heartbeat rate and SpO2 detector by using IOT for patients," *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC),* vol. 9, p. 630–636, Jul. 2020.

[9] A. Ometov, "A survey on wearable technology: History, state-of-the-art and current challenges," *Computer Networks,* vol. 193, p. 108074, Jul. 2021.

[10] "Apple Watch Series 10," Apple, [Online]. Available: https://www.apple.com/apple-watch-series-10/.

[11] D. Schoder, "Introduction to the internet of things," *Internet of Things A to Z,* p. 1–50, May 2018.

[12] "(PDF) a quantitative analysis of a novel network-based intrusion detection system over Raspberry Pi," [Online]. Available: https://www.researchgate.net/publication/336825947_A_quantitative_analysis_of_a_novel_Network-based_Intrusion_Detection_System_over_Raspberry_Pi.

[13] A. Calihman, “IOT architectures - common approaches and ways to design IOT at scale,” NetBurner, Inc, [Online]. Available: https://www.netburner.com/learn/architectural-frameworks-in-the-iot-civilization/ .

[14] L. W. Turner, Electronics Engineer’s Reference Book, London: Butterworth & Co. (Publishers) Ltd, 1976.

[15] Amphenol Advanced Sensors, “Thermometric MA300 Biomedical Chip NTC Thermistors,” [Online]. Available: https://www.amphenol-sensors.com/hubfs/Documents/AAS-920-321E-Thermometrics-NTC-Type-MA-033020-web.pdf.

[16] “ADC ad approssimazioni successive,” Elemania, [Online]. Available: http://www.elemania.altervista.org/adda/architetture/arc6.html.

[17] Microchip, “MCP3004/3008 (Datasheet),” [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MSLD/ProductDocuments/DataSheets/MCP3004-MCP3008-Data-Sheet-DS20001295.pdf.

[18] M. Benmessaoud and M. M. Nasreddine, “Optimization of MEMS capacitive accelerometer - Microsystem Technologies,” SpringerLink, [Online]. Available: https://link.springer.com/article/10.1007/s00542-013-1741-z#Sec3.

[19] V. Murthy, S. A, H. Suresh and A. S, “Simulation and analysis of suspension based single axis mems capacitive accelerometer,” *International Journal of Basic and Applied Science,* vol. 11, no. 3, pp. 107-116, Dec.2022.

[20] M. Andrejašic, “MEMS ACCELEROMETERS,” University of Ljubljana, 2008. [Online]. Available: https://faculty.uml.edu/xwang/16.541/2010/MEMS_accelerometers.pdf.

[21] Last Minute ENGINEERS, “Interface MPU6050 Accelerometer and Gyroscope Sensor with Arduino,” [Online]. Available: https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/.

[22] R. Magalhães, “How to use MPU6050 6-Axis MEMS Motion Tracking Sensor (Gyroscope + Accelerometer) with Arduino,” Compraco, 25 June 2024. [Online]. Available: https://compraco.com.br/en/blogs/tecnologia-e-desenvolvimento/como-usar-o-sensor-de-rastreamento-de-movimento-mems-mpu6050-de-6-eixos-giroscopio-acelerometro-com-arduino.

[23] InvenSense, “MPU-6000 and MPU-6050 Product Specification Revision 3.4 (Datasheet),” 08 19 2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[24] RudraNarayanG, “MPU 6050 Gyro,Accelerometer Communication With Arduino (Atmega328p),” AUTODESK Instructables, [Online]. Available: https://www.instructables.com/Accelerometer-MPU-6050-Communication-With-AVR-MCU/.

[25] InvenSense, “MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2,” 19 08 2013. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf.

[26] G. M. Azmal, A. Al-Jumaily and M. Al-Jaafreh, "Continuous Measurement of Oxygen Saturation Level using Photoplethysmography Signal," *ICBPE 2006 - Proceedings of the 2006 International Conference on Biomedical and Pharmaceutical Engineering,* pp. 504 - 507, 2007.

[27] InformedHealth.org, "In brief: How does the heart work?," National Library of Medicine, 6 June 2023. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK279249/.

[28] [Online]. Available: https://pt.dreamstime.com/ilustra%C3%A7%C3%A3o-do-vetor-de-fluxo-sangu%C3%ADneo-card%C3%ADaco-explica%C3%A7%C3%A3o-sistema-cardiol%C3%B3gico-rotulada-com-veia-dos-pulm%C3%B5es-e-corpo-%C3%A0-image183596531.

[29] [Online]. Available: https://www.wikilectures.eu/w/File:Circulations.jpg#file.

[30] HeartPlan, "HeartPlan Resource Booklet Edition 1," 02 2019. [Online]. Available: https://galwaymedicalcentre.com/wp-content/uploads/2020/04/heartplan_booklet_20190703.pdf.

[31] maxim integrated, "MAX30100 Pulse Oximeter and Heart-Rate Sensor IC (Datasheet)," [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX30100.pdf.

[32] P.-Y. Cheang and P. R. Smith, "An Overview of Non-contact Photoplethysmography," *ELECTRONIC SYSTEMS AND CONTROL DIVISION RESEARCH 2003,* 2023.

[33] M. Ghamari, "A review on wearable photoplethysmography sensors and their potential future applications in health care," *International Journal of Biosensors &amp, Bioelectronics,* vol. 4, no. 4, 2018.

[34] "Optical absorption of hemoglobin," [Online]. Available: https://omlc.org/spectra/hemoglobin/ .

[35] A. İ. Bülbül and S. Küçük, "Pulse Oximeter Manufacturing & Wireless telemetry for Ventilation Oxygen Support," *International Journal of Applied Mathematics, Electronics and Computers,* p. 211–211, Dec. 2016.

[36] RedHat, "What is IoT Edge computing?," 29 July 2022. [Online]. Available: https://www.redhat.com/en/topics/edge-computing/iot-edge-computing-need-to-work-together.

[37] [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/images/5062/MFG_SC0192%289%29.jpg.

[38] "Raspberry Pi: third best-selling computer of all time and the making of a global community," University of Cambridge, 3 August 2021. [Online]. Available: https://impactmap.cam.ac.uk/raspberry-pi-third-best-selling-computer-of-all-time-and-the-making-of-a-global-community/.

[39] Raspberry Pi Trading Ltd, “Raspberry Pi 4 Computer Model B,” June 2019. [Online]. Available: https://docs.rs-online.com/b1fb/0900766b816fa153.pdf.

[40] “Raspberry Pi Documentation,” [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

[41] G. Bernardo, “La tecnologia LoRa – La scheda di sviluppo TTGO LoRa32 – Parte 1 – introduzione,” SETTOREZERO, [Online]. Available: https://www.settorezero.com/wordpress/lora_lorawan_lilygo_ttgo_lora32_esp32/.

[42] “TTGO ESP32-Paxcounter LoRa32 V2.1 - 868Mhz,” OPENCIRCUIT, [Online]. Available: https://opencircuit.pt/product/ttgo-esp32-paxcounter-lora32-v2.1-868mhz.

[43] ESPRESSIF, “ESP32 Series Datasheet Version 4.7,” [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[44] Arduino, “Product Reference Manual,” 28 10 2024. [Online]. Available: https://docs.arduino.cc/resources/datasheets/ABX00080-datasheet.pdf.

[45] “Arduino® UNO R4 Minima,” [Online]. Available: https://store.arduino.cc/en-pt/products/uno-r4-minima.

[46] K. Söderby and J. Hylén, “Getting Started with Arduino IDE 2,” 28 10 2024. [Online]. Available: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/#overview..

[47] “I2C-Bus: What’s that?,” I2C-Bus, [Online]. Available: https://www.i2c-bus.org/.

[48] C. Wu, “Tutorial: Arduino and the I2C bus – Part One,” tronixstuff.com, 20 October 2010. [Online]. Available: https://tronixstuff.com/2010/10/20/tutorial-arduino-and-the-i2c-bus/.

[49] “Difference Between BJT, MOSFET and IGBT: BJT vs MOSFET vs IGBT,” NEVSEMI, [Online]. Available: https://www.nevsemi.com/blog/difference-between-bjt-mosfet-and-igbt-bjt-vs-mosfet-vs-igbt.

[50] i2c-bus, “Bit Addressing,” [Online]. Available: https://www.i2c-bus.org/addressing/10-bit-addressing/.

[51] F. Souza, “Comunicação I2C,” Embarcados, [Online]. Available: https://embarcados.com.br/comunicacao-i2c/.

[52] “IL PROTOCOLLO SPI,” Università del Salento, [Online]. Available: https://www.unisalento.it/documents.

[53] P. Dhaker, “Introduction to SPI Interface,” Signal Integrity Journal, 18 September 2018. [Online]. Available: https://www.signalintegrityjournal.com/articles/967-introduction-to-spi-interface.

[54] APFEN, “Comprehensive Guide to Identifying USB Connector Port Types,” 6 September 2023. [Online]. Available: https://www.szapphone.com/blog/usb-connector-types-guide/.

[55] Elettronica Conduttori, “Data Cables, Cavo BUS USB,” [Online]. Available: https://www.elettronicaconduttori.com/prodotti/cavo-usb-automotive-per-sistema-infotainment-2/.

[56] H. Siebeneicher, “Universal Asynchronous Receiver-Transmitter (UART),” 28 10 2024. [Online]. Available: https://docs.arduino.cc/learn/communication/uart/#basic-print-example..

[57] A. Tanenbaum, Reti Di Calcolatori, Milano: Pearson, 2023.

[58] K. Yasar, “Transmission Control Protocol (TCP),” [Online]. Available: https://www.techtarget.com/searchnetworking/definition/TCP#:~:text=To%20prevent%20sending%20too%20much,their%20order%20and%20sequence%20number..

[59] W. Siringoringo, “Teaching and Learning Wi-Fi,” *Selected Readings on,* pp. 22-40.

[60] Wi-Fi Alliance, “Who We Are, History,” [Online]. Available: https://www.wi-fi.org/who-we-are/history.

[61] C. Doctorow, “WiFi isn't short for "Wireless Fidelity",” Boing Boing, [Online]. Available: https://boingboing.net/2005/11/08/wifi-isnt-short-for.html.

[62] Random Nerd Tutorials, “What is MQTT and How It Works,” 16 December 2021. [Online]. Available: https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/.

[63] Semtech, “A Brief History of LoRa®: Three Inventors Share Their Personal Story at The Things Conference,” 08 January 2020. [Online]. Available: https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference.

[64] The Thingd Network, “What are LoRa and LoRaWAN?,” [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/#lora.

[65] LoRa Alliance, “RP002-1.0.3 LoRaWAN® Regional Parameters,” [Online]. Available: https://lora-alliance.org/wp-content/uploads/2021/05/RP002-1.0.3-FINAL-1.pdf.

[66] Science Direct, “Spread Spectrum,” [Online]. Available: https://www.sciencedirect.com/topics/computer-science/spread-spectrum#:~:text=Spread%20Spectrum%20refers%20to%20a,required%20to%20transmit%20the%20message.

[67] Sghoslya, “All About LoRa and LoRaWAN,” [Online]. Available: https://www.sghoslya.com/.

[68] B. Reynders and S. Pollin, “Chirp Spread Spectrum as a Modulation Technique,” *2016 Symposium on Communications and Vehicular Technologies (SCVT),* Nov. 2016.

[69] “LoRa Physical Layer Packet Format,” The Things Network, [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/lora-phy-format/.

[70] V. T. e. al., “Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies,” *Lora Backscatter,* vol. 1, no. 3, pp. 1-24, Sep. 2017.

[71] M. L. Liya and M. Aswathy, “Lora Technology for internet of things(iot):a brief Survey,” *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC),* Oct. 2020.

[72] J. Santa, R. Sanchez-Iborra, P. Rodriguez-Rey, L. Bernal-Escobedo and A. F. Skarmeta, “LPWAN-based vehicular monitoring platform with a generic IP network interface,” *Sensors,* vol. 19, no. 2, p. 264, Jan. 2019.

[73] LoRa Alliance Technical Committee, “LoRaWAN® Backend Interfaces,” 19 10 2020. [Online]. Available: https://lora-alliance.org/wp-content/uploads/2020/11/TS002-1.1.0_LoRaWAN_Backend_Interfaces.pdf.

[74] LoRa Alliance Technical Committee, “LoRaWAN™ 1.1 Specification,” 11 10 2017. [Online]. Available: https://net868.ru/assets/pdf/LoRaWAN-v1.1.pdf.

[75] The Things Industries, “End Device Activation,” [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/end-device-activation/.

[76] The Things Industries, “Architecture,” [Online]. Available: https://www.thethingsindustries.com/docs/the-things-stack/concepts/architecture/.

[77] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” Internet Engineering Task Force (IETF) , August 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8446#page-6..

[78] M. Guay, “What are webhooks?,” Zapier, [Online]. Available: https://zapier.com/blog/what-are-webhooks/.

[79] akenza.io, “Overview,” [Online]. Available: https://docs.akenza.io/akenza.io. .

[80] akenza.io, “Data flows,” [Online]. Available: https://docs.akenza.io/akenza.io/get-started/your-data-flow. .

[81] akenza.io, “MQTT,” [Online]. Available: https://docs.akenza.io/akenza.io/api-reference/mqtt..

[82] akenza.io, “The things network,” [Online]. Available: https://docs.akenza.io/akenza.io/get-started/your-integration/the-things-network. .

[83] akenza.io, “Rule logic - comparison,” [Online]. Available: https://docs.akenza.io/akenza.io/get-started/rule-engine/rule-logic. .

[84] akenza.io, “Rules,” [Online]. Available: https://docs.akenza.io/akenza.io/get-started/rule-engine/rule-logic. .

[85] akenza.io, “Dashboard Builder,” [Online]. Available: https://docs.akenza.io/akenza.io/get-started/dashboard-builder. .

[86] myDevices, “Cayenne Low Power Payload,” [Online]. Available: https://docs.mydevices.com/docs/lorawan/cayenne-lpp.

[87] A. K. Bourke, J. V. O'Brien and G. M. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait &amp Posture,* vol. 26, no. 2, pp. 194-199, Jul. 2007.

[88] G. M. Azmal, A. Al-Jumaily and M. Al-Jaafreh, "Continuous Measurement of Oxygen Saturation Level using Photoplethysmography Signal," *2006 International Conference on Biomedical and Pharmaceutical Engineering,* pp. 504-507, 2006.

APPENDIX A – WEARABLE DEVICES HISTORY

In the 13th century the English friar Roger Bacon explained, in his Opus Majus, the scientific principles on which the corrective lenses are based and designed. These glasses were the first wearable glasses.

In the 16th century Peter Henlein made the Pomander watch (Figure 201), (also named PHN1505), which is considered the first pocket mechanical watch. This watch was very convenient, as it was portable, but it didn't have a great precision. Another Pomander watch can be observed in Figure 202.

Figure 201 – Peter Henlein's Pomander watch, The Watch 1505.

Figure 202 – Another Pomander watch (1530) created by Peter Henlein.

Regarding the smart rings, the first one was the Abacus Ring (Figure 203), which dates back to the early 17th century during the Qing Dynasty era and was a compact smart accessory designed to help traders. The standard abacus was made of 10 parallel wires located between two boards with nine beads on each of them. It

represents the first step towards the modern wearable computers and the modern smart rings.

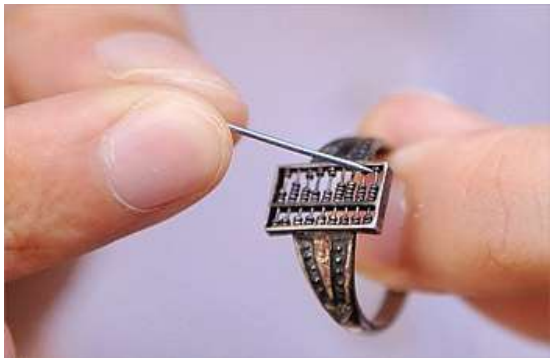

Figure 203 – A functioning Abacus Ring from the Qing Dynasty (1644-1912).

In 1937, during the World War I-II period, Donald Hings developed the "packset system", which later became known as "walkie-talkie" (Figure 204).

Figure 204 – Packset system.

Also, in the military area, wristwatches for the planning and coordination of various operations were developed, and the first wired hands-free devices were integrated with flight helmets and were destined for navy and pilots (Figure 205 and Figure 206).

Figure 205 – British Military Wristwatch.

Figure 206 – Original British WWII RAF 2nd Pattern Type C Leather Flying Helmet wit.

In 1960, after the World War II, in the VR field, Morton Heilig created the "Stereophonic Television Head-Mounted Display", which was the first VR simulator to be equipped with binocular display, vibrating seat, stereophonic speakers, cold air blower and odours generator. Subsequently, "Sensorama Simulator" was developed as its upgraded version (Figure 207).

Figure 207 – Sensorama Simulator.

The first wearable computer hidden in a shoe dates back to 1961, when the MIT researchers Edward O. Thorp and Claude Shannon developed a timing device in a shoe that was able to predict the ball's landing place on a roulette table (Figure 208).

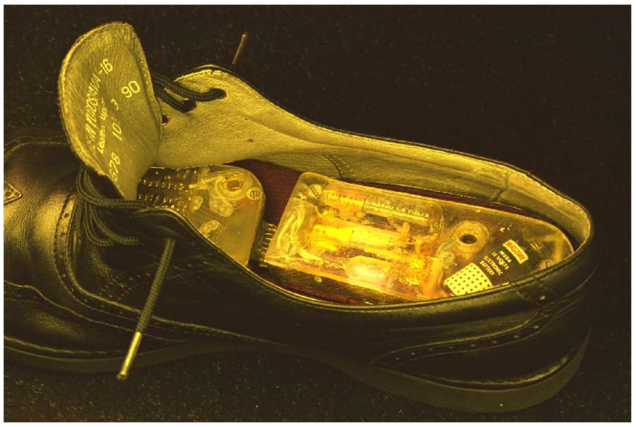

Figure 208 – Computer hidden in a shoe.

A few years later, Hugo Gernsback invented the TV glasses (Figure 209).

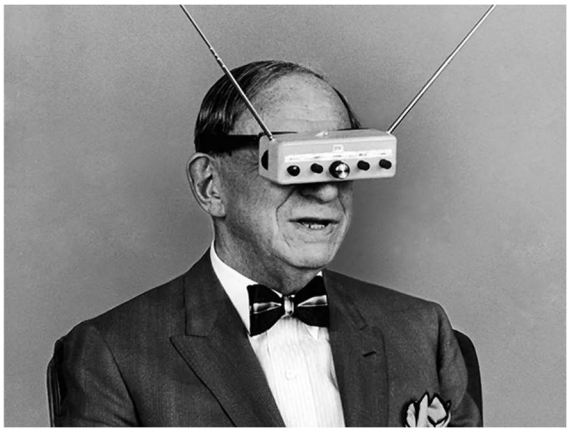

Figure 209 – TV Glasses.

The year 1975 was marked by the invention of the Pulsar Calculator Watch (Figure 210), and then, in 1977, Hewlett Packard developed the HP-01 (Figure 211),which was the first algebraic calculator watch, while cheaper versions were subsequently made by CASIO.

Figure 210 – Pulsar Calculator Watch.

Figure 211 – Hewlett Packard HP-01.

In 1979, the Portable Stereo Sony Walkman was released and it was the first commercially available portable personal stereo cassette player with headphones (Figure 212).

Figure 212 – Portable Stereo Sony Walkman.

In 1981, with his EyeTap project, Steve Mann developed the first backpack-like computer that had the capability to process the data from a next-to-eye-mounted camera and to show it on the screen in front of an eye. This device marked the first step toward modern AR glasses and is considered the ancestor of Google Glasses. Figure 213 presents the evolution of the WearComp wearable computer from backpack-based systems of the 1980s to his most advanced version of late 90s.

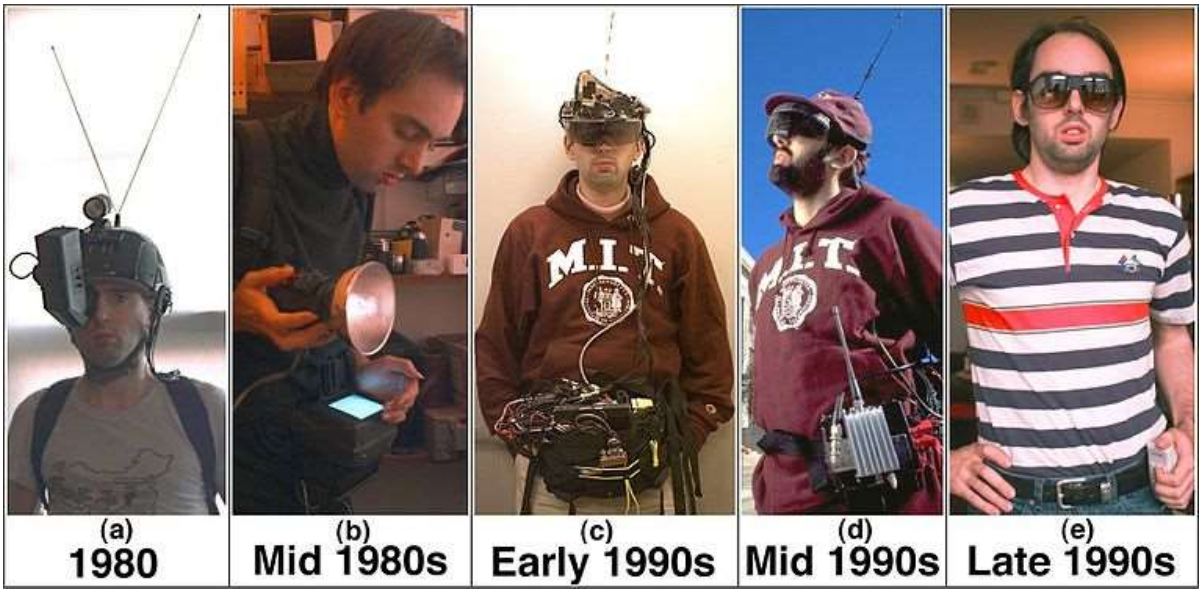

Figure 213 – Evolution of Steve Mann's WearComp wearable computer.

Then, in 1981, Seiko's UC 2000 WRIST PC was the first wearable computer and it was equipped with 2 kB of storage and with the possibility to spell the time and calculation (Figure 214).

Figure 214 – Seiko's UC 2000 with its keyboard.

In 1981 Nelsonic Space Attacker Watch, thanks to its two buttons, allowed playing popular arcade games anywhere and anytime (Figure 215).

Figure 215 – Nelsonic Space Attacker Watch.

The year 1989 saw the development of the Private Eye head-mounted display, which had a monochrome monitor with a 720 × 280 resolution and was sold by Reflection Technology. It is considered to be the second ancestor of the Google Glasses (Figure 216).

Figure 216 – Private Eye head-mounted display.

In 1990 Olivetti Research made the Active Badge, which was the first portable indoor location tracker and was able to transmit unique Identifiable Infrared signals to communicate a person's location (Figure 217).

Figure 217 – David Greaves' Active Badge (Cambridge University).

In 1994 Mik Lamming and Mike Flynn developed "Forget-Me-Not" (Figure 218), a continuous personal recording system that recorded the interaction with people and stored this information in a database for future use.

Figure 218 – Forget-Me-Not.

In 1996, Palm launched PalmPilot 1000, a one-chip computer which was the first mass-produced personal digital assistant (PDA) (Figure 219).

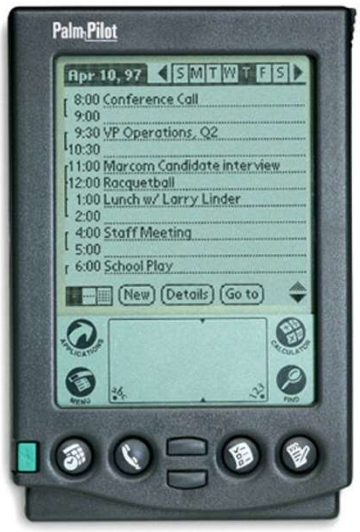

Figure 219 – PalmPilot 1000.

The mBracelet was a wrist-wearable computer designed for financial transactions with Automated Teller Machine (ATM).

Massimo Osti in collaboration with Philips designed Levi's Industrial Clothing Division Jacket, which had an internal network aimed at interconnecting electronic gadgets (Figure 220).

Figure 220 – Levi's Industrial Clothing Division Jacket.

In 2002 Nick Woodman founded the company that subsequently, in 2004, developed the first model of GoPro camera, that was small, light, waterproof and AAA battery-powered (Figure 221).

Figure 221 – First model of GoPro camera.

In 2006, "Nike + iPod Sport Kit" was released which, thanks to its small accelerometer installed or already built into shoes, was able to measure and record distance travelled and pace (Figure 222).

Figure 222 – Nike+ iPod transmitter in Nike+ Shoe.

The year 2007 saw the foundation of Fitbit, whose Fitbit Classic was the first wireless activity tracker able to synchronise data with the Internet, so that the same data could be available on both the Fitbit and a mobile phone (Figure 223).

Figure 223 – Fitbit Classic.

In 2009, Samsung S9110 Smart Watch was the first smartwatch to be equipped with full-colour touchscreen, Bluetooth connectivity, music player and voice recognition (Figure 224).

Figure 224 – Samsung S9110 Smart Watch.

In 2012, Google Plus social network information spread the first news on Project Glass, whose aim was to build a portable computer that will help "explore and share the world", thus marking the beginning of VR and AR mass adoption (Figure 225).

Figure 225 – Google Plus Project Glass.

Lastly, in 2014 there was a great spreading of personal activity trackers, among them the basis which was the most advanced device, since it was able to collect data such as heart rate, calorie consumption by activity, several sleep stages, sweating and skin temperature (Figure 226).

Figure 226 – Basis personal activity tracker.

APPENDIX B – NTC DATASHEET

MA300

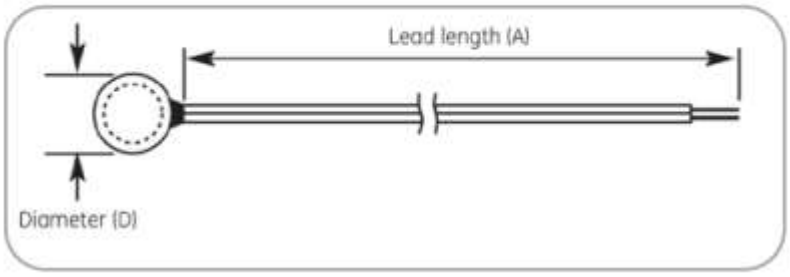

Figure 227 – MA300 dimensions.

Assembly Type	MA100	MA200	MA300
Standard Diameters (D)	0.030 in (0.762 mm) 0.050 in (1.27 mm) 0.070 in (1.78 mm) 0.080 in (2.03 mm)	0.156 in (3.96 mm)	0.375 in (9.52 mm)
Body Length (L)	3/8 in (9.52 mm)	3.75 in (95.25 mm)	N/A
Standard Lead Length (A)	24 in (610mm) GG: 36 in (914mm)	2 in (51 mm)	24 in (610mm)
Tolerance	See Tolerance Code and Temperature Table	See Tolerance Code and Temperature Table	See Tolerance Code and Temperature Table
Wire Gauge	28, 30, 32, 38, AWG	30 AWG	30 AWG
Standard Wire Insulation	*Heavy Isomid *Medical Grade PVC *Polyurethane with Nylon Overcoat	PTFE	Medical Grade PVC PTFE
Body Material	Molded Plastic or Kapton Sleeve	Lexan Shaft Aluminum Tip	Stainless Steel
Nominal R Values @ 77°F (25°C)		2252 Ω, 3000 Ω, 5000 Ω, 10,000 Ω (for all three types)	

Figure 228 – Features of the MA thermistors.

Resistance vs Temperature

Temperature °F (°C)	2252 Ω	3 kΩ	5 kΩ	10 kΩ	103(Y) Ω
32.00 (0)	7373.00	9821.93	16369.9	32739.8	29565.8
33.80 (1)	7005.80	9332.76	15554.6	31109.2	28224.8
35.60 (2)	6659.04	8870.84	14784.7	29569.5	26951.9
37.40 (3)	6331.49	8434.49	14057.5	28115.0	25743.3
39.20 (4)	6021.97	8022.17	13370.3	26740.6	24595.5
41.00 (5)	5729.40	7632.41	12720.7	25441.4	23505.0
42.80 (6)	5452.74	7263.87	12106.4	24212.9	22468.7
44.60 (7)	5191.06	6915.27	11525.4	23050.9	21483.8
46.40 (8)	4943.46	6585.43	10975.7	21951.4	20547.2
48.20 (9)	4709.11	6273.23	10455.4	20910.8	19656.6
50.00 (10)	4487.22	5977.64	9962.74	19925.5	18809.3
51.80 (11)	4277.07	5697.69	9496.15	18992.3	18003.1
53.60 (12)	4077.97	5432.47	9054.11	18108.2	17235.7
55.40 (13)	3889.29	5181.11	8635.19	17270.4	16505.1
57.20 (14)	3710.42	4942.83	8238.05	16476.1	15809.4
59.00 (15)	3540.80	4716.88	7861.46	15722.9	15146.7
60.80 (16)	3379.91	4502.54	7504.24	15008.5	14515.3
62.60 (17)	3227.24	4299.17	7165.28	14330.6	13913.6
64.40 (18)	3082.34	4106.14	6843.57	13687.1	13340.0
66.20 (19)	2944.77	3922.88	6538.13	13076.3	12793.0
68.00 (20)	2814.12	3748.83	6248.05	12496.1	12271.4
69.80 (21)	2690.01	3583.49	5972.48	11945.0	11773.8
71.60 (22)	2572.07	3426.38	5710.63	11421.3	11299.0
73.40 (23)	2459.96	3277.03	5461.72	10923.4	10845.8
75.20 (24)	2353.37	3135.04	5225.07	10450.1	10413.1
77.00 (25)	2252.00	3000.00	5000.00	10000.0	10000.0
78.80 (26)	2155.56	2871.53	4785.88	9571.77	9605.42
80.60 (27)	2063.79	2749.28	4582.13	9164.26	9228.45
82.40 (28)	1976.44	2632.91	4388.19	8776.38	8868.24
84.20 (29)	1893.27	2522.12	4203.54	8407.07	8523.96
86.00 (30)	1814.07	2416.61	4027.68	8055.35	8194.82
87.80 (31)	1738.61	2316.09	3860.15	7720.30	7880.09
89.60 (32)	1666.71	2220.31	3700.51	7401.03	7579.07
91.40 (33)	1598.18	2129.02	3548.36	7096.72	7291.10
93.20 (34)	1532.85	2041.98	3403.30	6806.60	7015.55
95.00 (35)	1470.54	1958.98	3264.97	6529.94	6751.83
96.80 (36)	1411.11	1879.81	3133.02	6266.03	6499.37
98.60 (37)	1354.41	1804.27	3007.12	6014.23	6257.64
100.40 (38)	1300.29	1732.18	2886.96	5773.93	6026.13
102.20 (39)	1248.63	1663.36	2772.27	5544.53	5804.36
104.00 (40)	1199.30	1597.65	2662.75	5325.50	5591.88
105.80 (41)	1152.19	1534.89	2558.15	5116.30	5388.26
107.60 (42)	1107.19	1474.94	2458.23	4916.46	5193.09
109.40 (43)	1064.18	1417.65	2362.75	4725.49	5005.96
111.20 (44)	1023.08	1362.89	2271.49	4542.98	4826.53
113.00 (45)	983.78	1310.55	2184.24	4368.49	4654.43
114.80 (46)	946.21	1260.49	2100.82	4201.64	4489.33
116.60 (47)	910.27	1212.62	2021.03	4042.05	4330.91
118.40 (48)	875.89	1166.81	1944.69	3889.38	4178.88
120.20 (49)	842.99	1122.99	1871.65	3743.29	4032.94
122.00 (50)	811.50	1081.04	1801.73	3603.46	3892.83

Figure 229 – Resistance versus Temperature for the MA thermistors.

Instituto Superior
de Engenharia
Politécnico de Coimbra