University of Alcalá

Polytechnic School

**Degree in Engineering in Industrial Electronics and Automation**

**Degree Final Project**

# Physical activity measurement and indoor location for the assessment of daily routines in older adults



**Author:** Andrea Jiménez Martín

**Supervisor:** Juan Jesús García Domínguez

**Co-supervisor:** Sergio Lluva Plaza

Septiembre 2022

UNIVERSITY OF ALCALÁ
Polytechnic School

**Degree in Engineering in Industrial Electronics and Automation**

Degree Final Project

# Physical activity measurement and indoor location for the assessment of daily routines in older adults

**Author:** Andrea Jiménez Martín

**Supervisor:** Juan Jesús García Domínguez

**Co-supervisor:** Sergio Lluva Plaza

**Examination board:**

**Chairperson:** D. Francisco Javier Dongil Moreno

**Assistant No.1:** D. Miguel Ángel García Garrido

**Assistant No.2:** D. Juan Jesús García Domínguez

**DATE**: September 2022

# RESUMEN

Este trabajo presenta una propuesta para la monitorización de adultos mayores con el fin de inferir las actividades de la vida diaria (ADLs), e identificar desviaciones en sus rutinas que podrían necesitar alguna clase de intervención. Esta monitorización se consigue analizando el tiempo que pasan en cada habitación de su lugar de residencia, el cual puede ser estimado con balizas basadas en tecnología BLE (Bluetooth Low Energy). Las balizas receptoras de BLE desplegadas en el entorno detectan la señal del dispositivo emisor que porta el usuario. La localización de la persona se realiza a través de algunos métodos de fingerprinting, procesando la intensidad de la señal recibida.

# SUMMARY

This paper presents a proposal for monitoring older adults in order to infer activities of daily living (ADLs), and identify deviations in their routines that might need some kind of intervention. This monitoring is achieved by analysing the time spent in each room of their place of residence, which can be estimated with beacons based on BLE (Bluetooth Low Energy) technology. The BLE receiving beacons deployed in the environment detect the signal of the transmitting device carried by the user. The location of the person is done through some fingerprinting methods by processing the received signal strength.

# KEYWORDS

BLE, routine monitoring, older adults, fingerprinting

# ACRONYMS

BLE (Bluetooth Low Energy)

RSSI (Receiver Signal Strength Indicator)

MAC (Media Access Control)

ML (Machine Learning)

AI (Artificial Intelligence)

ANN (Artificial Neural Network)

DT (Decision Tree)

RF (Random Forest)

SVM (Support Vector Machine)

MMH (Maximum Marginal Hyperplane)

ART (Adaptive Resonance Theory)

KNN (K-Nearest Neighbours)

# General index

# Figures index

# Tables index

# 1. Introduction

## 1.1 Motivation

Due to the advances in technology and quality of life, people all around the world are living longer, although mainly in developed countries, what means that older population is growing in every country. According to the WHO (World Health Organization), by 2030, 1.4 billion people will be aged 60 or over, which corresponds to 1 in 6 people being this age. Additionally, by 2050, it is expected that this number will double, having 2.1 billion citizens with 60 years or more. From these elderly people, most of them prefer to spend their last years at their own houses, rather than going to a nursing home, and almost a third of them live alone. This issue needs the help of the technological advances, which can lead to monitoring solutions that can assist seniors in managing themselves independently without having to give up on their safety or quality of life.

It is known that elderly people have rather routine lives, which allows us to monitor their activities, and identify whether their physical or cognitive state deteriorates in any way. The problem we sometimes encounter with monitoring people is that it can invade their privacy. However, the aim of this project is to follow the routines of these individuals in an unobtrusive and non-invasive way. In particular, what we intend to do in this study is to monitor the activities of the daily living (ADLs) of the inhabitants and identify deviations in their routines that may need some kind of intervention.

Nowadays, behavioural monitoring of people and robots has become a popular research field inside localization and positioning systems. Monitoring is used in several fields, such as energy efficiency or health. Some examples of this are: optimizing productivity by analysing work patterns, reducing energy consumption, or spotting behavioural deviations in behavioural routines of elderly people, between many others.

In order to carry out this monitoring, the most widely used method is indoor positioning systems, being radiofrequency (RF) one of the main base technologies. With this base technology, we can obtain positioning precisions at room level (using WiFi or Bluetooth) until centimetres (using Ultrawideband or UWB), being the first mentioned technologies much cheaper than the last one. Since we want to make a cost-efficient system for our project, we will use BLE, a technology included inside Bluetooth. Additionally, in our case, we will include machine learning techniques in order to obtain information about the routines of elderly people and be able to detect possible variations.

## 1.2 Objectives

The main objective of this project is the monitoring of elderly people in an indoor environment. To do so, we will detect and evaluate their daily routines with BLE-based indoor beacons and the KNN ML method. To fulfil these aims, we have to achieve several partial objectives:

- Obtain the contextual localization of the person with BLE technology.
- Obtain their routines from the localization results acquired previously, with the application of KNN (K-Nearest Neighbours), a machine learning (ML) method.

## 1.3 State of Art

In [1], researchers monitored seniors in their nursing homes. They found that the FBMS (Fixed-Beacon and Movable-Scanner) architecture could not be adopted, since elderly people did not carry smartphones inside the residence. Therefore, they had to use the MBFS (Movable-Beacon and Fixed-Scanner) architecture, so that the individuals just have to carry a BLE beacon. Their work consists in the monitoring of seniors in a closed environment (their nursing home) by associating each room with an activity. With this, they do not need the exact position of the person, but only their approximate location to identify the room that person is in at each moment. This localization is made through RSSI (Received Signal Strength Indicator). It also meets the requirements of privacy, as this system does not incorporate cameras or microphones.

The work [2] shows the monitoring of older adults who live alone in their homes. They carry smartphones, with which researchers can get samples of the localization of the senior at every moment. With these data, scientists can predict mild cognitive impairments. First, they tried Wi-Fi fingerprinting techniques, but there was no way of knowing if the results were correct, since the home map had to be done by the users. Then, they tried to solve these problems by using BLE-based methods, which showed much better results with respect to monitoring.

In [3], researchers use the accelerometer and gyroscope of a smartphone to monitor seniors. Their objective is to present a deep learning model that can learn local features and model the time dependence between them. To obtain these routines, they use a combination of two machine-learning techniques (CNN-LSTM). This combination must be done, since CNN (Convolution Neural Network) cannot be used solely, because this machine-learning technique has a good performance in extracting local features, but not in obtaining the relationships between time and features. Therefore, they had to add the LSTM (Long Short-Term Memory) machine-learning technique, which is a kind of recursive network that contains a memory to simulate a time dependent sequence problem. Despite the deficiencies that each machine-learning technique has if used independently, the mixture of both of them can accurately identify the basic and transitional features of activities.

In study [4], scientists aim to develop a BLE-based new approach for indoor positioning, avoiding the user of multilateration, trilateration and Least Square Estimation (LSE), for performing the positioning, since they have large prediction errors. The main signal transmitters they used were BLE beacons, and they applied several machine-learning regression models to predict user location. With these, they expect to increase the accuracy of the predictions. With BLE, fingerprinting and machine-learning regression models, researchers could create a positioning algorithm. They used 4 machine-learning regression models: Artificial Neural Network Regression (ANN), Multiple Linear Regression (MLR), Random Forest Regression (RF) and Support Vector Regression (SVR). Then, they compared each of these techniques and the fusion of all of them.

As can be derived from the state of the art, BLE is consolidating as an interesting technology for providing room-level indoor positioning.

# 2. Materials

In the following sections, we will describe the sensory system we will use. Specifically, we will work with the SensorTile.box and several Raspberry Pi 3 model b+.

## 2.1. SensorTile.box

Sensors are devices used in almost every microcontroller-based applications, which make them highly important elements in the technology world.

Normally, the designers choose the sensors they want to work with, and then they connect them to a processor. Most of these sensors can be directly connected to the processors, although there exist some sensors that need additional circuitry, such as the connection between an analogue sensor to a digital processor (ADC converter).

Additionally, the designers have to take into account the input and output voltages of the sensor and the processor. Nowadays, processors work at 3.3 V, but some sensors require 5 V. This causes an incompatibility between both devices, which needs to be fixed with other types of converters that can adapt this voltage difference.

Another problem that arises when working with external sensors is that they are quite complex. This implies that some additional software libraries are needed to manage them properly through a programmable system. Although there are cases in which these libraries can be found freely, there are other times in which the developers need to develop all these special libraries in order to work with the sensor and the processor.

The sensor we are going to use in this project is called SensorTile.box. It is a small multi-sensor circuit board inside a plastic case. It has been developed by the company STMicroelectronics. The circuit board includes a high performance 32-bit ARM Cortex-M4 processor and several sensors, which will be described in the following sections. Some of these sensors are: accelerometer, gyroscope, temperature sensor, humidity sensor, atmospheric pressure sensor… Additionally, it has wireless IoT and Bluetooth connectivity, both of which can be easily used with an application that can be installed in any compatible iOS or Android smartphone.

In order to turn the SensorTile.box, all the user has to do is to connect its long-life battery to the circuit. For example, the user can place the SensorTile.box in a room, and they can measure its temperature, humidity and pressure, which does not involve any hardware modifications or any programming. All the user has to do is to push a few buttons on the app, and the data will be collected in a spreadsheet, so that the user can show it in a list, plot it, make a graph with it, or whatever they wish to do with it.

SensorTile.box can be managed through a smartphone app called STE BLE by all users, independent on their level of expertise, since it can work in three modes: basic mode, expert mode and pro mode.

- **Entry level mode**. It is the most basic mode, which allows the users to handle the different functionalities of the SensorTile.box through some pre-developed examples (demo apps charged in the STE BLE app). This can be done with the STE BLE application, which can be easily installed in any Android or iOS mobile devices. In this mode, the user does not need any programming knowledge since the examples are already developed and ready to use though some simple buttons. All the user has to do is to choose the parameters they want to measure, and the measurements will show up in the screen directly. Then, they can choose whether to only display or also plot them.
- **Expert level mode**. It is the intermediate mode, in which users can build their personal simple programs, with the help of a graphical wizard. Although the users need to have certain programming skills, they do not need any knowledge in C programming, since it is not required in this mode.
- **Pro level mode**. It is the most complicated mode, and users need a solid programming background in order to use it. They open the box and connect the processor to STM32 programmer, which enables them to build their own applications and projects, which can be as complicated as the user desires, by writing programming code. After they have developed their programs, they can upload them to the STE BLE app.

If we want any microcontroller to work properly, it needs to communicate with the outside, and this is done though the reading of one or several input signals from a switch, a keyboard, a sensor or any other input device. The microcontroller processes all these input signals and generates one or several output signals, which can be lights, sounds, displays…

Transducers are essential devices used in microcontrollers, since they are widely present in the vast majority of applications that use microcontrollers. When we talk about transducers, we refer to sensors and actuators.

- Sensor. It is a device that generates an electrical signal when it measures some variable. Normally, this electrical signal is linearly proportional to the measured variable. For example, a temperature signal produces an electrical signal proportional to the temperature it detects, so that we can easily find out the temperature by analysing the value of this electrical signal.
- Actuator. It is a device, normally mechanical, which creates movement when triggered by an electrical signal. An example of an actuator is a motor.

The majority of real sensors are analogue, which means that these sensors generate analogue output voltages or currents proportional to the variables it measures or following a specific relationship. Since these analogue sensors cannot be directly connected to the digital microprocessors, we have to use ADC (Analogue-to-Digital Converter) modules. The resolution of the ADC depends on its data width, which go from 10 to 16 bits nowadays.

Although they are the minority, we also have digital sensors, whose advantage is they can be directly connected to the digital processor, since we have the same kind of data at the sensor output and at the processor input (digital data).

The use of external sensors can complicate the design, both hardware and software, for several reasons, which will be mentioned below:

- As most external sensors are analogue, we need ADC modules to connect them to the processor, which add complexity to the circuit. however, as these analogue external sensors are so common nowadays, most processors include built-in ADCs.

- The voltage levels may not be compatible between the output of the sensor and the input of the processor. This will imply additional circuitry, such as step-up or step-down circuits.
- As we mentioned in previous paragraphs, some of these external sensors are quite complex. This means that we will have to use special libraries for their programming. If these specialised libraries are available, we can use them directly. However, there can be the case that these libraries have to be developed by the user, which is more time-consuming and more complex.

In regard to the SensorTile.box, which is the device we are going to use in this project, we have some advantages. The first one is that all its sensors have already been connected to the processor. Additionally, we do not have to develop any libraries to access its sensors, since they have already been created and they are available for the user.

The SensorTile.box, whose actual product code is STEVAL-MKSBOX1V1, is a ready-to-use wireless IoT and wearable sensor evaluation and development kit, which was manufactured by STMicroelectronics. It was first demonstrated in 2019, at the IoT World Conference in California.

An early version of it was worn by Luca Colli, and it measured and stored his movements and general performance while he climbed Mount Everest. Nowadays, the SensorTile.box is a relatively small, flexible, low power, highly accurate, easy to use and easy to configure device. It can be used in a wide range of sensing, tracking and monitoring applications.

In the following sections, we will take a look at the general characteristics of the box, and the specific features of each of its sensors.

The SensorTile.box consists in a programming board, which is enclosed in a blue plastic box.

*Figure 1. The SensorTile.box*

This device is a Plug&Play module, which can be connected to a smartphone through BLE (Bluetooth Low Energy) technology. With it, users can observe, calculate and plot the information collected by all its sensors. This information can be useful for people monitoring, such as step counting, trajectory, speed and distance covered. It can also measure environmental monitoring, such as temperature, humidity, pressure and altitude.

The SensorTile.box has a rechargeable battery, which does not come connected to the processor to avoid breaking during transportation. We already have it connected to our SensorTile.box.

The SensorTile.box has a micro-B USB connector on its side, as we can see in Figure 2, which is used to charge the battery in real time and, simultaneously, communicate with the processor board.

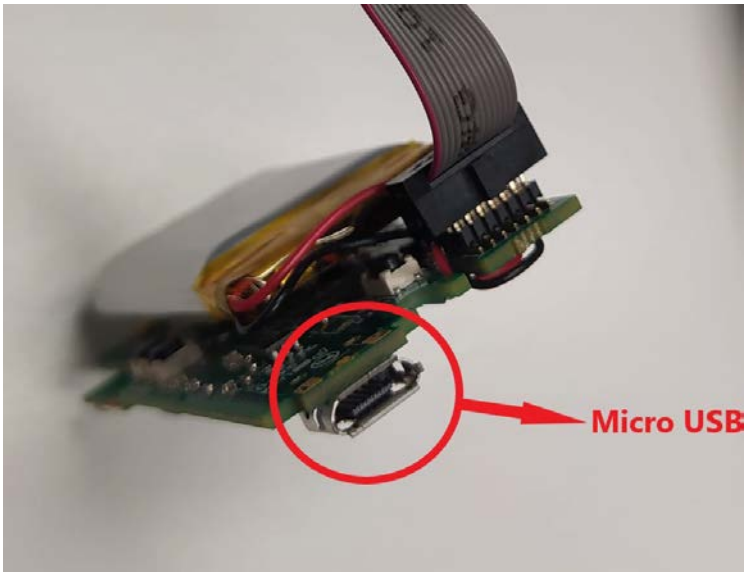


*Figure 2. Micro USB connector*

The box has 3 LEDs, and each one is used for a different purpose:

- Blue LED. It flashes once every 10 seconds to show that a correct Bluetooth connection has been made and it is available.
- Red LED. It flashes when the battery is charging. The blinking pace indicates the battery condition.
- Green LED. It turns on when the user is carrying specific functions or when the SensorTile.box firmware is being updated.

The SensorTile.box has several low power and high precision MEMS sensors, which are shown in Figure 3. The list of all SensorTile.box features is the following:

- STM32L4R9ZIJ6 ultra-low-power, 120 MHz ARM Cortex-M4 microcontroller with DSP and FPU and 2048 KB flash memory
- LQFP144 package
- 16 MHz crystal oscillator, 32 kHz crystal oscillator for RTC
- 3 x SPI bus, 3 x I2C bus
- MicroSD slot
- 3 pushbuttons (BOOT, PWR, USER)
- Digital temperature sensor (STTS751)
- 6-axis inertial measurement unit (LSM6DSOX)
- 3-axis accelerometer (LIS2DW12 and LIS3DHH)
- 3-axis magnetometer (LIS2MDL)
- Altimeter/pressure sensor (LPS22HH)
- Microphone/audio sensor (MP23ABS1)
- Humidity sensor (HTS221)
- Bluetooth connectivity (SPBTLE-1S)
- HCP602535ZC 500mAh, 3.7 V Li-Ion chargeable battery
- STBC02AJR Li-Ion linear battery charger
- STBB3JR 2 MHz DC-DC converter
- FTSH107 connector for SWD debugging and UART Tx/Rx Programming and debugging interface (for professional development)
- 57 mm x 38 mm x20 mm IP54 plastic container

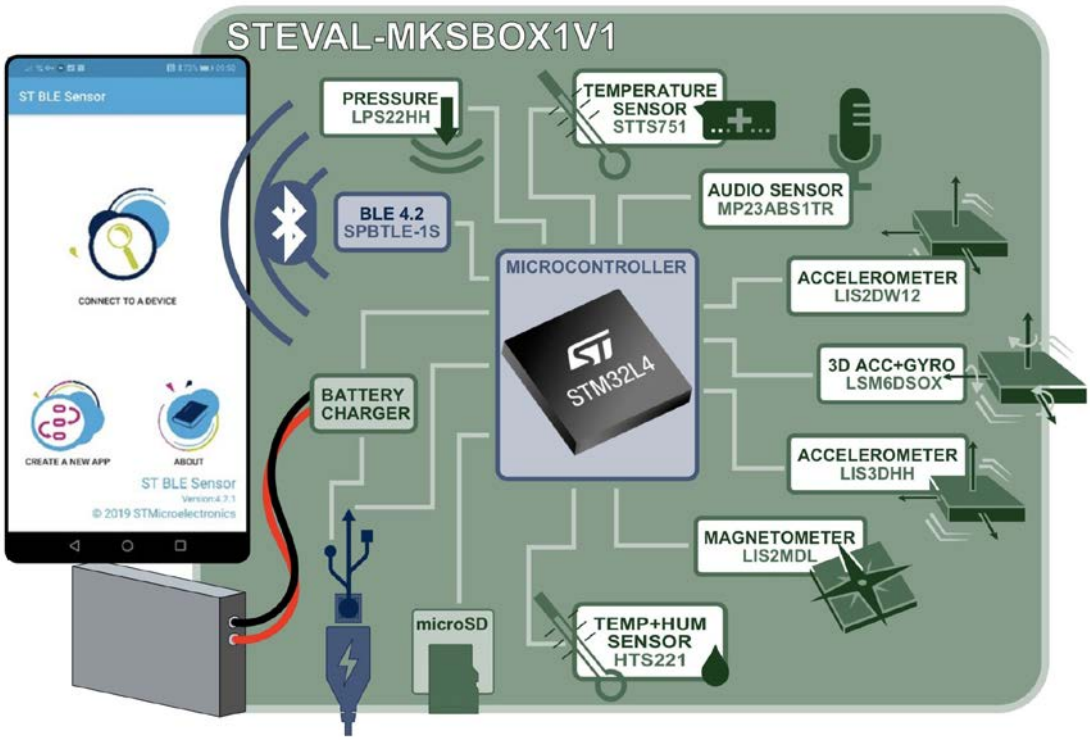


*Figure 3. Block diagram of the SensorTile.box*

In the next subsections, we will see the characteristics of each component in more detail.

### 2.1.1. The System clock

It can be driven by an internal or external oscillator, or by a main PLL clock. By default, the SensorTile.box is driven by the PLL clock at 80 MHz, which in turn is directed by the 16 MHz external crystal oscillator. Although these are the normal working conditions of the System clock, it can be boosted up to 120 MHz.

### 2.1.2. Pushbuttons

The SensorTile.box has 3 pushbuttons, as shown in Figure 4, whose functions are the following:

- BOOT. It is used to get the SensorTile.box into the DFU (Device Firmware Update) programming and into the debugging mode.
- PWR. It is used to turn the device on or off when the battery is connected.
- USER. It is a general-purpose user button.

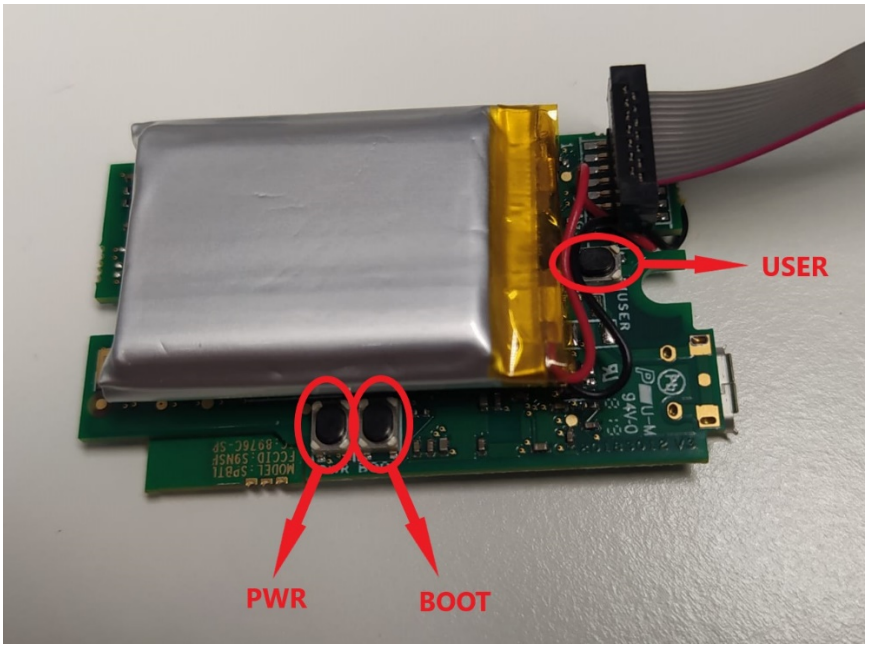


*Figure 4. Pushbuttons on the SensorTile.box*

All sensors on the SensorTile.box are identified in Figure 5, and they will be described in next sections.

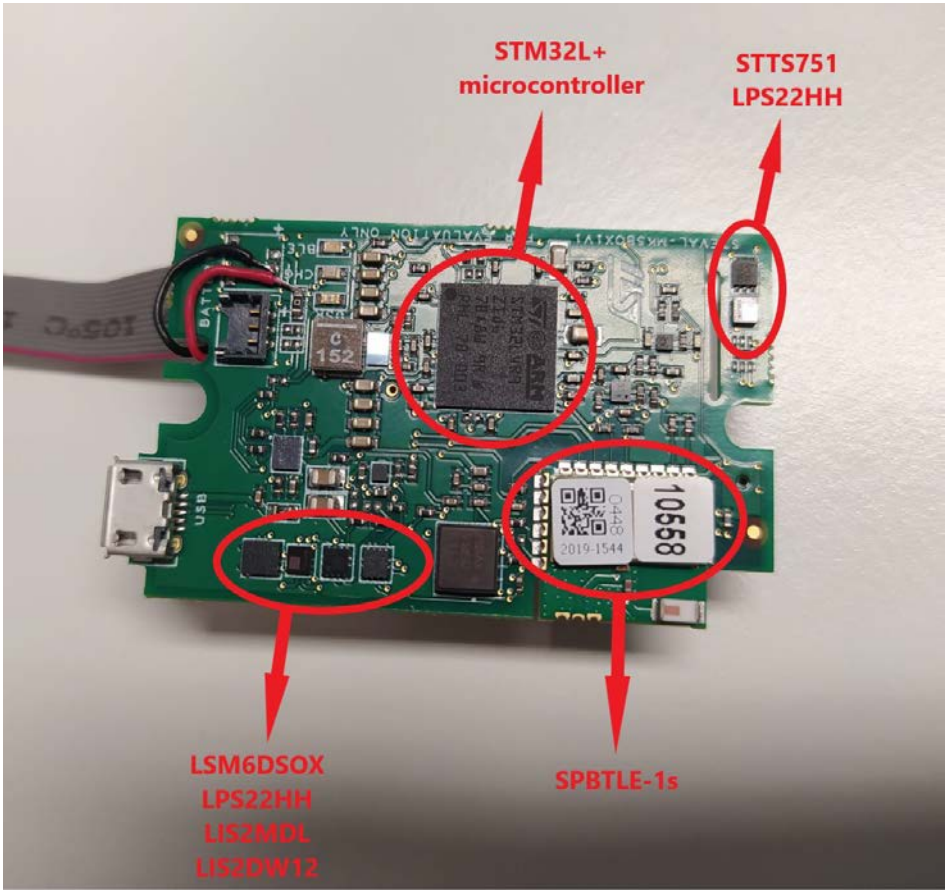


*Figure 5. Sensors on the SensorTile.box*

## 2.1.3. The STM32L4R9ZIJ6 microcontroller

It is a 32-bit powerful ARM processor at the heart of the SensorTile.box. It has the following features:

- 32-bit Cortex-M4 with FPU, adaptive real-time accelerator allowing 0-wait state execution, MPU and DSP instructions
- 2 MB flash memory
- 1.71 to 3.6 V operating range
- Up to 120 MHz operating frequency
- 125 nA standby mode power consumption
- 43 $\mu$ A/MHz run mode power consumption
- 1.25 DMIPS/MHz (Drystone 2.1)
- 4 to 48 MHz crystal oscillator
- RTC with calendar and alarm
- 24 capacitive sensing channels
- Graphics interface
- 16 timers
- 136 input/output ports
- 12-bit ADC

- 2x 12-bit DAC
- 2x operational amplifiers
- 2x comparators
- 20x communication interfaces (2x serial audio, 4x I2C, 6x UART, 3x SPI, CAN)
- 14-channel DMA controller
- 8-14 bit camera interface (black & white and colour)

### 2.1.4. Digital temperature sensor (STTS751)

The STTS751 is a 6-pin ultra-low current digital temperature sensor chip, which communicates over the 2-wire SMBus, designed by STMicroelectronics. It measures temperature in a very accurate way between 9 bits (step size of 0.5 °C) and 12 bits (step size of 0.00625 °C), being 10 bits (step size of 0.25 °C and conversion time of 21 ms) the default configuration.

Additionally, we can program it so that an alarm goes off if the measured temperature goes above a certain limit established by the user.

The basic characteristics of the STTS751 digital temperature sensor are the following:

- Operating voltage 2.25 V to 3.6 V
- Operating temperature -40 °C to +125 °C
- Up to 12-bit resolution with 0.0625 °C/LSB
- ±0.5 °C accuracy
- 3 $\mu$ A standby current
- 20 $\mu$ A supply current (for 1 conversion/sec)
- 21 ms conversion time (at 10-bit resolution))

### 2.1.5. 6-axis inertial measurement unit (LSM6DSOX)

It is a low-power inertial module that features a 3D digital accelerometer and a 3D digital gyroscope. It was designed by STMicroelectronics.

Its basic characteristics are listed below:

- Operating voltage 1.71 V to 3.6 V
- 9 Kbyte FIFO
- Very small footprint (2.5 mm x 3 mm x 0.83 mm)
- 0.55 mA current consumption
- ±2/±4/±8/±16 g full-scale acceleration range
- ±125/±250/±500/±100/±2000 dps full-scale acceleration range
- SPI / I2C & MIPI I3CSM serial interface compatible
- Motion detection, tilt detection, pedometer, step detector and step counter
- Embedded temperature sensor
- Interrupts for free-fall, wakeup, 6D/4D orientation
- Machine learning core

### 2.1.6. 3-axis accelerometer (LIS2DW12 and LIS3DHH)

The LIS3DHH is an ultra-high resolution 3-axis linear accelerometer, designed by STMicroelectronics. The measured acceleration is available through a standard SPIwire digital interface.

Its basic characteristics are the following:

- 16-bit data output
- 3-axis ±2.5 g full-scale acceleration range
- SPI interface
- Embedded 12-bit temperature sensor
- Embedded 32 level FIFO
- High stability over temperature (< 0.4 mg/°C) and time
- Operating temperature -40 °C to +85 °C

The LIS2DW12 is an ultra-low power MEMS digital motion sensor chip, which is a high performance 3-axis linear accelerometer chip, designed by STMicroelectronics. It can detect free-fall, wake-up, single and double tap, and activity and inactivity. Additionally, it has motion detection, portrait detection, landscape detection, and 6D/4D orientation.

Its basic features are the following:

- Operating voltage 1.62 V to 3.6 V
- Ultra-low power consumption (< 1 $\mu$ A in active mode, 50 nA in power-down mode)
- ±2/±4/±8/±16 g full-scale acceleration range
- Output data rates from 1.6 Hz to 1600 Hz
- I2C/SPI digital interface to the host processor
- 16-bit data output
- Embedded temperature sensor
- 32-level FIFO buffer to store local data
- Single data conversion (on demand)

### 2.1.7. 3-axis magnetometer (LIS2MDL)

It is an ultra-low power 3-axis digital output magnetometer, which can be used to detect magnetic fields. We can configure this sensor to produce an interrupt when a magnetic field is detected.

The basic characteristics of this sensor are the following:

- Operating voltage 1.71 V to 3.6 V
- Current consumption 200 $\mu$ A in high-resolution mode, 50 $\mu$ A in low-power mode
- ±50 gauss magnetic dynamic range with 3 magnetic field channels
- 15 mgauss to 500 mgauss self-test range
- Operating temperature ‑40 °C to +85 °C
- 16-bits output data

- I2C serial interface compatible

### 2.1.8. Altimeter/pressure sensor (LPS22HH)

It is a low-power MEMS absolute pressure sensor chip, which can be used as a digital output barometer, designed by STMicroelectronics. We can read data from the sensor though an interrupt service routine.

The basic features of this device are the following:

- Operating voltage 1.7 V to 3.6 V
- 260 hPa to 1260 hPa absolute pressure range
- 0.5 hPa absolute pressure accuracy with embedded temperature compensation
- Current consumption < 4 $\mu$ A
- 24-bit pressure data output
- Interrupt functions: Data-Ready, FIFO flags, pressure thresholds
- I2C, MIPI, I3CSM or SPI serial interface compatible

### 2.1.9. Microphone/audio sensor (MP23ABS1)

It is a low-power, MEMS capacitive microphone and an IC interface, which can be used to detect audio waves, designed by STMicroelectronics.

Its basic characteristics are the following:

- Operating voltage 1.52 V to 3.6 V
- Omnidirectional
- Flat audio frequency response
- High signal-to-noise ratio (64 dB)
- Temperature range -40 °C to +85 °C
- Sensitivity of 38 dBV ± 1 dB

### 2.1.10. Humidity sensor (HTS221)

It is a low-power digital relative humidity and temperature sensor chip, designed by STMicroelectronics. It is based on a polymer dielectric planar capacitor structure.

It main features are the following:

- Operating voltage from 1.7 V to 3.6 V
- 2 $\mu$ A low-power consumption
- 1 to 100% relative humidity range
- Humidity accuracy: ±4.5% rH (20% to 80% rH)
- Temperature accuracy of ±0.5 °C (from +15 °C to +40 °C)
- 16-bits humidity and temperature output data

- I2C and SPI interface
- Factory calibrated
- Operating temperature range from -40 °C to +120 °C

### 2.1.11. Bluetooth connectivity (SPBTLE-1S)

It is a low-power, low-energy Bluetooth v4.2 compliant module (BLE), used to provide Bluetooth connectivity to the SensorTile.box. It was designed by STMicroelectronics.

Its basic characteristics are the following:

- Operating voltage from 1.7 V to 3.6 V
- Cortex-M0 32-bits processor based
- Supports both master and slave modes
- 1 x UART, 1 x I2C, 1 x SPI, 14 x GPIO, 2 x multifunction timer, 10-bit ADC, watchdog and RTC, DMA controller, and SWD debug interface
- AES security
- Pre-programmed UART bootloader
- Extended operating temperature (from -40 °C to +85 °C)

### 2.1.12. STBC02AJR Li-Ion linear battery charger

It is a linear Li-Ion battery charger, which provides 150 mA, and which was designed by STMicroelectronics. This device includes a charger enable input to stop the charging process anytime. It is powered off automatically if a battery is not connected to the device. Additionally, an external thermistor can be used to detect a battery under or over the temperature conditions set by the programmer.

Its basic features are the following:

- Fast charging (up to 450 mA)
- Adjustable voltage (up to 4.45 V)
- Auto recharge function
- Protection circuit against overcharge, over-discharge and battery overcurrent
- Charge/fault status output
- Low battery leakage in over-discharge and shutdown modes
- Two 3 Ω SPDT load switches
- Smart reset/watchdog block

### 2.1.13. STBB3JR 2MHz DC-DC converter

It is a fixed voltage DC-DC converter, which provides the input and output voltage ranges listed below, although we can apply higher voltages. It was designed by STMicroelectronics. This converter is based on N-channel and P-channel MOSFET switches.

Its main characteristics are the following:

- Minimum input voltage range from 1.8 V to 5.5 V
- Output voltage range (adjustable) from 1.2 V to 5.5 V
- Output current of 2A (with Vin from 3.6 to 5.5 V)
- Output current of 800 mA (with Vin = 2 V)
- High efficiency (94%)
- 2 MHz switching frequency
- Load disconnect during shutdown
- Less than 50 $\mu$ A quiescent current (<1 $\mu$ A shutdown current)

## 2.2. Raspberry Pi

The Raspberry Pi is a series of simple board computers (SBC) with a reduced cost, developed in the United Kingdom by the Raspberry Pi Foundation. Technology has evolved over time with the purpose of making lives easy and convenient. This device was a major development in the technology that made computer learning easy enough that anyone with little effort can use it. Although the purpose of the original model was to promote computing in schools, it ended up being more popular than expected, since it has even been sold for robotics use, among others. This device does not include peripherals (such as keyboard or mouse) or casing. However, there have been cases in which some accessories have been included, both in official and unofficial packages.

Although it is not sure whether its hardware is open source or not, we can affirm that its software is open source. The official operating system is an adapted version of Debian, and it is called Raspberry Pi OS. However, this device can also use other operating systems, including a version of Windows 10.

Every Raspberry Pi model includes a Broadcom processor, RAM memory, graphics processing unit (GPU), USB ports, HDMI, Ethernet (except the first model), 40 GPIO pins (since Raspberry Pi 2) and a camera connector. None of its editions includes memory, but the first version had a SD card, and following versions incorporate a MicroSD card.

The main programming language that the Raspberry Pi Foundation promotes is Python, although the boards can also admit other languages, such as Tiny BASIC, C, Perl and Ruby.

The specific board we are going to use is the Raspberry Pi 3 B+. It came out in March 2018 to update the previous model (the Raspberry Pi 3 Model B, introduced in 2016), turning into the latest product in the Raspberry Pi 3 range. It includes a 64-bit quad core processor running at 1.4GHz, dual-band 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet (up to 300Mbps), and PoE capability via a separate PoE HAT.

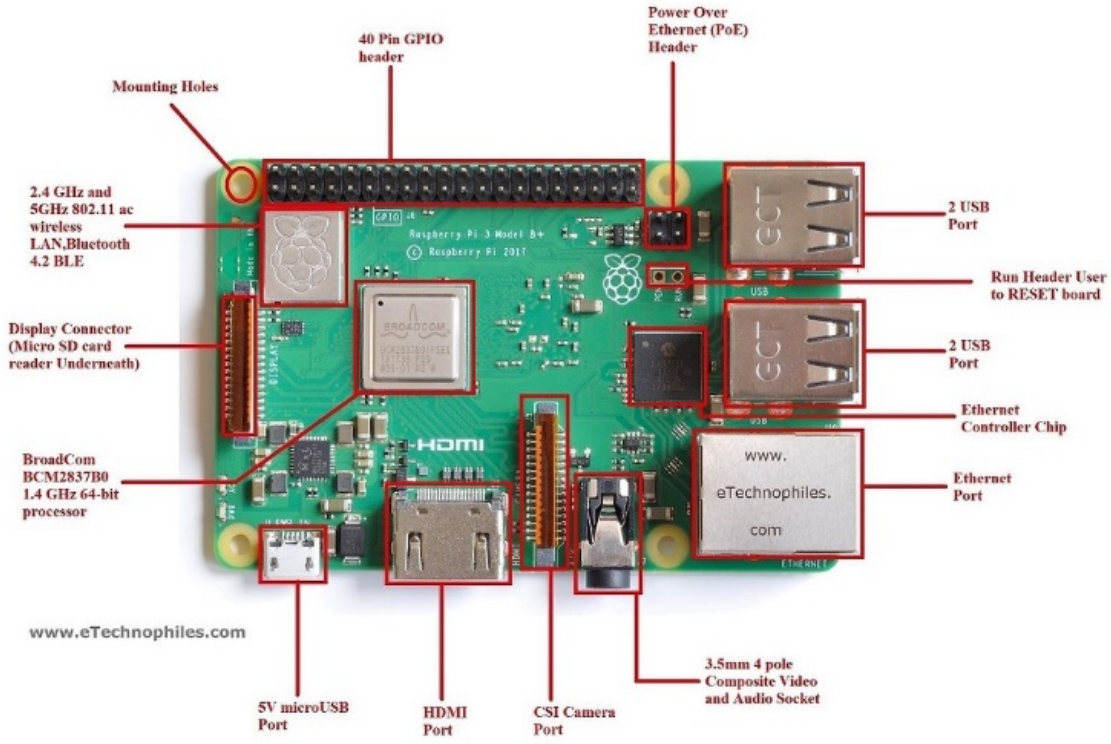


*Figure 6. Parts of the Raspberry Pi 3 B+*

The main processor is housed in a metal package, which allows keeping the temperature and pressure of the board as low as possible, with the aid of a heat sink. It can operate between 0 °C and 50 °C.

The dual-band wireless LAN comes with modular compliance certification. Testing for interference (also known as EMC) can be incredibly costly and difficult to isolate. However, thanks to the WLAN modular compliance certification, one can expect significantly lower EMC issues when integrating the board into a product, improving both cost and time to market.

It has a SDRAM memory of 1GB.

It includes four USB ports, which are two more than the USB ports that the previous version had, avoiding the hassle of having to use al external USB hub when aiming to join a number of peripherals with the device. The USB hard drive is used to boot the device, and it is identical to the hard drive of any regular computer.

We can access the Raspberry Pi through a 40-pin GPIO. Figure 7 shows the pinout of the Raspberry Pi 3 B+. As we can see, out of these 40 pins, 26 are used a digital I/O pins, 9 are dedicated I/O pins (they do not come with alternative function), pins 3 and 5 come with an on-board pull up resistor of 1.8 kΩ, and pins 27 and 28 are dedicated to ID EEPROM.

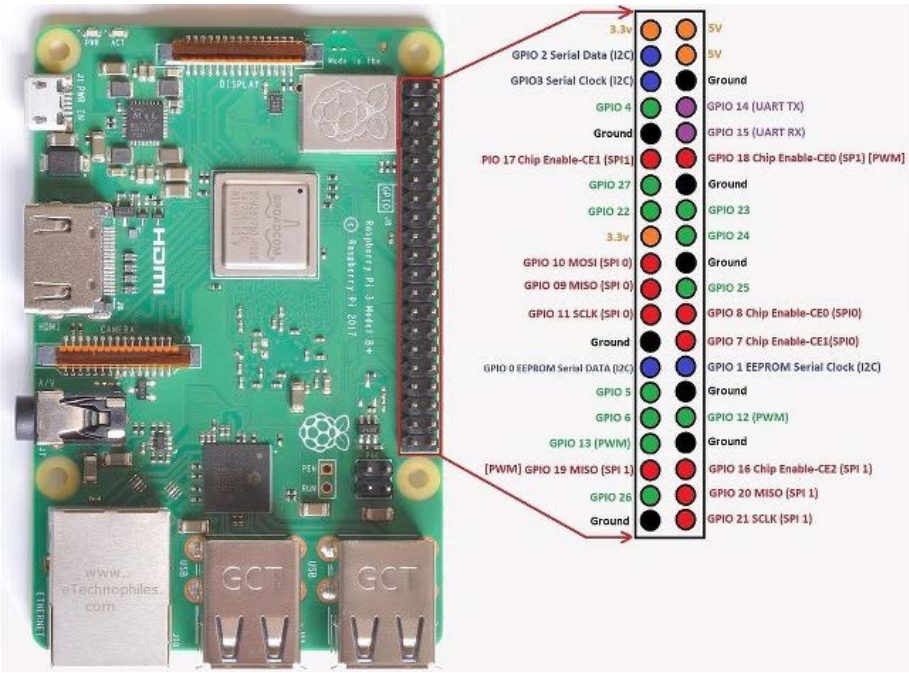


*Figure 7. Pinout of Raspberry Pi 3 B+*

A four-pin header is added on the board, and it is placed near the 40-pin header, allowing the use of data cables instead of power cords. This is very useful, since it reduces the number of cables required for the installation of the device in a bigger project.

With respect to video and sound, this device has one full-size HDMI, a MIPI DSI display port, a MIPI CSI camera port, and four pole stereo output and composite video ports.

We can load the operating system and store data through a micro SD.

The input power via micro USB connector is 5V/2.5A, whereas via GPIO header is 5V.

# 3. Methods

In this chapter, we will introduce a general idea about Bluetooth-based localization, we will describe several machine learning algorithms (in particular KNN), and we will give an overview of the routine generation and classification.

## 3.1. Bluetooth localization

Bluetooth Low Energy (BLE) is a subset of the Bluetooth 4.0 standard, which works in the range of 2.4 GHz. It is designed and commercialized by Bluetooth SIG, and it is used in several applications, such as health, fitness, beacons or security. It allows the communication between button cell devices and Bluetooth devices. If we compare it with the classical Bluetooth, it consumes much less power, around a 10% of the total power needed for the classical Bluetooth. However, it can maintain a similar communication range (up to 100 meters). Since 2010, this BLE protocol gained strength, and its use became generalized because it allowed wireless communications in the short and medium ranges. A BLE device operates in suspension mode the majority of the time, becoming active only to make connexions that last a few milliseconds, and this makes BLE the preferred protocol for applications that only inform sporadically.

One of the most important advantages of BLE is the acceptance it has from most of the huge nowadays platforms, such as IOs, Android, BlackBerry, macOS, Microsoft or Linux, between others. Additionally, BLE is compatible with all these platforms, avoiding the need for additional hardware. Moreover, Bluetooth SIG predicts that 100% of Bluetooth devices commercialized until 2024 will withstand BLE.

Its most extended and used application is beacons (BLE emitting devices), which have already been tested in several sectors, such as retail trade, industry, logistics or medical attention, between others. As it offers a good experience with low expenditure, it is an excellent option for those applications that need a more continuous monitoring, with a high data processing and precision. Therefore, the BLE protocol has been widely used to monitor people and mobile units in closed environments.

As we have previously seen in the overview of the state of art, there are many indoor-monitoring applications where Bluetooth localization has been chosen. As a result, we can say that Bluetooth localization is one of the most common options with regards to indoor-monitoring, since it is non-invasive, easy to use and the BLE beacons are easy to install.

## 3.2. Machine learning algorithms

Machine learning (ML) can be framed in the area of artificial intelligence (AI), which uses historical input samples (training data) to make new predictions or decisions without being explicitly programmed to do so. This is done through algorithms. Its behaviour is similar to the learning process of a human being, and it gradually improves its accuracy. The main objective of ML is to allow computers to learn autonomously without the need for human intervention.

In general, we can divide ML into 3 parts:

1. Decision process. ML algorithms use historical data as inputs, in order to obtain a prediction.
2. Error function. It evaluates the accuracy of the prediction of the model by comparing it with some existing examples.
3. Model optimization process. If we can make the model to fit better the training data, weights are adjusted to make the model more accurate. This “evaluate and optimize” process will be repeated continuously, until the model reaches an acceptable error threshold.

We can find several ML methods, which are going to be described in the following sections.

### 3.2.1. Supervised ML

It is the simplest ML model to understand. It uses labelled datasets to train the algorithms to classify data or predict outcomes accurately. The key point is to create a correct function that can be generalized to data it has never seen before. We can test this model by giving it certain training examples, with one or several known inputs and a known output. This means that both the input and the output are known variables. In the mathematical model, each training example is represented by an array, called feature vector, and the training data is represented by a matrix. The model adjusts its weights until it fits the input data with an acceptable threshold error, avoiding overfitting and underfitting.

Inside the supervised ML umbrella, we can find several sub-methods, and some of them will be briefly explained below:

- **Artificial neural networks (ANNs)**. They are composed of node layers, including an input layer, one or more hidden layers, and an output layer. Each node, called artificial neuron, is connected to one another through weights, which alter the importance of certain input over others. These weights are given a threshold. If the output of any artificial neuron is above this threshold, that node is activated, sending data to the next layer of the ANN. On the other hand, if the output of a node does not exceed the pre-set threshold, that node passes no data to the next layer of the ANN.

- **Decision trees (DT)**. They create trees that predict the result of an input vector based on decision rules applied from the data characteristics. They are useful because they are easy to visualize, and we can better understand what lead to the output. There exist two types of decision trees, which are classification trees (the target variable is a discrete value and the leaves represent class labels) and regression trees (the target variable can take continuous values).
- **Random forest (RF).** It is an ensemble of decision trees combined with bagging. This combination with bagging makes different DT see different data proportions. As none of the trees sees all training data, each tree is trained with different data samples for the same problem. By doing this, if we combine the results of all decision trees, some errors compensate with others, and we have a more accurate generic prediction.
- **Naïve Bayes**. It is an easily implementable algorithm that uses the Bayes' theorem to classify objects. It assumes strong or naïve independence between attributes of data points.
- **Support vector machine (SVM)**. It is a representation of different classes in a hyperplane in multidimensional space. This hyperplane will be generated iteratively to minimize the error. The final objective of the SVM method is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

### 3.2.2. Unsupervised ML

This algorithm analyses and clusters unlabelled and unclassified datasets. It never knows the correct output with certainty, so its goal is to build a mapping function that sorts the data into different classes, based on hidden data features, without the need for human intervention. Although we can extract data from this first level clusters, we can sub-divide them if necessary.

The main unsupervised machine learning algorithms are the following:

- **K-means clustering**. Its goal is to divide examples from a dataset into *k* clusters. Each of this example is assigned a numerical vector, so that we can calculate the Euclidean distance between them. It is a very simple algorithm, both to understand and implement. We randomly assign each example from the dataset to a cluster. Then, we calculate the centroid, which will tell us whether each example is in its correct cluster or not. In each iteration, examples are moved to their corresponding clusters. This process is repeated until no example moves clusters, it means, until every example is in its definitive cluster. We can see that this process is made without supervision, since this algorithm does not need any understanding of the example vectors characteristics.
- **Adaptive resonance theory (ART)**. It is a self-organizing neural network architecture. Similar to k-means clustering, we can use ART for clustering, with the advantage that ART can create new clusters or eliminate unnecessary ones based on the dataset. It has three key steps: a comparison field (determines how a new feature vector fits in the existing clusters), a recognition field (has neurons that represent the active clusters), and a reset module. This approach allows learning new mappings while keeping the existing knowledge, since weights of existing neurons are not updated when a new neuron is created.

### 3.2.3. Semi-supervised ML

It falls between supervised ML (high performance) and unsupervised ML (high efficiency). It is trained with a small labelled dataset, which belongs to a larger unlabelled dataset. The algorithm can learn the dimensions of the dataset, and it can generalize to unlabelled data. This method solves the problem of not having enough labelled data for supervised ML algorithms, and also reduces the costs of having to label a huge amount of data for training.

### 3.2.4. Deep learning

It is a neural network made up of more than three layers (including the input and the output layers). It can use labelled datasets as inputs for training, which is also called supervised ML, as we have seen in previous sections. However, it can have unstructured data in its raw form (for example, text or images) as training inputs. Either way, it can determine the characteristics that differentiate some data categories from others. This avoids the need for human intervention, and allows the use of larger datasets.

### 3.2.5. Reinforcement ML

It is a similar model to supervised ML, but this algorithm is not trained with sample data. Instead, it uses the "trial and error" method. The model tries to learn actions for some given states that produce a goal state. Unlike supervised ML, in reinforcement ML we do not give feedback (either positive or negative) after each action. On the contrary, we reinforce the algorithm with a reward when several successful outcomes have been produced.

One of its advantages is that it is able not just to learn how to map an input into an output, but also a set of inputs into outputs with dependencies. Its aim is to maximize the number of rewards it is given. Its main applications are multi-step processes with clearly defined rules to accomplish an ultimate goal.

The most widespread approach for reinforcement ML is Q-learning, whose general algorithm consists in learning rewards in an environment in $Q$ stages. During learning, actions are probabilistically selected as a function of the $Q$ values, which allows exploration of the state-action space. $Q$ values are updated for each state, and a reward is given. When the goal state is reached, the process begins again, starting from an initial position. When the algorithm has finished, we use the $Q$ values greedily, meaning that we use the action with the largest $Q$ value for a given state. By doing this, we can exploit the gained knowledge to optimally reach the goal state.

### 3.2.6. KNN

It is a supervised ML technique. It compares the new data to the existing data and puts this new data into the category that resembles the most to the available categories. It can be used both for regression and classification problems. It is a non-parametric algorithm, meaning it does not make any guesses on primary data. It is sometimes called a lazy learner algorithm, since it does not learn immediately from the training set, but it stores the dataset and only performs an action on that dataset at the time of classification.

In this project, we will use this ML algorithm to identify the position where a person is in an indoor-environment. To do so, once we have all data, we will separate it, in a way that a certain percentage will be used to train the system, and the remaining data will be used to test the algorithm.

The KNN algorithm follows the steps described below:

1. Select the value *K* to set the number of neighbours. We have no particular way to determine the optimal value of parameter K, so we have to try with several values and select the one that gives us the best results. Typically, we choose *K*=5. If K is very low (1 or 2), it can lead to mistakes because the model would be noisy. If we select a very large value for K, the results will be better, but we would have computational difficulties.
2. Calculate the Minkowski distance between the *K*-nearest neighbours, which can be done applying the formula below. *X* and *Y* are the Cartesian coordinates of the point we want to calculate; $x_i$ and $y_i$ are the distances from the origin for the point we want to calculate. The parameter *p* has to be an integer value.

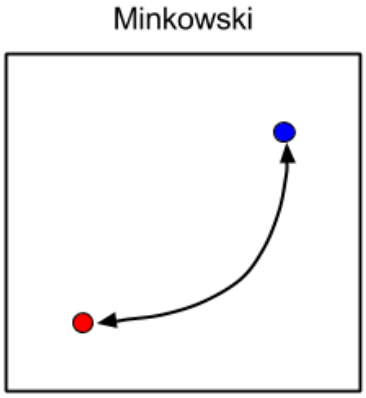


*Figure 8. Minkowski distance*

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

3. Identify the K-nearest neighbours by analysing the Euclidean distance.
4. Count the number of points in each category that belong to the K-nearest neighbours.
5. The new point belongs to the category with the highest number of neighbours.
6. The algorithm has ended.

The KNN algorithm has several advantages, such as its simple implementation, its robustness to the noisy training data, or its higher effectiveness if the training data is large.

However, it also has some drawbacks, like the mandatory need to determine the value of *K*, which can sometimes be difficult, or the high computational cost that derives from the need to calculate the Minkowski distance between data points for all the training samples.

## 3.3. Routine generation and classification

In this section, we will first talk about the additional configuration we need to do to the SensorTile.box and the Raspberry Pi, so that we have them completely ready to carry out the project. Then, we will explain in what consists the routine and how we have generated it. After this, we will decipher the format of the files we obtain from the Raspberry Pi. Finally, we will analyse in what consists the localization process, including the explanation of the different steps we must follow.

### 3.3.1. SensorTile.box's configuration

Concerning the SensorTile.box, we will use the default program, which allows this gadget emits BLE signals continuously, when it is turned on. Therefore, we do not have to write any additional coding.

### 3.3.2. Raspberry Pi's configuration

To configure the Raspberry Pi, we have to load their operating system inside its SD card. To do so, we have to follow several steps, which will be described below.

First, we have to install an application on the computer, called Raspberry Pi Imager. To this end, we get into the original Raspberry Pi web page (raspberrypi.es) through any browser, and we select "Install Raspberry Pi Imager". When we are asked to choose the operating system version, we choose the first one (the recommended one), which is version 1.7.2.

After we have installed this application correctly, we will have to insert the Raspberry Pi's SD card in the computer, in order to load the operating system inside it. To do this, we open Raspberry Pi Installer v1.7.2 on the computer. In this application, we have to choose two parameters:

- Operating system. Raspberry Pi OS (32-bit).
- Storage. Raspberry Pi SD card.

Finally, we have to click on "Write" for loading the operating system inside the SD card. When this process has been completed, we have to safely extract the SD card from the computer and introduce it in the Raspberry Pi. With this, when we connect the Raspberry Pi to a monitor, the operating system will start automatically.

In order to use the different Raspberry Pi as beacons, we have to configure them as such. The first thing we have to do is to load the operating system inside each Raspberry Pi SD card, which we do following the steps described in the section above. Then, we have to connect the Raspberry Pi to a monitor through an HDMI cable. The first time, we do this, we will need to configure several parameters.

- Select country (Spain), language (European Spanish) and time zone (Madrid).
- Choose username and password. We have to confirm this password.
- In case the edges are cut off, we can reduce the size of the desktop to adjust it to the monitor.
- If we want to connect the Raspberry Pi to a WiFi network, we can also do this in these steps, and we will also have to write the password. We can skip this step if we wish.
- We can update the software.
- Finally, we must restart the Raspberry Pi so that the new setting will take effect.

Once the Raspberry Pi has been correctly restarted, we will observe that a developing system similar to Linux will show up.

The 4 Raspberry Pi installed in the elderly's home capture data coming from every device that has BLE. All these generated files are uploaded to Google Drive and Github. We do this so that we do not have to unplug the Raspberry Pi every time we want to collect the stored data. In Google Drive, we create 4 folders, one per Raspberry Pi, so that the corresponding data associated to each beacon is saved in the corresponding folder. In Github, we have to create a repository specific for our project, and create 4 folder in it, whose purpose will be the same as we have described for Google Drive.

#### 3.3.2.1. Google Drive authentication files

We have two files corresponding to the Google Drive authentication, which are client_secrets.json and settings.yaml.

The json file gives us the client id, the project id, the authentication uri, the token uri, the certification url, the client secret, the redirect uris, and the javascript origins. This file is automatically created when we complete the Google Drive authentication through Google Cloud Platform.

The yaml file includes more specific information related to the authentication keys and tokens. It also generates the credentials file (credentials.json).

#### 3.3.2.2. RP_script.py

The RP_script.py Python code is the responsible for saving the data files and storing them in Google Drive and Github. Its most relevant parts are briefly described below:

- "connected_to_internet" function. This code line checks whether Internet connection exists. If it does, we can execute the functions to upload the data files to Google Drive and Github. If it does not, the data files will be kept in the Raspberry Pi "pi2" folder, waiting for the Internet connection to be back.
- upload_file_GD function. If we have Internet connection, this function verifies the Google authentication code, and then uploads the data files to the Google Drive folder we specify in that function.
- upload_file_Github function. It works in a similar way to the previously defined upload function. The difference is that it uploads the data files to the folder "Home" inside the corresponding repository we have created beforehand.
- From the last "while" loop until the end of the script. These lines are the ones responsible for generating the data files each hour, and storing them inside the "pi2" folder of each Raspberry Pi.

In order to use this file correctly in the command window of the Raspberry Pi operating system, we have to make it executable, by writing the next command: "chmod –x ble_scan.py". After we have done this, if we look inside the "pi2" folder in the command window with "ls", we will see that this file shows up in green, confirming we have made this file executable correctly.

#### 3.3.2.3. Shell files

We have two shell files inside the "pi2" folder, which are run_blescan.sh and kill_blescan.sh.

The first one follows a simple set of steps. First, it restarts the Bluetooth and waits for 5 seconds to assure that this reset has been completed correctly. Then, it accesses the "pi2" folder inside the Raspberry Pi and, finally, it executes the RP_script.py code.

The second shell file terminates the RP_script.py file, stopping its execution.

#### 3.3.2.4. Libraries

We have to create a folder in the file explorer, inside home – admin, called "pi2". In this way, the path to open this folder will be "cd /home/admin/pi2". In this folder, we have to copy several files, which correspond to the ones we generated previously with our computer. We can copy these files through a pen drive or via email. Either way, we must compile all files in a .zip format so that we can decompile them correctly inside the Raspberry Pi.

Then, we open the command window and start the configuration by installing all necessary libraries, which are listed below, with their corresponding description:

- PyGithub. It is used to connect the Raspberry Pi to Github and to upload the files there.

- PyDrive. It is used to connect the Raspberry Pi to Google Drive and to upload the files there.
- Libbluetooth. It is the BLE controller, it means, it is used to communicate the Raspberry Pi chip to its Bluetooth module.
- Bluez. It is the official Linux stack to work with Bluetooth. It is based on several open-source protocols. Since it is integrated in the system, the deployment of applications based on BlueZ is easier.

#### 3.3.2.5. Automatic execution

With all this configured, we are prepared to execute the Python code, which is the one that makes the Raspberry get the data and upload it to Google Drive and Github, as we said previously. In order to check whether the executable works properly, we can make an easy test by executing the shell file "run_blescan.sh" manually. To do so, we first have to access the "pi2" folder. Then, we write the following command: "sh run_blescan.sh".

The first time we execute this command, a link will appear. We have to open that url to confirm the authentication of Google Drive to upload the files in that folder. To do so, we are asked to introduce the email address associated to the folder where we are going to upload the files. Additionally, we also have to write the password of that email address. If we have done this correctly, a confirmation message will show up in the screen.

When this instruction has been executed completely, we look at the folders of both Google Drive and Github. If the files have been uploaded, we know that "run_blescan.sh" is working correctly, and we can continue programming.

We do not want to manually execute this shell file each time we want the Raspberry to receive and upload BLE files. Therefore, we have to automate this process with the daemon "crontab". It is a Linux tool used to set the moment when we want a certain file to be executed. To edit crontab, we can write 2 instructions on the command window:

- crontab –e. We have restricted licence because we access it with our user (admin).
- sudo crontab –e. We have all licences because we access it with the root user (root). It will be the one we use in this case so that we can perform all actions we need.

When we execute the command to edit the crontab file, we have to choose between several text editors. In our case, we will use nano, which is option 1. After we execute this code line, a nano text file will open, in which we will see several commented lines. In these lines, the working mode of crontab is briefly explained.

The general line to program the execution of a shell file to a certain moment is the next one:

minute(0-60) hour(0-23) day(1-31) month(1-12) weekday(0-6) command path

For example, if we want to execute the shell file "run_blescan.sh" on Monday 4$^{th}$ of July at 12:23, we will write the following code line:

23 12 4 7 1 sh /home/admin/pi2/run_blescan.sh

However, we do not want to set up a specific time to execute the shell file. We want it to run when the Raspberry is turned on, it means, when we connect it to the power grid. To do this, we will have to use a more specialized command inside crontab. We write the following code line:

@reboot /home/admin/pi2/run_blescan.sh

Once we have written all code lines inside the nano text editor, we save the changes (ctrl + o) and exit to the command window (ctrl + x).

### 3.3.3. Routine generation

Our objective with this project is to analyse older adults' routines and detect possible deviations, which may lead them to some physical or cognitive issues. By doing this using BLE beacons, we maintain the elderly's privacy.

In our case, these BLE beacons will be 4 Raspberry Pi, whose configuration will be described in detail in following sections. We have determined that the key rooms are: the bedroom, the living room/dining room, the kitchen and the bathroom. Therefore, we install one of our Raspberry Pi in each of these rooms. They are easy-to-install, since we just have to plug the charger into a socket. They are also non-invasive devices because they are small and they do not take up very much space.

These 4 beacons will be continuously storing information coming from each Bluetooth device in their range. However, we are just interested in the data coming from the SensorTile.box, which is the wearable device that the elderly will be carrying with themselves all day long. This is also a very compact device, which will not interfere in the normal life of the person.

In order to carry out the experiments, we asked a volunteer about the routine he had during the week, which is the one we see in Table 1, Table 2 and Table 3.

| Monday, Wednesday, Thursday and Friday | | |
|---|---|---|
| Initial hour | Final hour | Room |
| 00:00 | 08:30 | Bedroom |
| 08:30 | 08:50 | Bathroom |
| 08:50 | 09:00 | Kitchen |
| 09:00 | 14:00 | Bedroom |
| 14:00 | 14:30 | Kitchen |
| 14:30 | 15:00 | Living room/Dining room |
| 15:00 | 15:05 | Bathroom |
| 15:05 | 16:00 | Bedroom |
| 16:00 | 18:00 | Living room/Dining room |
| 18:00 | 20:00 | Outside |
| 20:00 | 20:30 | Kitchen |
| 20:30 | 21:00 | Living room/Dining room |
| 21:00 | 21:10 | Bathroom |
| 21:10 | 00:00 | Bedroom |

*Table 1. Monday, Wednesday, Thursday and Friday routine*

| Tuesday | | |
|---|---|---|
| Initial hour | Final hour | Room |
| 00:00 | 06:45 | Bedroom |
| 06:45 | 07:00 | Bathroom |
| 07:00 | 07:15 | Kitchen |
| 07:15 | 07:30 | Bedroom |
| 07:30 | 15:00 | Outside |
| 15:00 | 15:05 | Bathroom |
| 15:05 | 16:00 | Bedroom |
| 16:00 | 18:00 | Living room/Dining room |
| 18:00 | 20:00 | Outside |
| 20:00 | 20:30 | Kitchen |
| 20:30 | 21:00 | Living room/Dining room |
| 21:00 | 21:10 | Bathroom |
| 21:10 | 00:00 | Bedroom |

*Table 2. Tuesday routine*

| Saturday and Sunday | | |
|---|---|---|
| Initial hour | Final hour | Room |
| 00:00 | 09:00 | Bedroom |
| 09:00 | 09:20 | Bathroom |
| 09:20 | 09:40 | Kitchen |
| 09:40 | 13:00 | Living room/Dining room |
| 13:00 | 14:00 | Kitchen |
| 14:00 | 15:00 | Living room/Dining room |
| 15:00 | 15:05 | Bathroom |
| 15:05 | 17:00 | Bedroom |
| 17:00 | 21:00 | Outside |
| 21:00 | 21:30 | Kitchen |
| 21:30 | 22:30 | Living room/Dining room |
| 22:30 | 22:40 | Bathroom |
| 22:40 | 00:00 | Bedroom |

*Table 3. Saturday and Sunday routine*

Due to the fact that we do not have enough data for training the algorithms, this routine is artificially built based on real information provided by the volunteer and the carers. We can create this relatively accurate timelines because older people are quite routine, so the activities they perform, and their duration, are quite consistent throughout the week. In tables 1, 2 and 3, we can see that we have an initial and final hour (a specific time slot) for each room (bedroom, bathroom, kitchen, living/dining room, outside). These are symbolic locations, not specific ones, since we use the BLE beacons for a room level location.

After the tests have been completed, we end up with four folders, each one of them containing information obtained from each Raspberry Pi. These four folders will be stored in Google Drive and Github. Apart from these four folders, we need to create two additional files, which are described in the following bullet points:

- labels.txt. This file has each visit to each room and their duration. Its format is: In_date, Out_date, Room. Each date has the following format: YYYY-MM-DD HH:MM:SS.
- macs.txt. This file has the Bluetooth MAC for each Raspberry Pi and the room they are in. Its format is: Raspberry_Pi_MAC, Room.

In Table 4, we can see the names we refer to every Raspberry Pi (RP for short) and the SensorTile.box, their MACs, and the rooms they are located.

| NAME | ROOM | MAC |
|---|---|---|
| SensorTile.box | Not-fixed | C0:F8:BB:9D:12:93 |
| RP1 | Kitchen | B8:27:EB:6E:78:74 |
| RP2 | Bathroom | B8:27:EB:2F:C6:C9 |
| RP3 | Bedroom | B8:27:EB:28:CF:46 |
| RP4 | Dining room / Living room | B8:27:EB:B2:A0:90 |

*Table 4. Devices with their MACs and locations*

### 3.3.4. Raspberry Pi file formats

The Raspberry Pi receives signals from every Bluetooth device in its range. It scans for RSSI every few milliseconds, and stores that data inside a .csv (comma-separated values) file. It automatically gives these files a name, which follows the next nomenclature.

year_month_day_hour_minute_second_MAC.csv

The MAC (Media Access Control) we refer to in this file is the one belonging to the Raspberry Pi.

An example of a real .csv file is shown in the following figure.

2022_06_29_13_42_00_B827EB28CF46.csv

If we analyse it, we know that this file was generated the 29th of June 2022, at 13:42, by the Raspberry Pi with the B827EB28CF46 MAC.

Moreover, if we open these .csv files, we see it is made of columns and rows (i.e. a table). We can also find a specific format inside the file, in which each column refers to a different parameter.

- Column 1. It indicates that we work with Bluetooth (BLE). Although it is not our case in this project, we could also obtain these data with IMUs, for example.
- Column 2. Unix time. It is the time in nanoseconds since January 1st 1970 (UTC) until the moment when the file was generated.
- Column 3. Emitter MAC. It is the identification of the emitter, which includes every device which sends Bluetooth signals. In our case, we only need the SensorTile.box MAC, since it Is the emitter device we will use for the elderly people monitoring. In order to obtain only the data corresponding to the SensorTile.box, we will need to program a filter, which will be explained later.
- Column 4. RSSI (Received Signal Strength Indicator). It is the power (in dBm) with which the Raspberry Pi receives the MAC of the emitters (the ones in column 3).

We can see a real example in Figure 9.

| | | | | |
|---|---|---|---|---|
| 1 | BLE | 1656502919799944 | C8:69:CD:61:6F:2C | -68 |
| 2 | BLE | 1656502919811603 | 3B:26:C6:68:1E:E4 | -31 |
| 3 | BLE | 1656502919817428 | C0:F8:BB:9D:12:93 | -40 |
| 4 | BLE | 1656502919826266 | D0:03:DF:C9:BE:CF | -59 |

*Figure 9. Example of a .csv file*

## 3.3.5. Localization process

The localization process has been programmed with Pyhton programming language, using PyCharm as the developing environment and Anaconda as the library repository.

We have captured data from July 8th 2022 15:00:00 until July 19th 2022 23:59:59. This adds a total of 11 days and 9 hours. We can translate it to seconds, so that we have 982800 seconds of data.

However, we will consider 12 complete days for the routine analysis, assuming the volunteer was outside from 00:00:00 to 14:59:59 the first analysed day.

Once we have all .csv files from all Raspberry Pi (which we can get from Google Drive or Github), we have to process them in order to obtain the routine that the old person has performed. To do so, we have to follow the flowchart in Figure 10.

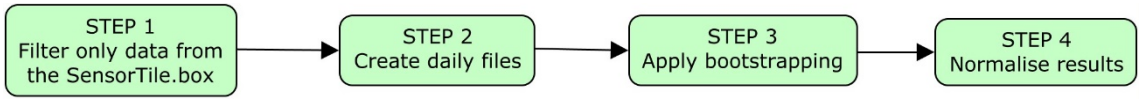


*Figure 10. Localization process flowchart*

### 3.3.5.1. Step 1. Filter only data from the SensorTile.box

The raw hourly files coming from the Raspberry Pi contain data associated to all devices that emit Bluetooth in their range each hour. As we only want the data coming from the SensorTile.box because it is the device that the elderly will be carrying with themselves, this script processes the raw files in order to obtain just the rows corresponding with the information given by the SensorTile.box (whose MAC is C0:F8:BB:9D:12:93). The remaining information detected by the Raspberry Pi is deleted.

### 3.3.5.2. Step 2. Create daily files

This script processes the output files of the previous step. Its purpose is joining the previously filtered hourly files into daily files for each Raspberry Pi.

#### 3.3.5.3. Step 3. Apply bootstrapping

Up to this moment, we have a huge number of samples in each daily file, since each Raspberry Pi detects the RSSI of the SensorTile.box at the maximum rate that the CPU allows, which is around 1 sample every millisecond. Therefore, what we have to do is to reduce this number of samples for the signal to be softened.

To do so, we use the bootstrapping method, with a 60-second window and a 30-second step. This way, we have a 30-second overlapping. For each window, we randomly take 5 values and calculate their mean and their standard deviation. We repeat this process 100 times for each window, and then we calculate the mean and the standard deviation of the previously obtained means and standard deviations, so that we have 2 identification measures for each Raspberry Pi per 60-second window. By doing this, we soften the signal, and, at the same time, we stay true to the raw data. Since we work with 4 Raspberry Pi, we have to repeat this process 4 times. After doing this, we will have 8 identification measures for the total beacon system per 60-second window. Since we have less samples (a more reduced dataset), we obtain a more stable and less noisy version of the raw data.

To carry out this process, we use the next code line:

```
x, y, t = get_features(device, window_size, step, feature='BOOT')
```

These function parameters are:

- device = C0:F8:BB:9D:12:93 (SensorTile.box MAC)
- window_size = 60 (seconds)
- step = 30 (seconds)

With this information, we have 2880 RSSI values per Raspberry Pi per day. We obtain this value from the following mathematical calculation.

$$\frac{24h \cdot 60min \cdot 60s}{30s} = 2880\ RSSI\ values$$

As we have previously said, we have 982800 seconds of data (11 days and 9 hours). By applying the bootstrapping technique with 60-second windows and 30-second steps, we have 32760 RSSI samples per Raspberry Pi, in total. This result is obtained applying the following equation:

$$\frac{982800s}{30s} = 32760\ RSSI\ samples$$

#### 3.3.5.4. Step 4. Normalise results

Database normalisation is a technique which is commonly used to organise database table contents and data storing. It must be a mandatory step to guarantee a reliable and successful database.

If we do not normalise data, we will have the following consequences:

- Inaccuracy of database systems.
- Slowdown of processes.
- Inefficiency in operations.

In order to create a coherent database normalisation, we must have the following four key points in mind:

- Organise the data in logical groups, so that each group describes a part of the whole system.
- Minimise the amount of duplicated stored data.
- Optimise the data organisation so that, when we need to make any modification, this can be easily done by applying the change in just one place.
- Build a fast-accessible database, so that we can manage data efficiently and avoid compromising its integrity.

We will normalise the RSSI data by using the mean and the standard deviation, as it is shown in the following code line.

```
x = (x - np.mean(x, axis=0)) / np.std(x, axis=0)
```

# 4. Results

In this section, we will see the testing environment where the volunteer will be performing his daily activities. Additionally, we will see the results we obtain with regards to the three aspects we want to analyse with regard to the routines (localization, routine analysis and anomaly detection).

## 4.1. Testing environment

The operating structure that we are going to follow in this project involves two main devices. The first one is a Raspberry Pi, and the second one is a SensorTile.box. In previous sections, we described each one of them from a technical point of view, and we specified their function in our project. In short, the Raspberry Pi 3 B+ (from now on, referred as Raspberry Pi) will be used as the receiver device, whereas the SensorTile.box will play the role of emitter.

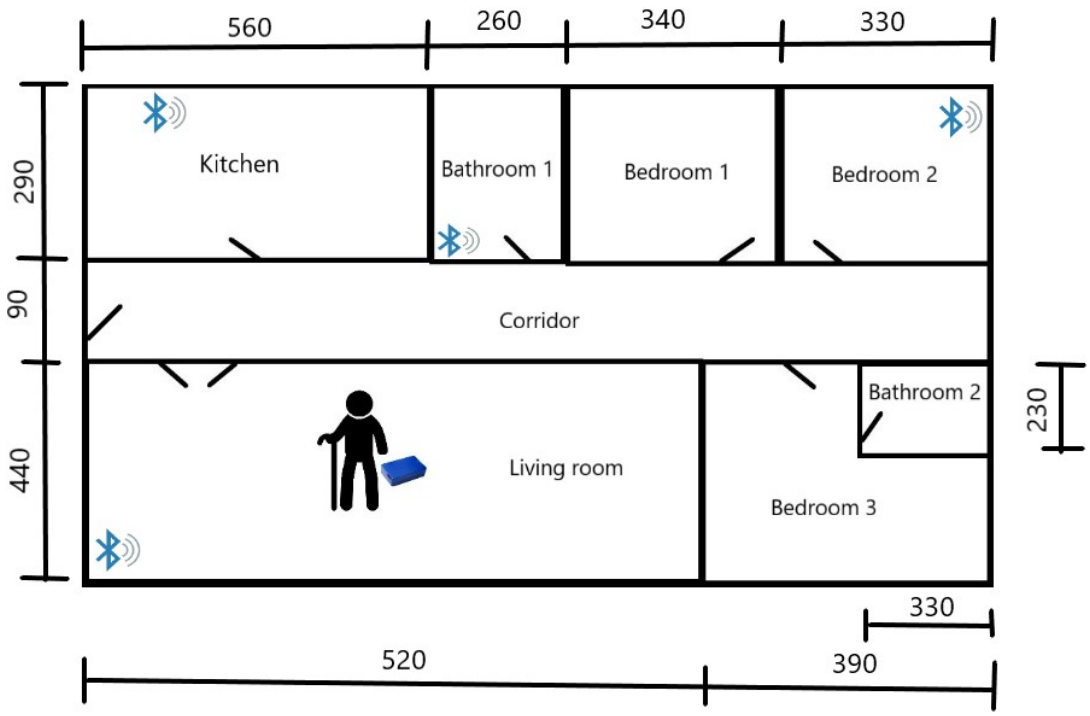


*Figure 11. Testing environment (the dimensions are given in centimetres)*

In Figure 11, we can see the environment where the testing will be made. All measurements are in centimetres.

First, the SensorTile.box will send signals with their corresponding powers. Then, the Raspberry Pi receives these signals, processes them, and saves them (both on the device itself, on Google Drive and on Github).

The elderly individuals will carry a SensorTile.box with them (whether it is in a pocket or elsewhere).

There will be several Raspberry Pi throughout the house (in general, one per room of interest). These devices receive every BLE signals within its range, not only the ones that the SensorTile.box emits. Therefore, an additional program will be necessary in order to filter these signals, and only obtain the necessary data for the follow-up of the elderly adults.

Once the Raspberry Pi has received and stored the filtered signals, we can proceed to their analysis. Each Raspberry Pi picks up a different power from the SensorTile.box, depending on the distance at which the person is from the receiver. The intensity of this power is shown in the parameter RSSI (Received Signal Strength Indicator). The bigger this number is, the closer the individual is to the corresponding Raspberry Pi.

After we have all these data related to the power of the signals, we have to locate the person. In order to do so, we will use fingerprinting techniques. In particular, the technique we are going to follow is the one explained in next lines. We will place ourselves with the SensorTile.box in several parts of the house, including different rooms and different locations inside each room. With this, all the Raspberry Pi will store the different RSSI that the SensorTile.box is emitting at each moment. Then, we will take note of the times in which we have been in each room. After this, we will look at the values of the RSSI in each noted time, and we will know that each time the person is located in a similar place as the one we have recorded previously, we can detect the exact position of the person.

## 4.2. Localization

After we have all normalised RSSI samples, we will use the KNN machine learning method, which has been described in detail in previous sections.

### 4.2.1. KNN processing

In order to complete the localization of the elderly at every time, we have to follow several steps, corresponding to the KNN algorithm.

#### 4.2.1.1. k-value

We know that, in order to use KNN method, we have to determine the optimal value of k, which is the number of nearest neighbours. We do this by executing the following code, where we consider values of k from 1 to 40.

```
k_range = range(1, 40)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(x_train, y_train)
    scores.append(knn.score(x_test, y_test))
plt.figure()
plt.xlabel('k')
plt.ylabel('accuracy')
plt.scatter(k_range, scores)
plt.xticks([0,5,10,15,20,25,30,35,40])
plt.show()
```

The output of this code is the graph on Figure 12.

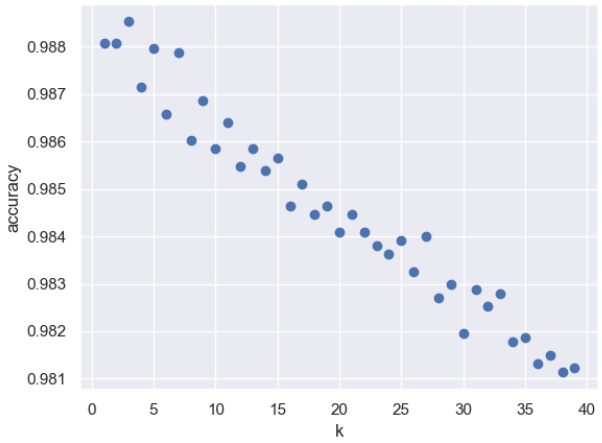


*Figure 12. Accuracy of KNN according to k-value*

In this graph, we can see the accuracy that the KNN reaches depending on the value of k chosen. In general, the bigger the value of k, the less accurate the system becomes. The optimal k-value is 3, since it is the one with the highest reached accuracy.

#### 4.2.1.2. Data separation

To use KNN algorithm, we have to separate the data between training and test. The training data are a percentage of the obtained data, which is used to train the system. The remaining data are the ones that are processed by the already trained KNN. This separation of data is carried out randomly with the following code line, and it allows the rooms to be proportional to one another.

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.33, stratify=y, random_state=42)
```

The function “train_test_split” is a predefined Pyhton function.

The output parameters are:

- x_train. They are the RSSI data used to train the system.
- x_test. They are the RSSI data used to test the system.
- y_train. They are the labels used to train the system.
- y_test. They are the labels used to test the system.

The input parameters are:

- x. RSSI values
- y. Labels
- test_size. It represents the proportion of the dataset to include in the test split. In this case, we test a 33% of the total data.
- stratify. The data is split in a stratified way. As we can see in the code, we split it by class label.
- random_state. It controls the shuffling of the dataset before we apply the split. The most popular integer random seeds are 0 and 42. In our case, we use 42, which is in the required range [0 , $2^{32}$-1].

#### 4.2.1.3. KNN training and test

To carry out the proper training and test of the KNN algorithm, we have to execute the next code lines:

```
classifier = KNeighborsClassifier(n_neighbors=3, metric='minkowski')
classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
```

first, we have to specify the value of k, which is 3, as we have previously calculated. We also have to determine the method with which we want to calculate distances, which is going to be the Minkowski distance in this case.

After this, we train the system with the predefined Python function “classifier.fit”, whose parameters are the train RSSI values and the train labels.

Finally, we store in the variable “y_pred” the test results, after having been processed by the prediction classifier.

#### 4.2.1.4. Metrics calculation

Now that we have all test data processed, we have to build a confusion matrix. A confusion matrix is a summarized table used to evaluate the performance of a classification model. The number of correct and incorrect predictions are summarized with the count values and are broken down by each class.

The structure of a confusion matrix is the following:

- Positive (P). The observation is positive.
- Negative (N). The observation is not positive.
- True Positive (TP). Result in which the model predicts correctly the positive class.
- True Negative (TN). Result in which the model predicts correctly the negative class.
- False Positive (FP). Result in which the model predicts incorrectly the positive class when in fact it is negative. It is also called type 1 error.
- False Negative (FN). Result in which the model predicts incorrectly the negative class when in fact it is positive. It is also called type 2 error.

Some of the most important metrics we can obtain from a confusion matrix are the following:

- Precision or positive predictive value. It is the proportion of relevant instances among the retrieved instances, which means that it indicates the proportion of correct positive identifications.

$$Precision = \frac{TP}{TP + FP}$$

- Recall, hit rate or real positive rate. It is the proportion of the total number of relevant instances that were actually recovered, which means that it indicates the proportion of true positives that was correctly identified.

$$Recall = \frac{TP}{TP + FN}$$

- F1-score. It is the measure of the accuracy of a test, which means that it is the harmonic mean of precision and recall. Its maximum value is 1 (perfect precision and recall) and its minimum value is 0. In brief, it is a measure of the accuracy and robustness of a model.

$$F1 - score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

The confusion matrix we obtain in our case is the one shown in Figure 13.

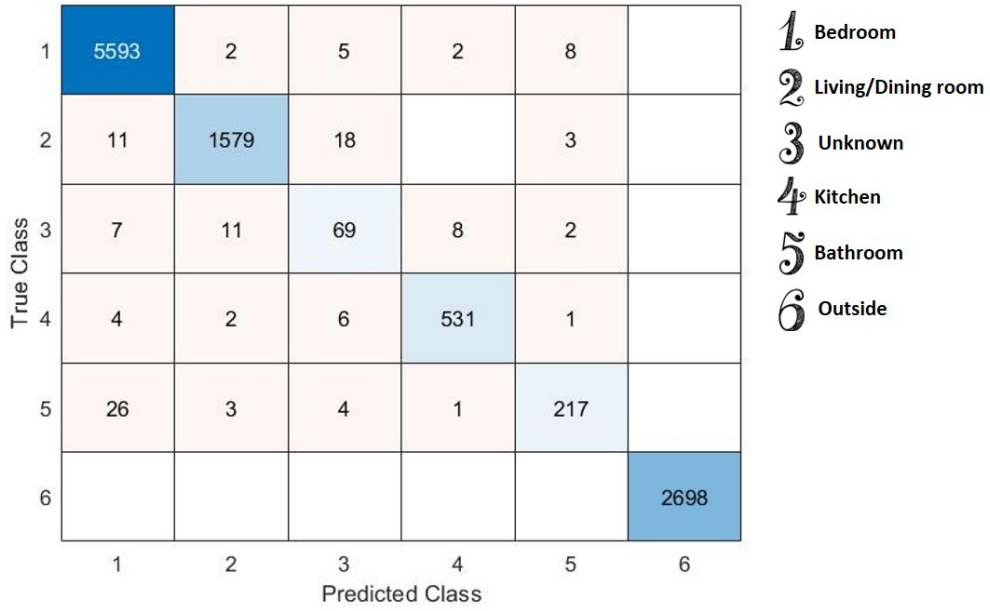


*Figure 13. Confusion matrix*

In this confusion matrix, we can see the number of data that have been identified like each class. For example, cell 1x1 indicates that 5593 samples that truly belong to class 1 have been correctly identified as class 1 samples in the prediction process. Another example, cell 1x2 show that 2 samples that truly belong to class 1 have been incorrectly placed as class 2 samples in the prediction process.

By analysing all data, we have Table 5, which describes the most important metrics associated to the confusion matrix.

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 1.0 | 0.99 | 1.00 | 0.99 | 5612 |
| 2.0 | 0.99 | 0.98 | 0.98 | 1612 |
| 3.0 | 0.68 | 0.75 | 0.71 | 98 |
| 4.0 | 0.98 | 0.98 | 0.98 | 544 |
| 5.0 | 0.94 | 0.88 | 0.91 | 247 |
| 6.0 | 1.00 | 1.00 | 1.00 | 2698 |

*Table 5. Confusion matrix metrics*

All in all, our KNN algorithm has a total accuracy of 0.9885.

## 4.3. Routine analysis

The routine analysis is programmed with Matlab, specifically with the 2020b version.

After we have all the final files with the definitive data, we can represent the routine of the elderly in a more visual way, through an image. This graph is shown in Figure 14, where we see that each room is represented by a colour. This has been done by a different Matlab script.

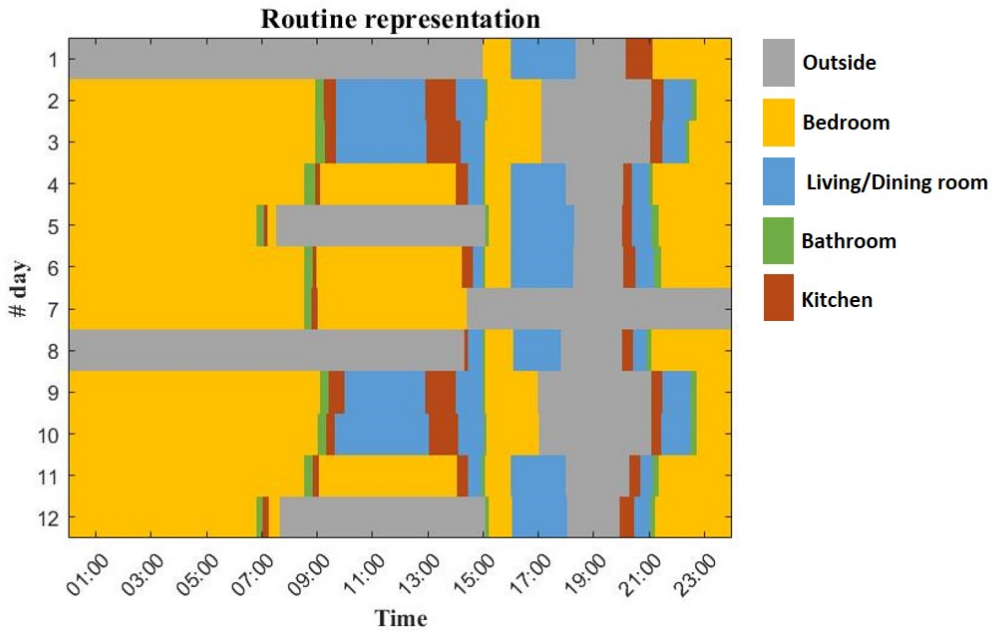


*Figure 14. Routine representation graph*

As we can see in Figure 14, the volunteer has carried out the routine throughout 12 days. With our naked eye, we can observe that we have several days that share characteristics, based on the time slots that the volunteer spends in each room, and noticing that they start and end in similar time instants. In particular, days 1, 7 and 8 are independent among them and with respect to the other days, since they do not have anything in common. Days 2, 3, 9 and 10 are similar. Days 4, 6 and 11 resemble each other. Finally, days 5 and 12 are similar, too.

Although there are 12 days to work with, we can analyse the volunteer's routine. We do this with the DBSCAN (Density Based Spatial Clustering of Applications with Noise) algorithm, which is a clustering algorithm that does not need the number of clusters to be specified as an input parameter of the algorithm. It can find clusters of different sizes and shapes, even if there is noise or outlier points, from a big number of data. To do so, it uses two parameters:

- Minimum number of points. It is the minimum number of points (threshold) grouped together for a region to be considered dense enough.
- Epsilon. It is the distance at which a point should be from the core point so that it can be included into the cluster.

Once we have finished the DBSCAN clustering, we can identify 3 types of points:

- Core. It is a point that has, at least, a certain number of points (n) within a distance epsilon from itself.
- Border. It is a point that has, at least, one core point within a distance epsilon from itself.
- Noise. It is a point that has less than a certain number of points (n) within a distance epsilon from itself, and it is neither a core point nor a border point.

The DBSCAN clustering algorithm follows certain steps:

1. First, the algorithm arbitrarily picks up a point from the data. This is done until all points from the dataset have been considered.
2. By taking this arbitrary point, the algorithm checks whether there are, at least, the established minimum number of points within a radius epsilon to the chosen point. If this is the case, all points inside that circumference become part of the same cluster.
3. The clusters continue expanding by recursively repeating the calculation described in the previous step for each neighbouring point.

Inside each cluster, we can find points (in our case, days) with similar properties. However, within those clusters, we can also detect anomalous days, which differ more than a certain threshold from the baseline day associated with that cluster. To identify these anomalous days, we have to calculate the MSE (Mean Square Error) between the days in each cluster and its baseline day. If this MSE exceeds a certain threshold, the analysed day is considered anomalous. The formula to calculate the MSE is the following, where N is the number of data points, $y_i$ is the actual value and $\tilde{y}_i$ is the predicted value.

$$MSE = \frac{1}{N} \cdot \sum_{i=1}^{N} (y_i - \tilde{y}_i)^2$$

To obtain all the MSE values between each day, we have the following code:

```
for i=1:1:days
    for j=1:1:days
        dist_corr_matrix(i, j) = immse(routine(i, :), routine(j, :));
%mean square error (MSE)

    end
end
```

If we execute the code above, we obtain the following matrix:

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.7452 | 0.7410 | 0.8141 | 0.4402 | 0.8241 | 1 | 0.0698 | 0.7492 | 0.7520 | 0.8196 | 0.4486 |
| 2 | 0.7452 | 0 | 0.0099 | 0.0836 | 0.2166 | 0.0876 | 0.1897 | 0.4577 | 0.0134 | 0.0072 | 0.0781 | 0.2075 |
| 3 | 0.7410 | 0.0099 | 0 | 0.0809 | 0.2132 | 0.0855 | 0.1980 | 0.4533 | 0.0157 | 0.0123 | 0.0743 | 0.2042 |
| 4 | 0.8141 | 0.0836 | 0.0809 | 0 | 0.2560 | 0.0170 | 0.2115 | 0.5017 | 0.0885 | 0.0843 | 0.0099 | 0.2442 |
| 5 | 0.4402 | 0.2166 | 0.2132 | 0.2560 | 0 | 0.2555 | 0.4343 | 0.2916 | 0.2204 | 0.2206 | 0.2557 | 0.0151 |
| 6 | 0.8241 | 0.0876 | 0.0855 | 0.0170 | 0.2555 | 0 | 0.2022 | 0.5233 | 0.0893 | 0.0873 | 0.0145 | 0.2582 |
| 7 | 1 | 0.1897 | 0.1980 | 0.2115 | 0.4343 | 0.2022 | 0 | 0.7111 | 0.1876 | 0.1870 | 0.1992 | 0.4298 |
| 8 | 0.0698 | 0.4577 | 0.4533 | 0.5017 | 0.2916 | 0.5233 | 0.7111 | 0 | 0.4603 | 0.4614 | 0.5121 | 0.2873 |
| 9 | 0.7492 | 0.0134 | 0.0157 | 0.0885 | 0.2204 | 0.0893 | 0.1876 | 0.4603 | 0 | 0.0075 | 0.0811 | 0.2108 |
| 10 | 0.7520 | 0.0072 | 0.0123 | 0.0843 | 0.2206 | 0.0873 | 0.1870 | 0.4614 | 0.0075 | 0 | 0.0791 | 0.2113 |
| 11 | 0.8196 | 0.0781 | 0.0743 | 0.0099 | 0.2557 | 0.0145 | 0.1992 | 0.5121 | 0.0811 | 0.0791 | 0 | 0.2489 |
| 12 | 0.4486 | 0.2075 | 0.2042 | 0.2442 | 0.0151 | 0.2582 | 0.4298 | 0.2873 | 0.2108 | 0.2113 | 0.2489 | 0 |

*Figure 15. MSE matrix*

We can see that the principal diagonal is composed of zeroes because the MSE between a day and itself is null, since they are the same. The larger the MSE is between one day and another, the least similar they are. For example, cell 1x2 indicates that the similarity between days 1 and 2 is of 0.7452, which means they are quite different. Another example, cell 1x8 indicates that the similarity between days 1 and 8 is of 0.0698, which means they are almost identical.

After we have this matrix on Figure 15, we have to normalise the data. To do so, we apply the following formula:

$$Normalised\ data = \frac{Actual\ data + Minimum\ data}{Maximum\ data - Minimum\ data}$$

Once we have all MSE data, we introduce them to the DBSCAN algorithm, using the "dbscan" Matlab function.

```
idx_dbscan = dbscan(dist_corr_matrix, 0.1, 1); % The default distance
metric is Euclidean distance
```

By doing this, we obtain the clusters associated to each day. As we can see in Figure 16, they coincide with the clusters we previously saw visually.

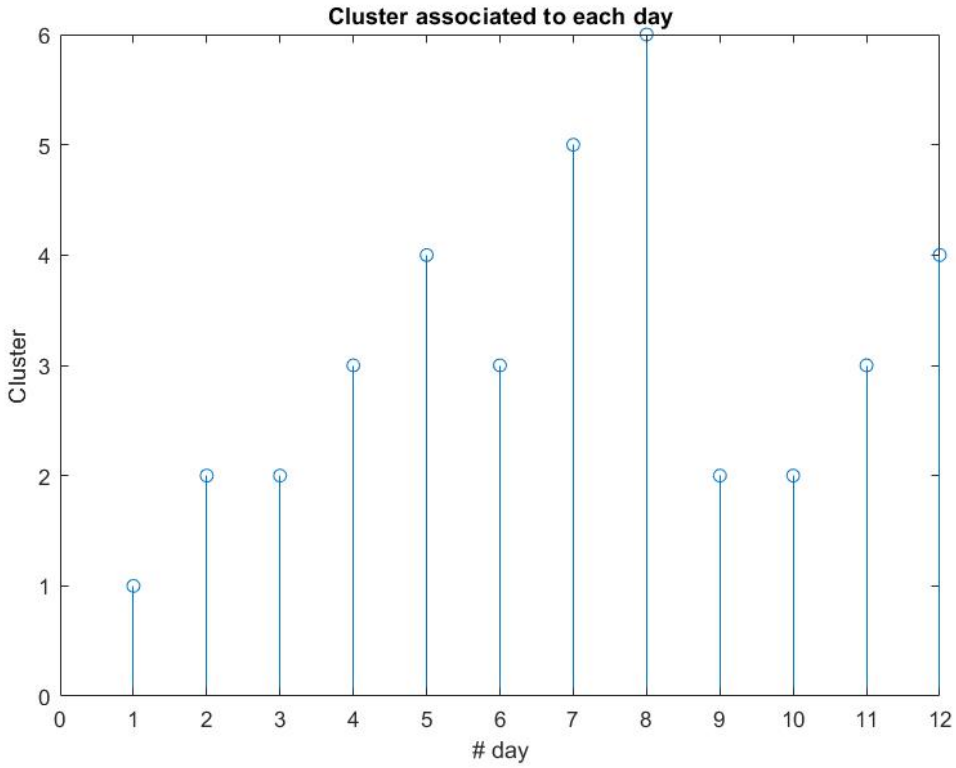


*Figure 16. Clusters associated to each day*

Since we have a small amount of days in the dataset, it is difficult to make an anomaly analysis. However, once the MSE has been calculated between each studied day, the DBSCAN clustering algorithm creates 6 baseline days, which can be seen in Figure 17.

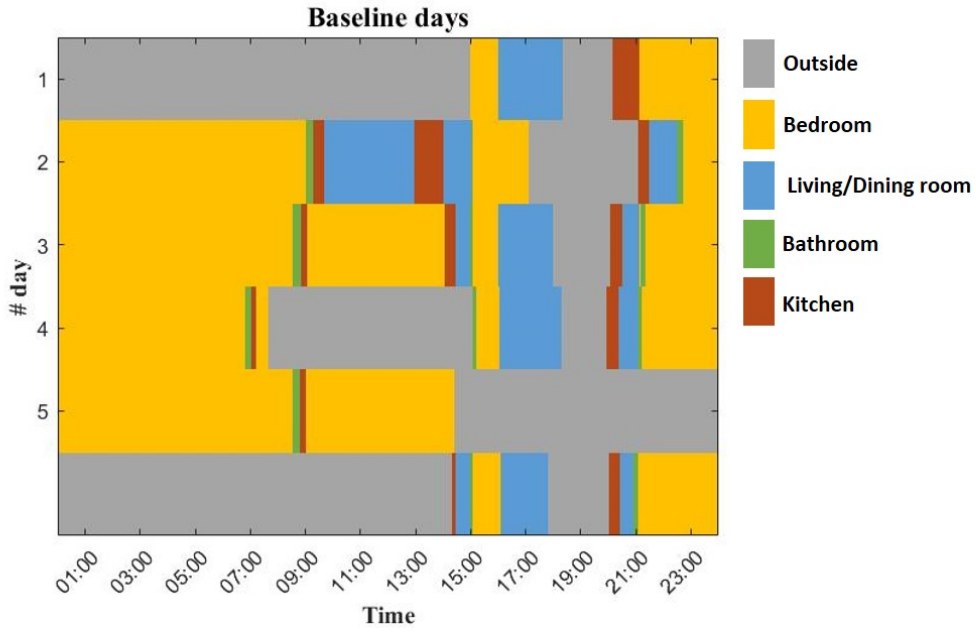


*Figure 17. Baseline days graph*

We identify that 3 baseline days correspond to the independent days we saw visually before (baseline days 1, 5 and 6 correspond to days 1, 7 and 8, respectively). The remaining 3 baseline days correspond to the 3 routines we saw on tables 1, 2 and 3. Specifically, baseline day 2 corresponds to days 2, 3, 9 and 10 (Saturday and Sunday), baseline day 3 corresponds to days 4, 6 and 11 (Monday, Wednesday, Thursday and Friday), and baseline day 4 corresponds to days 5 and 12 (Tuesday). The colours in this graph represent the most likely room the elderly is at each time instant.

## 4.4. Anomaly detection

As we only have 12 days' worth of information, we cannot make a reliable anomaly analysis. Therefore, we have to generate an artificial 120-days routine, which will have a weekly meaning. In this week, we will consider Monday, Tuesday, Wednesday, Thursday and Friday as workdays. On Saturday and Sunday, the volunteer will follow a different routine.

To simulate the weekdays, we will use original routine days 2, 3, 9 and 10 as models. To simulate weekends, we will use original routine days 4, 6 and 11.

In order to check that the anomaly detection algorithm works properly, we will introduce several anomalous days inside both groups, using original routine days 1, 5, 7, 8 and 12. In particular, we introduce anomalous working days in days 5, 52 and 101, and anomalous weekend days in days 20 and 97.

Therefore, the algorithm should generate two groups, one of them corresponding to days from Monday to Friday, and another one for the weekends.

The routine representation of these 120 days is shown in Figure 18.

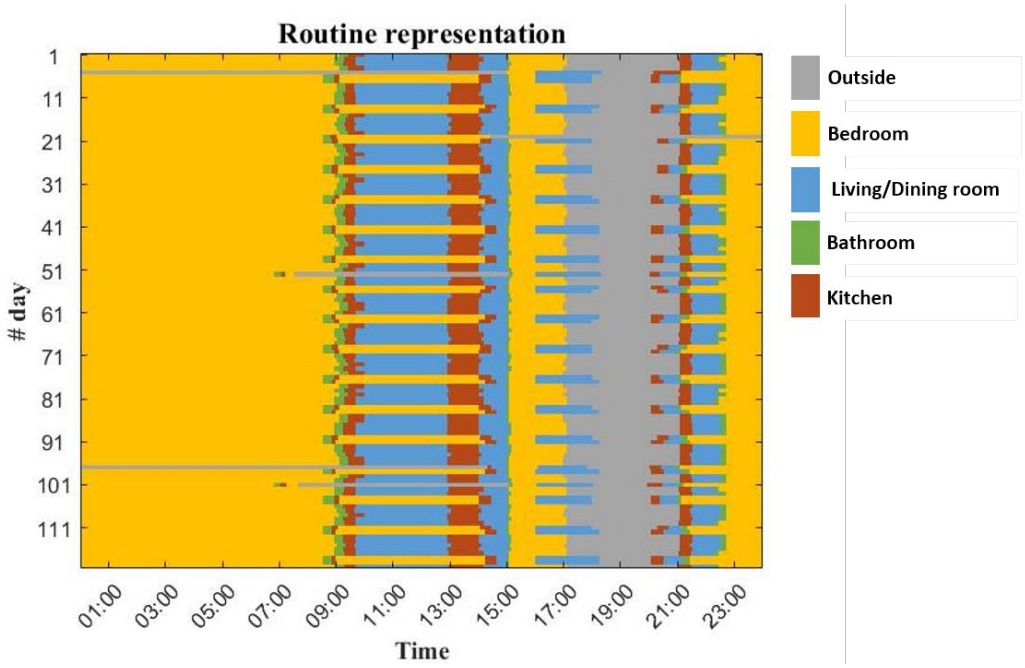


*Figure 18. 120-days routine representation*

Once we have all days identified, we can pass them through the DBSCAN algorithm, so that it classifies them in different clusters. We can see the three created clusters in Figure 19. Cluster 1 corresponds to all weekdays, cluster 2 includes all weekend days, and cluster -1 is made of the outlier samples. Cluster -1 is created automatically by the DBSCAN algorithm when it detects anomalous days.

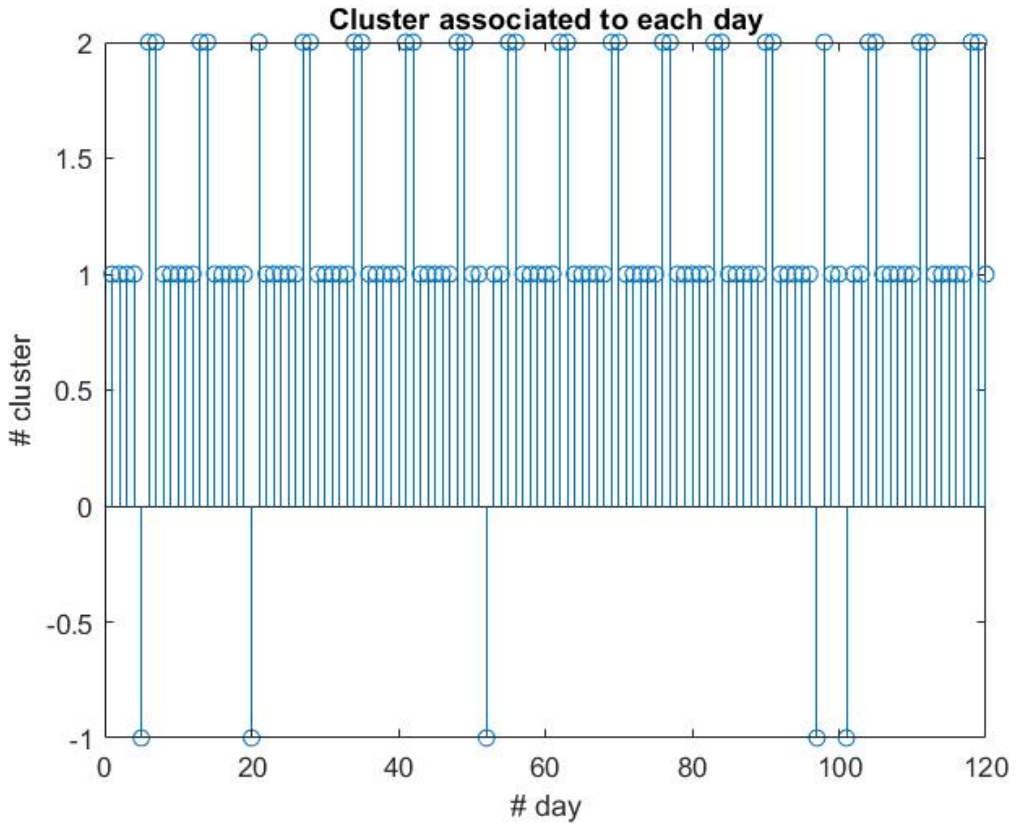


*Figure 19. Cluster associated to each day in the 120-days routine*

We can see that these 2 clusters correspond to two baseline days, which are represented in Figure 20. The first baseline day corresponds to a normal weekday, whereas the second baseline day represents a normal weekend day.

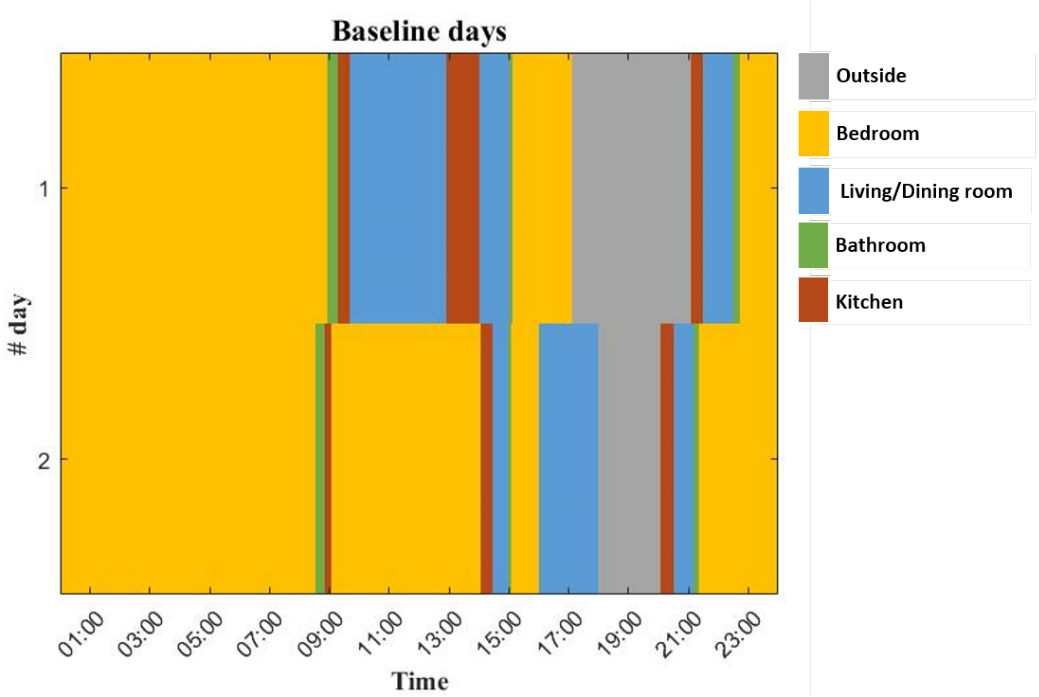


*Figure 20. Baseline days in 120-days routine*

The following step we have to take is the calculation of the mean square error (MSE) between each day and the others. Figure 21 and Figure 22 show a clearer representation of these calculations.

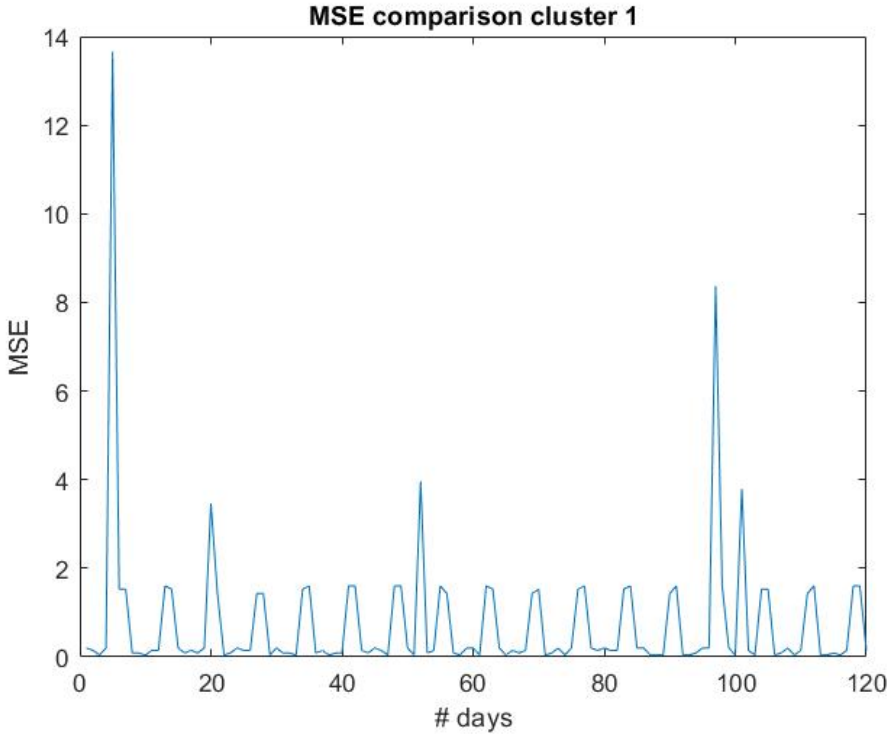


*Figure 21. MSE comparison for cluster 1 in 120-days routine*

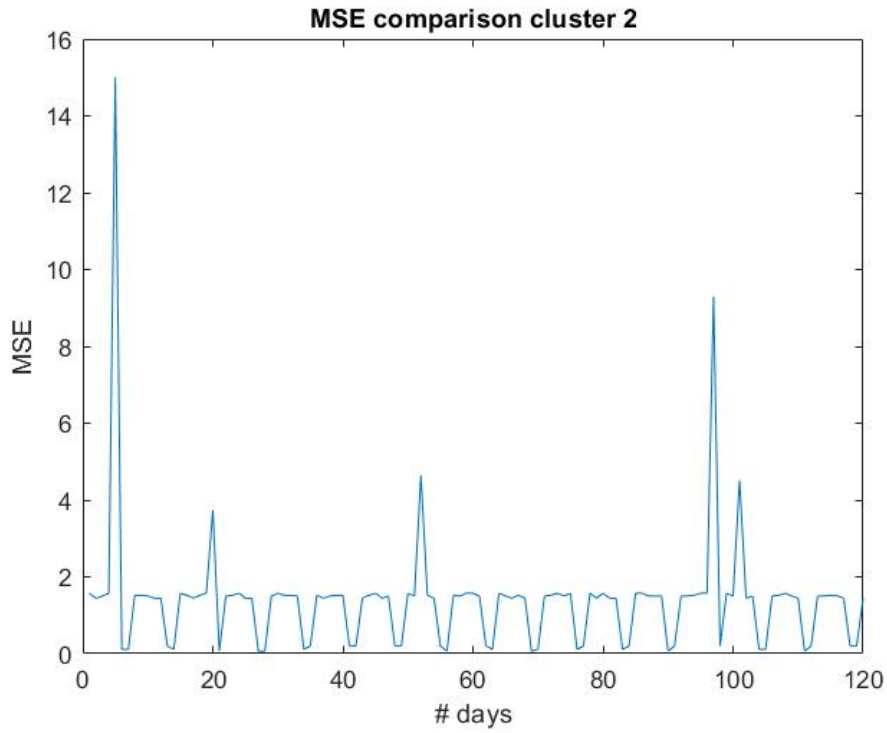


*Figure 22. MSE comparison for cluster 2 in 120-days routine*

In Figure 21, we can see the MSE between the reference day associated to cluster 1 (weekdays) and the rest of the routine days. We can clearly see five peaks, which correspond to the anomalous days, since we know that the higher the MSE the more different the days are between each other. The small periodic peaks correspond to the MSE between the evaluated weekday and a weekend day. This is why they are different, but not completely divergent between each other. The flat areas close to 0 are the weekdays, since the MSE is almost null, meaning that the analysed days are very close to each other.

In Figure 22, we can see the MSE between the reference day associated to cluster 2 (weekend days) and the rest of the routine days. We can observe a similar scenario to the one we analysed in Figure 21. We can see five abrupt peaks, corresponding to the anomalous days. The periodic flat areas with a MSE close to 2 indicate the comparison between the days in cluster 2 and weekdays. The valleys that repeat across the graph are the juxtaposition of the days in cluster 2 and the weekend days. That is why they are very close to 0, since they are the same kind of days, if we refer to the routine the elderly follows.

At the start of this section, we said that we have included five anomalous days inside out 120-days artificial routine (3 weekdays and 2 weekend days). We have already identified these 5 days with the DBSCAN clustering algorithm (they are grouped in cluster -1). Now, we have to detect whether there are more anomalous days or not.

To do so, we have to use the MSE results we have previously obtained. We calculate the moving average and the threshold (the moving average plus the standard deviation). If the moving average is below the threshold, we have a normal day. However, if the moving average is higher than the threshold, we have an anomalous day.

In Figure 23, we can see the anomalous days detector for the weekend days associated to cluster 2 (weekend days cluster). In the Y axis, we have the MSE results for each day. In the X axis, we have the weekend days included in cluster 2. Therefore, day 1 will correspond to the first Saturday, day 2 with the first Sunday, and so on. We only represent the weekend days that are in cluster 2. As we do not see any points where the moving average is higher than the threshold, it means, we do not have any additional weekend anomalous days in our 120-days artificial routine.

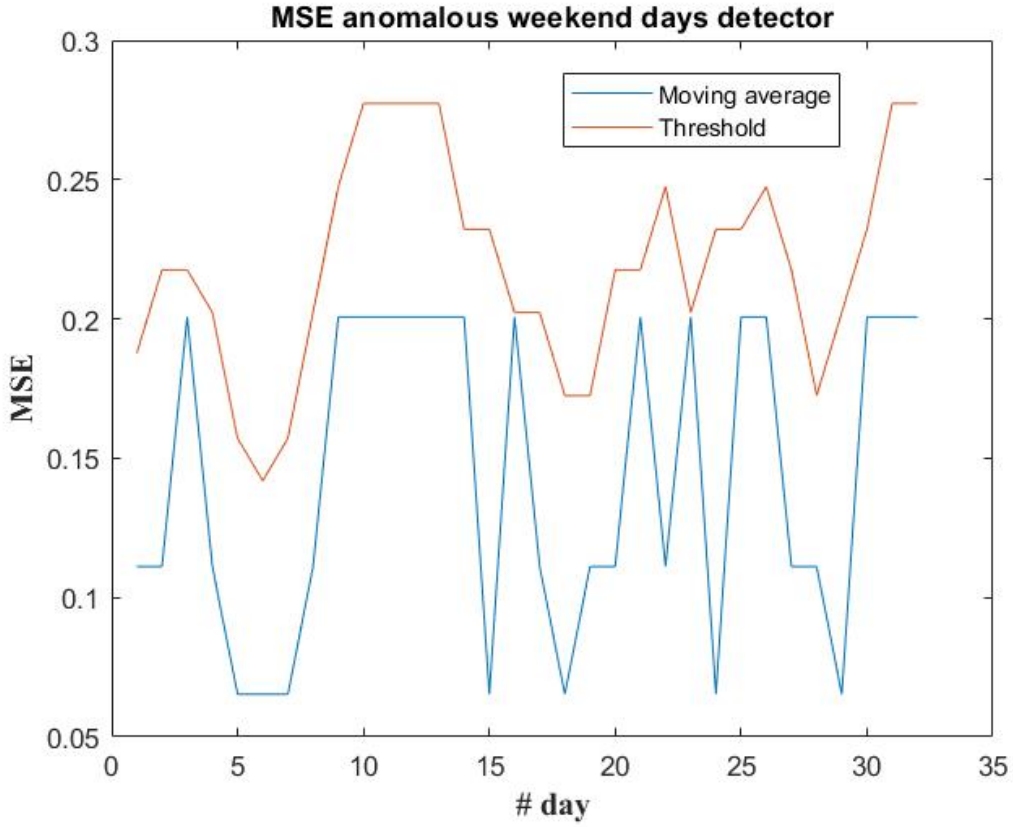


*Figure 23. MSE anomalous weekend days detector for 120-days routine*

In Figure 24, we can see the anomalous days detector for the weekdays associated to cluster 1 (weekdays cluster). Similar to the previous case, the Y axis represents the MSE values for each day, and the X axis shows the weekdays included in cluster 1. In this way, day 1 corresponds to the first Monday, day 2 to the second Monday, and so on. These days do not include the anomalous weekdays, since we only represent the days included inside cluster 1. In contrast to the weekend days case, in Figure 24 we can see that days 35 and 44 have a moving average higher than the threshold. Therefore, we conclude that these two days are also considered anomalous days. In conclusion, we have five anomalous weekdays (3 of them detected with the DBSCAN clustering algorithm, and 2 of them detected with the MSE calculation) in out 120-days artificial routine.

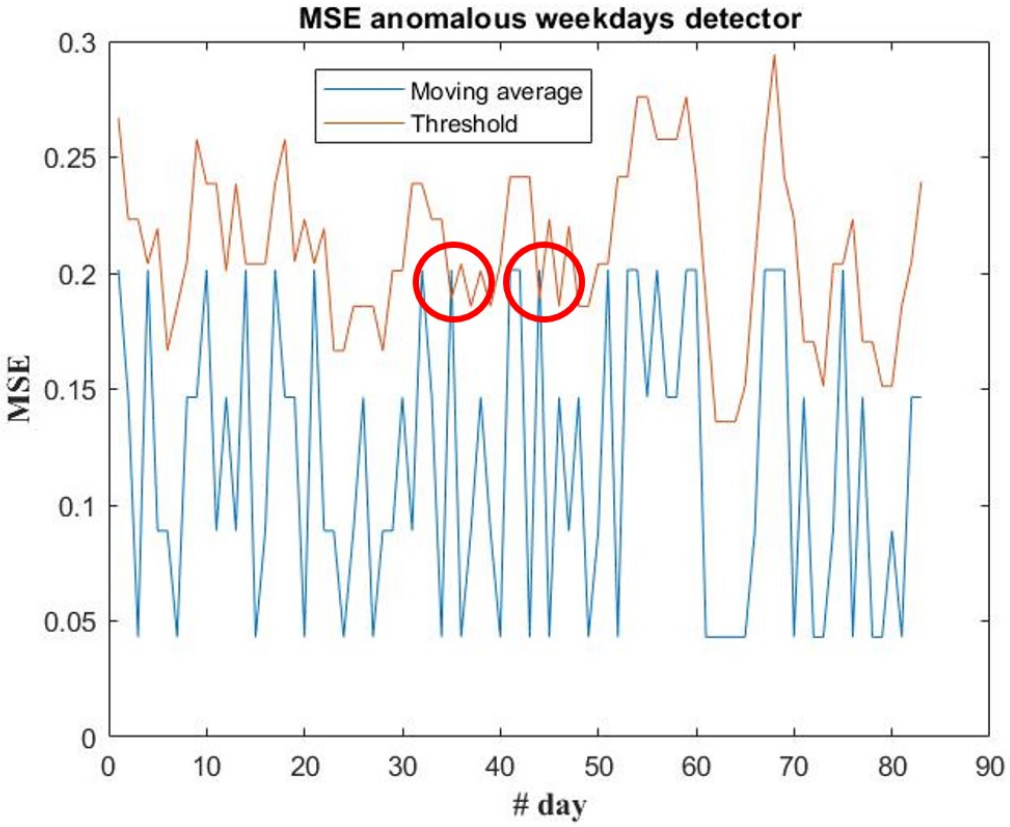


*Figure 24. MSE anomalous weekdays detector for 120-days routine*

# 5. Conclusions and future work

This Degree Final Project has helped us to learn about radiofrequency technologies in the localization area. Additionally, the tests that we have carried out allow us to study the viability of these Bluetooth (BLE) systems in the evaluation and analysis of behavioural routines. In these tests, we monitor a volunteer throughout 12 days, when he does his normal daily activities. These activities are not altered by the testing, since the SensorTile.box that the elderly has to carry with himself is very compact and non-invasive. Additionally, by using the DBSCAN clustering algorithm and the KNN machine learning method, we can identify the routine days and the anomalous days. By detecting these anomalous days, we can analyse what happened that day and carry out an early identification of a possible cognitive of physical problem that the elderly may be suffering.

In future work, the idea is to include other procedures for the evaluation of these behavioural routines, such as the PCA (Principal Component Analysis). Additionally, we also intend to include the Affinity Propagation Algorithm, which will be able to detect several routines automatically.

# 6. Bibliography

[1] Hindawi, "Beacon-Based Time-Spatial Recognition toward Automatic Daily Care Reporting for Nursing Homes". Available online: https://downloads.hindawi.com/journals/js/2018/2625195.pdf (accessed date: July 5th 2022).

[2] MDPI, "Indoor Positioning for Monitoring Older Adults at Home: Wi-Fi and BLE Technologies in Real Scenarios". Available online: https://www.mdpi.com/2079-9292/9/5/728 (accessed date: July 5th 2022).

[3] Hindawi, "Wearable Sensor-Based Human Activity Recognition Using Hybrid Deep Learning Techniques". Available online: https://downloads.hindawi.com/journals/scn/2020/2132138.pdf (accessed date: July 7th 2022).

[4] International Journal of Scientific and Technology Research, "Predicting Indoor Position Using Bluetooth Low Energy and Machine Learning". Available online: http://www.ijstr.org/final-print/sep2019/Predicting-Indoor-Position-Using-Bluetooth-Low-Energy-And-Machine-Learning.pdf (accessed date: July 7th 2022).

[5] World Health Organisation, "Ageing and Health". Available online: https://www.who.int/news-room/fact-sheets/detail/ageing-and-health (accessed date: July 8th 2022).

[6] ELT, "BLE (Bluetooth Low energy)". Available online: https://www.elt.es/ble-bluetooth-low-energy (accessed date: July 10th 2022).

[7] Global Tag, "Tecnología BLE". Available online: https://www.global-tag.com/es/tecnologia-ble/ (accessed date: July 11th 2022).

[8] Novida, "BLE (Bluetooth Low Energy): ¿qué es y cómo usarlo en IoT?". Available online: https://www.novida.com/es/blog/ble/ (accessed date: July 11th 2022).

[9] STMicroelectronics, "STBOX1 Quick Start Guide". Available online: https://www.st.com/content/ccc/resource/sales_and_marketing/presentation/product_presentation/group0/5c/4e/96/c2/a6/98/4a/7f/FP-SNS-STBOX1_Quick_Start_Guide/files/FP-SNS-STBOX1_Quick_Start_Guide.pdf/jcr:content/translations/en.FP-SNS-STBOX1_Quick_Start_Guide.pdf (accessed date: July 13th 2022).

[10] Digikey Electronics, "Meet the new Raspberry Pi 3 model B+". Available online: https://www.digikey.com/en/maker/blogs/2018/meet-the-new-raspberry-pi-3-model-b-plus (accessed date: July 17th 2022).

[11] The Engineering Projects, "Introduction to Raspberry Pi 3 Model B+". Available online: https://www.theengineeringprojects.com/2018/07/introduction-to-raspberry-pi-3-b-plus.html (accessed date: July 17th 2022).

[12] Raspberry, "Raspberry Pi 3 Model B+". Available online: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf (accessed date: July 17th 2022).

[13] IBM, "Machine Learning". Available online: https://www.ibm.com/cloud/learn/machine-learning (accessed date: July 18th 2022).

[14] IBM, "Models for machine learning". Available online: https://developer.ibm.com/articles/cc-models-machine-learning/#reinforcement-learning (accessed date: July 18th 2022).

[15] TechTarget, "Definition: machine learning". Available online: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML (accessed date: July 18th 2022).

[16] expert.ai, "What is Machine Learning? A Definition". Available online: https://www.expert.ai/blog/machine-learning-definition/ (accessed date: July 20th 2022).

[17] techopedia, "Naive Bayes". Available online: https://www.techopedia.com/definition/32335/naive-bayes (accessed date: July 20th 2022).

[18] IArtificial.net, "Random Forest (Bosque Aleatorio): combinando árboles". Available online: https://www.iartificial.net/random-forest-bosque-aleatorio/ (accessed date: July 20th 2022).

[19] JavaPoint, K-Nearest Neighbor (KNN) Algorithm for Machine Learning". Available online: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning (accessed date: July 20th 2022).

[20] KDnuggets, "DBSCAN Clustering Algorithm in Machine Learning". Available online: https://www.kdnuggets.com/2020/04/dbscan-clustering-algorithm-machine-learning.html (accessed date: September 15th 2022).

# APPENDIX 1. Specifications

As we had already specified in Section 3, the materials we are going to use in this project are a SensorTile.box and several Raspberry Pi model 3 B+.

The SensorTile.box has the following characteristics, which have been extracted from its user manual [9].

- STM32L4R9ZIJ6 ultra-low-power, 120 MHz ARM Cortex-M4 microcontroller with
- DSP and FPU and 2048 KB flash memory
- LQFP144 package
- 16 MHz crystal oscillator, 32 kHz crystal oscillator for RTC
- 3 x SPI bus, 3 x I2C bus
- microSD slot
- 3 pushbuttons (BOOT, PWR, USER)
- Digital temperature sensor (STTS751)
- 6-axis inertial measurement unit (LSM6DSOX)
- 3-axis accelerometer (LIS2DW12 and LIS3DHH)
- 3-axis magnetometer (LIS2MDL)
- Altimeter/pressure sensor (LPS22HH)
- Microphone/audio sensor (MP23ABS1)
- Humidity sensor (HTS221)
- Bluetooth connectivity (SPBTLE-1S)
- HCP602535ZC 500mAh, 3.7 V Li-Ion chargeable battery
- STBC02AJR Li-Ion linear battery charger
- STBB3JR 2 MHz DC-DC converter
- FTSH107 connector for SWD debugging and UART Tx/Rx Programming and debugging
- interface (for professional development)
- 57 mm x 38 mm x20 mm IP54 plastic container

The Raspberry Pi model 3 B+ has the following characteristics, which have been extracted from its user manual [15].

- Processor: Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4 GHz
- Memory: 1GB LPDR2 SDRAM
- Connectivity: 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac Wireless LAN, Bluetooth 4.2 BLE, Gigabit Ethernet over USB 2.0 (maximum performance of 300 Mbps), 4 x USB ports 2.0
- Access: 40-pin GPIO header
- Video and sound: 1 x HDMI, DSI display port, MIPI CSI for camera module, stereo audio output and composed video port
- SD card support: MicroSD format to load the operating system and the data storing
- Power supply input: 5 V / 2 A DC through a micro USB connector, 5 V dc through the GPIO header, power supply through Ethernet (PoE) enabled (requires PoE HAT separately)
- Operating temperature: 0 – 50 ºC

Used software:

- PyCharm 2022.1.2
- Anaconda Navigator 2.1.4
- Matlab 2020b
- Microsoft Office 2016
- Windows 10

Used Python packages:

- matplotlib 3.5.3
- numpy 1.23.1
- pandas 1.4.3
- python 3.10.4
- scikit-learn 1.1.1
- sklearn 0.0

# APPENDIX 2. Budget

- Personnel costs. In Table 6, we include the hours spent developing the whole project, both for investigation, algorithm development, test development and documentation development.

| Concept | Worked hours | Price per hour (€) | Cost (€) |
|---|---|---|---|
| Engineering | 250 | 60 | 15000 |
| Writing | 50 | 30 | 1500 |
| **Total** | | | 16500 |

*Table 6. Personnel costs table*

- Software costs. In Table 7, we see the costs derived from the hardware programs necessary to carry out this project.

| Licence | Cost (€) | Payback period (years) | Use (months) | Payback cost (€) |
|---|---|---|---|---|
| Matlab 2020b | 2000 | 2 | 6 | 500 |
| Microsoft Office 2016 | 450 | 2 | 6 | 112.5 |
| Windows 10 | 0 | 2 | 6 | 0 |
| **Total** | 2450 | | | 612.5 |

*Table 7. Software costs table*

- Material costs. In Table 8, we have the costs associated to the equipment we have used in this project.

| Material | Price per unit (€) | Units | Total price (€) | Payback period (years) | Use (months) | Payback cost (€) |
|---|---|---|---|---|---|---|
| Lenovo ideapad 320 | 800 | 1 | 800 | 2 | 6 | 200 |
| Xiaomi Redmi Note 8T | 200 | 1 | 200 | 2 | 6 | 50 |
| ST-Link debugger | 5 | 1 | 5 | 2 | 6 | 1.25 |
| STEVAL-MKSBOX1V1 (SensorTile.box) | 55 | 1 | 55 | 2 | 6 | 13.75 |
| Raspberry Pi 3 model B+ | 195 | 4 | 780 | 2 | 6 | 195 |
| **Total** | | | 1840 | | | 460 |

*Table 8. Material costs table*

- Total costs. In Table 9, we gather the complete costs of the whole project.

| Concept | Total cost (€) |
|---|---|
| Personnel costs | 16500 |
| Software costs | 612.5 |
| Material costs | 460 |
| **Total** | 17572.5 |

*Table 9. Total costs table*

# APPENDIX 3. Codes

## RP_script.py

```
import sys
import os
import time
import struct
import requests
import subprocess
from threading import Thread
from datetime import datetime
from ctypes import (CDLL, get_errno)
from ctypes.util import find_library
from github import Github
from socket import (socket, AF_BLUETOOTH, SOCK_RAW, BTPROTO_HCI, SOL_HCI, HCI_FILTER)
from pydrive.drive import GoogleDrive
from pydrive.auth import GoogleAuth

#Global variables
path = "/home/admin/pi2/"

#Github
mytoken = "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"
myrepo = "ble_tfg"
myfolder = "Home/"

#Get process identifier and store it in pid.txt
pid_file = os.getpid()
try:
    open('pid.txt', 'w').write(str(pid_file) + '\n')
except IOError:
    pass

if not os.geteuid() == 0:
    #sys.exit('script only works as root')
    sys.exit()

#Bluetooth library management
lib_bt = find_library('bluetooth')
if not lib_bt:
    #raise Exception('Cannot find required bluetooth libraries (need to install bluez)')
    a=1
bluez = CDLL(lib_bt, use_errno=True)

dev_id = bluez.hci_get_route(None)

#Receive bluetooth information via socket
socket_bt = socket(AF_BLUETOOTH, SOCK_RAW, BTPROTO_HCI)
sock.bind((dev_id,))

#Configuration of bluetooth scanning parameters
window = 0xC
interval = 0xC
err = bluez.hci_le_set_scan_parameters(sock.fileno(), 0x00, interval, window, 0x00,
0x00, 1000)
if err < 0:
    #raise Exception('Set scan parameters failed')
    a=1
    # occurs when scanning is still enabled from previous call

# allows LE advertising events
hci_filter = struct.pack('<IQH', 0x00000010, 0x4000000000000000, 0)
sock.setsockopt(SOL_HCI, HCI_FILTER, hci_filter)

err = bluez.hci_le_set_scan_enable(
    sock.fileno(),
    1,  # 1 - turn on;  0 - turn off
    0,  # 0-filtering disabled, 1-filter out duplicates
    1000  # timeout
)
```

```
if err < 0:
    errnum = get_errno()
    #raise Exception(f'{os.strerror(errnum)}')

# --------------------------------------------------------------------------------
# Try connect to internet
# Yes -> Return true
# No  -> Return false
# --------------------------------------------------------------------------------
def connected_to_internet(url='http://www.google.com/', timeout=5):
    try:
        _ = requests.head(url=url, timeout=timeout)
        return True
    except requests.ConnectionError:
        #print('No internet connection available.')
        a=1
    return False

# --------------------------------------------------------------------------------
# Upload file to Google Drive and Github
# If file exit in Github -> Error
# --------------------------------------------------------------------------------

def upload_file_GD(arg):
    if connected_to_internet():
        # print(f'Uploading file {arg}')
        authgd = GoogleAuth()
        drivegd = GoogleDrive(authgd)
        filegd = drive.CreateFile({'parents': [{'id': '1T-
gCFikYYnH0XElvjTj5wY75HuJH0f1X'}]})
        filegd.SetContentFile(arg)
        filegd.Upload()


def upload_file_github(mytoken, myrepo, myfolder, myfile_upload):
    git = Github(mytoken)

    repogit = git.get_user().get_repo(myrepo)
    all_files = []
    contents = repogit.get_contents("")
    while contents:
        file_content = contents.pop(0)
        if file_content.type == "dir":
            contents.extend(repogit.get_contents(file_content.path))
        else:
            file = file_content

all_files.append(str(file).replace('ContentFile(path="','').replace('")',''))

    with open(myfile_upload, 'r') as file:
        content = file.read()

    # Upload to github
    git_file = myfolder + myfile_upload
    if git_file in all_files:
        contents = repogit.get_contents(git_file)
        repogit.update_file(contents.path, "committing files", content, contents.sha,
branch="main")
        #print(git_file + ' UPDATED')
    else:
        repogit.create_file(git_file, "committing files", content, branch="main")
        #print(git_file + ' CREATED')

# --------------------------------------------------------------------------------
# Upload files to Google Drive and Github and remove all files uploaded
# If there is no internet connection, the files ar uploaded when it returns
# --------------------------------------------------------------------------------
def upload_files(path):
    if connected_to_internet():
        directory = os.listdir(path)
        for filename in directory:  # for each file
            if filename[0:2] == '20': # is a data file
                upload_file_GD(filename)
                upload_file_github(mytoken, myrepo, myfolder, filename)
                #print("Uploaded file: " + filename)
                os.remove(filename)
                #print("Removed file: " + filename)
```

```
cmd = 'hciconfig'
device_id = 'hci0'
status, output = subprocess.getstatusoutput(cmd)
mac_bt = output.split('{}:'.format(device_id))[1].split('BD Address: ')[1].split('
')[0].strip().replace(':', '')

scan_data = []
current_hour = datetime.now().hour
current_minute = datetime.now().minute

while True: # main loop
    data = sock.recv(1024)
    mac = ':'.join('{0:02x}'.format(x) for x in data[12:6:-1]).upper()
    scan_data.append(f'BLE,{time.time_ns()},{mac},{int(data[-1] - 255)}')

    #Hourly data uploads
    if datetime.now().hour != current_hour:
        filename = f'{time.strftime("%Y_%m_%d_%H_%M_%S")}_{mac_bt}.csv'
        data_str = '\n'.join(scan_data)
        with open(filename, 'w+') as f:
            f.write(data_str)
        current_hour = datetime.now().hour
        current_minute = datetime.now().minute
        scan_data = []

        thread = Thread(target=upload_files, args=(path, ))
        thread.start()
```

## macs.txt

```
B827EB28CF46, dormitorio
B827EBB2A090, comedor
B827EB6E7874, cocina
B827EBD03936, aseo
```

## labels.txt

```
2022-07-08 15:00:00,2022-07-08 16:00:00,dormitorio
2022-07-08 16:00:00,2022-07-08 18:20:00,comedor
2022-07-08 18:20:00,2022-07-08 20:09:00,fuera
2022-07-08 20:09:00,2022-07-08 21:07:00,cocina
2022-07-08 21:07:00,2022-07-08 23:59:59,dormitorio
2022-07-09 00:00:00,2022-07-09 08:56:00,dormitorio
2022-07-09 08:56:00,2022-07-09 09:15:00,aseo
2022-07-09 09:15:00,2022-07-09 09:41:00,cocina
2022-07-09 09:41:00,2022-07-09 12:55:00,comedor
2022-07-09 12:55:00,2022-07-09 13:59:00,cocina
2022-07-09 13:59:00,2022-07-09 15:05:00,comedor
2022-07-09 15:05:00,2022-07-09 15:09:00,aseo
2022-07-09 15:09:00,2022-07-09 17:06:00,dormitorio
2022-07-09 17:06:00,2022-07-09 21:04:00,fuera
2022-07-09 21:04:00,2022-07-09 21:33:00,cocina
2022-07-09 21:33:00,2022-07-09 22:31:00,comedor
2022-07-09 22:31:00,2022-07-09 22:44:00,aseo
2022-07-09 22:44:00,2022-07-09 23:59:59,dormitorio
2022-07-10 00:00:00,2022-07-10 08:57:00,dormitorio
2022-07-10 08:57:00,2022-07-10 09:18:00,aseo
2022-07-10 09:18:00,2022-07-10 09:41:00,cocina
2022-07-10 09:41:00,2022-07-10 12:58:00,comedor
2022-07-10 12:58:00,2022-07-10 14:10:00,cocina
2022-07-10 14:10:00,2022-07-10 15:01:00,comedor
2022-07-10 15:01:00,2022-07-10 15:05:00,aseo
2022-07-10 15:05:00,2022-07-10 17:05:00,dormitorio
2022-07-10 17:05:00,2022-07-10 21:02:00,fuera
2022-07-10 21:02:00,2022-07-10 21:30:00,cocina
2022-07-10 21:30:00,2022-07-10 22:19:00,comedor
2022-07-10 22:19:00,2022-07-10 22:28:00,aseo
2022-07-10 22:28:00,2022-07-10 23:59:59,dormitorio
```

```
2022-07-11 00:00:00,2022-07-11 08:31:00,dormitorio
2022-07-11 08:31:00,2022-07-11 08:54:00,aseo
2022-07-11 08:54:00,2022-07-11 09:05:00,cocina
2022-07-11 09:05:00,2022-07-11 14:02:00,dormitorio
2022-07-11 14:02:00,2022-07-11 14:26:00,cocina
2022-07-11 14:26:00,2022-07-11 15:00:00,comedor
2022-07-11 15:00:00,2022-07-11 15:06:00,aseo
2022-07-11 15:06:00,2022-07-11 16:00:00,dormitorio
2022-07-11 16:00:00,2022-07-11 18:01:00,comedor
2022-07-11 18:01:00,2022-07-11 20:03:00,fuera
2022-07-11 20:03:00,2022-07-11 20:22:00,cocina
2022-07-11 20:22:00,2022-07-11 21:00:00,comedor
2022-07-11 21:00:00,2022-07-11 21:08:00,aseo
2022-07-11 21:08:00,2022-07-11 23:59:59,dormitorio
2022-07-12 00:00:00,2022-07-12 06:48:00,dormitorio
2022-07-12 06:48:00,2022-07-12 07:03:00,aseo
2022-07-12 07:03:00,2022-07-12 07:13:00,cocina
2022-07-12 07:13:00,2022-07-12 07:30:00,dormitorio
2022-07-12 07:30:00,2022-07-12 15:04:00,fuera
2022-07-12 15:04:00,2022-07-12 15:13:00,aseo
2022-07-12 15:13:00,2022-07-12 16:00:00,dormitorio
2022-07-12 16:00:00,2022-07-12 18:18:00,comedor
2022-07-12 18:18:00,2022-07-12 20:01:00,fuera
2022-07-12 20:01:00,2022-07-12 20:23:00,cocina
2022-07-12 20:23:00,2022-07-12 21:05:00,comedor
2022-07-12 21:05:00,2022-07-12 21:22:00,aseo
2022-07-12 21:22:00,2022-07-12 23:59:59,dormitorio
2022-07-13 00:00:00,2022-07-13 08:33:00,dormitorio
2022-07-13 08:33:00,2022-07-13 08:51:00,aseo
2022-07-13 08:51:00,2022-07-13 08:58:00,cocina
2022-07-13 08:58:00,2022-07-13 14:13:00,dormitorio
2022-07-13 14:13:00,2022-07-13 14:38:00,cocina
2022-07-13 14:38:00,2022-07-13 14:59:00,comedor
2022-07-13 14:59:00,2022-07-13 15:06:00,aseo
2022-07-13 15:06:00,2022-07-13 16:01:00,dormitorio
2022-07-13 16:01:00,2022-07-13 18:16:00,comedor
2022-07-13 18:16:00,2022-07-13 20:03:00,fuera
2022-07-13 20:03:00,2022-07-13 20:30:00,cocina
2022-07-13 20:30:00,2022-07-13 21:10:00,comedor
2022-07-13 21:10:00,2022-07-13 21:28:00,aseo
2022-07-13 21:28:00,2022-07-13 23:59:59,dormitorio
2022-07-14 00:00:00,2022-07-14 08:32:00,dormitorio
2022-07-14 08:32:00,2022-07-14 08:48:00,aseo
2022-07-14 08:48:00,2022-07-14 09:00:00,cocina
2022-07-14 09:00:00,2022-07-14 14:25:00,dormitorio
2022-07-14 14:25:00,2022-07-14 23:59:59,fuera
2022-07-15 00:00:00,2022-07-15 14:20:00,fuera
2022-07-15 14:20:00,2022-07-15 14:27:00,cocina
2022-07-15 14:27:00,2022-07-15 14:59:00,comedor
2022-07-15 14:59:00,2022-07-15 15:06:00,aseo
2022-07-15 15:06:00,2022-07-15 16:04:00,dormitorio
2022-07-15 16:04:00,2022-07-15 17:50:00,comedor
2022-07-15 17:50:00,2022-07-15 20:02:00,fuera
2022-07-15 20:02:00,2022-07-15 20:25:00,cocina
2022-07-15 20:25:00,2022-07-15 20:55:00,comedor
2022-07-15 20:55:00,2022-07-15 21:06:00,aseo
2022-07-15 21:06:00,2022-07-15 23:59:59,dormitorio
2022-07-16 00:00:00,2022-07-16 09:07:00,dormitorio
2022-07-16 09:07:00,2022-07-16 09:24:00,aseo
2022-07-16 09:24:00,2022-07-16 09:58:00,cocina
2022-07-16 09:58:00,2022-07-16 12:56:00,comedor
2022-07-16 12:56:00,2022-07-16 14:00:00,cocina
2022-07-16 14:00:00,2022-07-16 14:58:00,comedor
2022-07-16 14:58:00,2022-07-16 15:05:00,aseo
2022-07-16 15:05:00,2022-07-16 16:58:00,dormitorio
2022-07-16 16:58:00,2022-07-16 21:05:00,fuera
2022-07-16 21:05:00,2022-07-16 21:30:00,cocina
2022-07-16 21:30:00,2022-07-16 22:29:00,comedor
2022-07-16 22:29:00,2022-07-16 22:42:00,aseo
2022-07-16 22:42:00,2022-07-16 23:59:59,dormitorio
2022-07-17 00:00:00,2022-07-17 09:02:00,dormitorio
2022-07-17 09:02:00,2022-07-17 09:19:00,aseo
2022-07-17 09:19:00,2022-07-17 09:39:00,cocina
2022-07-17 09:39:00,2022-07-17 13:03:00,comedor
2022-07-17 13:03:00,2022-07-17 14:06:00,cocina
2022-07-17 14:06:00,2022-07-17 15:01:00,comedor
2022-07-17 15:01:00,2022-07-17 15:08:00,aseo
```

```
2022-07-17 15:08:00,2022-07-17 17:01:00,dormitorio
2022-07-17 17:01:00,2022-07-17 21:04:00,fuera
2022-07-17 21:04:00,2022-07-17 21:27:00,cocina
2022-07-17 21:27:00,2022-07-17 22:30:00,comedor
2022-07-17 22:30:00,2022-07-17 22:44:00,aseo
2022-07-17 22:44:00,2022-07-17 23:59:59,dormitorio
2022-07-18 00:00:00,2022-07-18 08:31:00,dormitorio
2022-07-18 08:31:00,2022-07-18 08:50:00,aseo
2022-07-18 08:50:00,2022-07-18 09:03:00,cocina
2022-07-18 09:03:00,2022-07-18 14:04:00,dormitorio
2022-07-18 14:04:00,2022-07-18 14:26:00,cocina
2022-07-18 14:26:00,2022-07-18 14:57:00,comedor
2022-07-18 14:57:00,2022-07-18 15:05:00,aseo
2022-07-18 15:05:00,2022-07-18 16:00:00,dormitorio
2022-07-18 16:00:00,2022-07-18 18:01:00,comedor
2022-07-18 18:01:00,2022-07-18 20:17:00,fuera
2022-07-18 20:17:00,2022-07-18 20:42:00,cocina
2022-07-18 20:42:00,2022-07-18 21:07:00,comedor
2022-07-18 21:07:00,2022-07-18 21:21:00,aseo
2022-07-18 21:21:00,2022-07-18 23:59:59,dormitorio
2022-07-19 00:00:00,2022-07-19 06:47:00,dormitorio
2022-07-19 06:47:00,2022-07-19 07:02:00,aseo
2022-07-19 07:02:00,2022-07-19 07:15:00,cocina
2022-07-19 07:15:00,2022-07-19 07:38:00,dormitorio
2022-07-19 07:38:00,2022-07-19 15:05:00,fuera
2022-07-19 15:05:00,2022-07-19 15:12:00,aseo
2022-07-19 15:12:00,2022-07-19 16:02:00,dormitorio
2022-07-19 16:02:00,2022-07-19 18:03:00,comedor
2022-07-19 18:03:00,2022-07-19 19:56:00,fuera
2022-07-19 19:56:00,2022-07-19 20:27:00,cocina
2022-07-19 20:27:00,2022-07-19 21:03:00,comedor
2022-07-19 21:03:00,2022-07-19 21:14:00,aseo
2022-07-19 21:14:00,2022-07-19 23:59:59,dormitorio
```

University of Alcalá

Polytechnic School



ESCUELA POLITECNICA SUPERIOR

Universidad de Alcalá