Real-Time Human Activity Recognition on Embedded Equipment: A Comparative Study

Houda Najeh, Christophe Lohr, Benoit Leduc

applied sciences

MDPI

Article

Real-Time Human Activity Recognition on Embedded Equipment: A Comparative Study

Houda Najeh [1,2,*], Christophe Lohr [1] and Benoit Leduc [2]

[1] Lab-STICC, IMT Atlantique, 29200 Brest, France; christophe.lohr@imt-atlantique.fr
[2] Delta Dore Company, 35270 Bonnemain, France; bleduc@deltadore.com
* Correspondence: houda.najeh@imt-atlantique.fr

Abstract: As living standards improve, the growing demand for energy, comfort, and health monitoring drives the increased importance of innovative solutions. Real-time recognition of human activities (HAR) in smart homes is of significant relevance, offering varied applications to improve the quality of life of fragile individuals. These applications include facilitating autonomy at home for vulnerable people, early detection of deviations or disruptions in lifestyle habits, and immediate alerting in the event of critical situations. The first objective of this work is to develop a real-time HAR algorithm in embedded equipment. The proposed approach incorporates the event dynamic windowing based on space-temporal correlation and the knowledge of activity trigger sensors to recognize activities in the case of a record of new events. The second objective is to approach the HAR task from the perspective of edge computing. In concrete terms, this involves implementing a HAR algorithm in a "home box", a low-power, low-cost computer, while guaranteeing performance in terms of accuracy and processing time. To achieve this goal, a HAR algorithm was first developed to perform these recognition tasks in real-time. Then, the proposed algorithm is ported on three hardware architectures to be compared: (i) a NUCLEO-H753ZI microcontroller from ST-Microelectronics using two programming languages, C language and MicroPython; (ii) an ESP32 microcontroller, often used for smart-home devices; and (iii) a Raspberry-PI, optimizing it to maintain accuracy of classification of activities with a requirement of processing time, memory resources, and energy consumption. The experimental results show that the proposed algorithm can be effectively implemented on a constrained resource hardware architecture. This could allow the design of an embedded system for real-time human activity recognition.

Keywords: real-time human activity recognition; embedded equipment; smart home; home automation sensors; edge computing; resource-constrained devices

check for updates

Citation: Najeh, H.; Lohr, C.; Leduc, B. Real-Time Human Activity Recognition on Embedded Equipment: A Comparative Study. *Appl. Sci.* **2024**, *14*, 2377. https://doi.org/10.3390/app14062377

Academic Editors: Gang Wei, Luigi Bibbò and J. Artur Serrano

Received: 11 January 2024
Revised: 26 February 2024
Accepted: 28 February 2024
Published: 12 March 2024

1. Introduction

Based on sensor data, human activity recognition (HAR) focuses on detecting automatically what an occupant is doing. It is at the core of various applications such as smart homes [1,2], assistance of elderly people [3–7], and industrial applications such as Human–Robot applications [8].

In many real-time scenarios, human activities must be tracked online, in real-time. This requires that the HAR frameworks use only pre-segmented datasets. Segmentation consists of dynamically dividing the sensor stream over time into segments, forming a set of sensor events associated with one activity; which is complicated in real-time. However, human activities must be tracked online with streaming data to provide real-time services or scenarios. Related works in this research field are relatively few. For example, Ref. [9] investigated a dynamic segmentation technique of sensor events. The study used two scenarios, non-overlapping and overlapping time windows. Authors in [10] proposed another dynamic segmentation technique on streaming data based on semantic analysis and statistical learning. In order to achieve the goal of dynamic adaptation, this method studies the input event sequence and selects the more appropriate time-window size. The work presented in [11] proposed a method of dynamic segmentation on streaming data that integrated time correlation and event correlation and performs real-time HAR on streaming data. In [11], a real-time HAR algorithm in smart homes is developed. It is based on real-time dynamic segmentation that determines the beginning and the end of each activity and the classification using the concept of triggering sensor and the habitual duration of activities. The work reported in this paper is a continuity of the work presented in [11]. The main contributions of this work are:

- Demonstrate that, through the dynamic sensor windowing and the identification of the triggering sensor, we can improve the classification of activities.
- Test the efficiency of the HAR algorithm in terms of processing time, memory usage, and current consumption in three hardware architectures and prove that the proposed framework can be implemented in a low-power, low-cost device while preserving good performance in terms of classification accuracy.

This paper is organized as follows: The related works are summarized in Section 2. The proposed methodology is detailed in Section 3. The methodology of implementation on embedded equipment and the obtained results are discussed in Sections 4 and 5, respectively. Finally, concluding remarks are given in Section 6.

2. Related Works

This section presents a summary of the most recent HAR techniques on embedded equipment.

In [12], a machine learning (ML) and data fusion technique applied to physical activity classification using inertial measurement unit signals is adopted with an accuracy of classification of 96.42%, and the algorithm is implemented both on Intel i7 CPU and NVIDIA 1080 GPU. The authors in [13] used an accelerometer-based method for human activity recognition and implemented their work on the CPU with an accuracy rate of 78.00%. A biometric user identification based on human activity recognition using wearable sensors is investigated in [14]. The framework uses deep learning models and is implemented on: (1) Intel i5-8400 CPU with an Accuracy of 91.23% and (2) NVIDIA RTX2070 GPU with an Accuracy of 85.57%.

The deployment of deep neural networks on microcontrollers is adopted in [15]. A CNN-based approach is implemented on Nucleo-L452RE-P and SparkFun Edge board (ARM Cortex-M4F) with an accuracy rate of 92.46%. Also, a Recurrent Neural Network is used for human activity recognition in STM32L476RG microcontroller (ARM Cortex-M4F) using accelerometer data with an accuracy rate of 95.54%. In [16], a deep residual bidir-LSTM for human activity recognition using wearable sensors is developed and implemented on NVIDIA GTX 960M GPU with an accuracy of classification of 93.54%. A lightweight deep learning model for human activity recognition on edge devices is investigated and implemented on Raspberry Pi in [17] with 95.78% accuracy. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition in [18]. Table 1 summarizes the state-of-the-art HAR methods.

Table 1. State of the art HAR techniques on embedded equipment: a performance study.

Reference	Algorithm	Performance	Hardware Architecture
[12]	ML & data fusion	96.42%	Intel i7 CPU and NVIDIA 1080 GPU
[13]	accelerometers based method	78.00%	CPU
[14]	biometric user identification	91.23% 85.57%	Intel i5-8400 CPU NVIDIA RTX2070 GPU

Table 1. *Cont.*

Reference	Algorithm	Performance	Hardware Architecture
[15]	CNN	92.46%	Nucleo-L452RE-P SparkFun Edge board (ARM Cortex-M4F)
[19]	RNN & LSTM	95.54%	STM32L476RG (ARM Cortex-M4F)
[16]	LSTM	93.54 %	Intel-i7 CPU, NVIDIA GTX 960M GPU
[17]	RNN and LSTM	95.78 %	Raspberry Pi 3 (ARM Cortex A53)
[18]	LSTM	86.40%	GPU

The authors in [20] used HAR and embedded applications based on the Convolutional Neural Network. A stochastic gradient descent algorithm is used to optimize the parameters of the CNN and the trained network model is compressed on STM32CubeMX-AI. In [21], a HAR using a machine learning technique in a low-resource embedded system is investigated. The authors present a prototype of a wearable device that identifies a person's activity: walking, running, or staying still. The system consists of a Texas Instruments MSP-EXP430G2ET launchpad, connected to a BOOSTXL-SENSORS booster pack with a BMI160 accelerometer. A Recurrent Neural Network for HAR in Embedded Systems Using PPG and Accelerometer Data is developed in [19]. The aim of this paper is to address the human activity recognition (HAR) task directly on the device, implementing a recurrent neural network (RNN) in a low-cost, low-power microcontroller, ensuring the required performance in terms of accuracy and low complexity. [22] introduced a novel approach to HAR by presenting a sensor that utilizes a real-time embedded neural network. The sensor incorporates a low-cost microcontroller and an inertial measurement unit (IMU), which is affixed to the subject's chest to capture their movements. Most existing methods in the literature for activity recognition are based on accelerometer wearable sensors. They demonstrate that it is possible to execute HAR AI algorithms on constrained computers. The novelty in this work consists of implementing a non-based AI activity recognition algorithm based on home automation sensor data on embedded equipment.

3. Proposed Framework for Real-Time HAR

This section outlines the real-time human activity recognition (HAR) framework, comprising three essential stages: (1) the identification of optimal features; (2) a dynamic segmentation method in real-time, which discerns the initiation and conclusion of each activity segment; and (3) a classification step. Prior to these stages, there is an initial data pre-processing step (i.e., cleaning, filtering, and standardizing the raw sensor data before feature extraction and analysis).

The selection of a pertinent sensor configuration is an important aspect of HAR. In fact, some of them are not relevant or redundant for activities recognition. The objective of selecting a minimum sensor configuration is not to exclude the other sensors from the house but to filter the sensors as input to the real-time HAR algorithm so that it goes faster with an acceptable compromise on recognition performance, but with a performance gain in terms of real-time processing time. In this work, the minimum sensor setup is selected using the mutual information criterion proposed by [23].

Thus, one of the challenges is how to split the sensor sequence into a series of event chunks that are associated with corresponding activities. In [11], the dynamic segmentation of events is proposed. It is determined using the Pearson product moment correlation (PMC) coefficient between the events [24]. PMC, or more formally $\rho_{X,Y}$, is a measure of the linear correlation between two variables, X and Y (two events in our case study). It is calculated using Equation (1).

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{1}$$

where: $cov(X,Y)$, σ_X and σ_Y represent, respectively, the covariance of X and Y are the standard deviation of X and Y, respectively. It produces a value between -1 and 1. Three cases are distinguished:

- X and Y are totally positively correlated if $\rho_{X,Y} = 1$
- X and Y are totally negative correlated if $\rho_{X,Y} = -1$
- X and Y are no correlated if $\rho_{X,Y} = 0$

The aim of this work is to implement trade-offs that enable real-time activity recognition on constrained embedded hardware. We seek to maintain acceptable performance in terms of decisions while being moderate in terms of consumption of computing resources. The work described in [11] precisely proposes to simplify one step: real-time segmentation, with the general idea of relying on the "usual correlations" between events within activities. The results show that the proposed approach segments the flow of events well and is even able to guess the desired activities (Sleeping, Bed to toilet, Meal preparation, Eating, Housekeeping, Working, Enter home, and Leave home) with an F1 score of 0.63 to 0.99. Note that in our context, "real-time" means being able to make a decision before the next event occurs. For this, the minimization of the setup described in [23] further simplifies the calculations.

The proposed framework is visually depicted in Figure 1. These steps are presented sequentially in the following paragraphs.

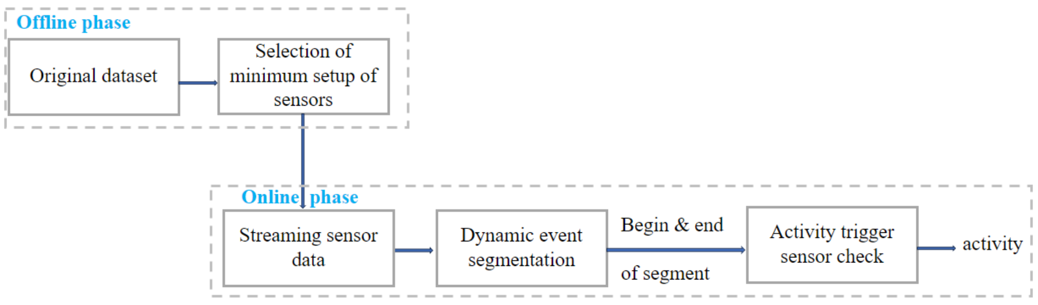

Offline phase
Original dataset
Selection of minimum setup of sensors
Online phase
Streaming sensor data
Dynamic event segmentation
Begin & end of segment
Activity trigger sensor check
activity

Figure 1. Proposed framework for human activities estimation.

The subsequent phase of the proposed framework (depicted in Figure 1) involves the execution of recognition. This study employs real-time dynamic segmentation on streaming data as part of the recognition step, incorporating spatial correlation between events. The method assesses whether two consecutive sensor events should be categorized within the same activity segment, preventing events from markedly distinct zones from being grouped together. This technique, detailed in [11], facilitates the determination of the beginning and end of each segment upon the addition of new events, simultaneously providing the labeled identification of the recognized activity.

The notion of dynamic segmentation operates as follows: for each incoming event, the inquiry is whether it belongs to the current segment or marks the initiation of a new segment. This determination relies on the calculation of the Pearson Product Moment Correlation (PMC) coefficient, as detailed in [24], measuring the linear correlation between two sensor events.

Figure 2 illustrates an example of the identification of the beginning and the end of activities' segments, i.e., how to process the real-time dynamic segmentation.

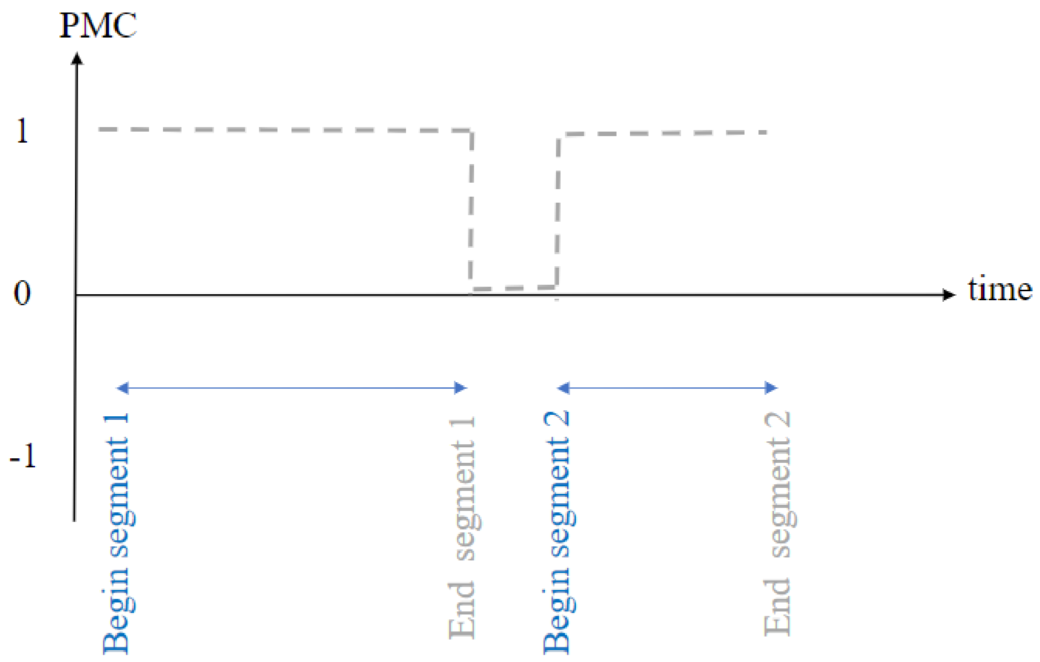


Figure 2. Real-time dynamic segmentation.

Once the correlation between events is established, the ongoing assessment for each incoming event involves monitoring whether the correlation consistently equals 1. If the correlation remains constant at 1, it indicates that the event is part of the current segment. However, when the correlation deviates from 1, the last sample signifies the end of the segment. Once the correlation returns to 1, this juncture marks the inception of a new segment. In the following, this methodology is applied to a real case study.

4. Implementation on Embedded Equipment

This section discusses the performance of the implementation of the following framework in terms of processing time, memory management, and power consumption. The implementation of hardware needs: (1) A study of data volume ratio (Section 4.2); (2) A study of the difference between the architectures (Section 4.3); (3) A study of different tools of Performance Metrics Measurement. Before detailing the methodology of implementation, let us introduce the test bed and dataset.

4.1. Test Bed and Dataset Description

The case study is conducted using the Aruba dataset, which captures human activities recorded in a smart apartment managed by the Center for Advanced Studies in Adaptive Systems (CASAS) [25]. This apartment is inhabited by an adult woman and experiences regular visits from her children and grandchildren throughout the year. The period of the test is two years, from 4 November 2010 to 30 June 2011. The layout of the home and the placement of sensors are illustrated in Figure 3.

The sensor network setup comprises 31 motion detection sensors, 5 temperature sensors, and 3 door contact sensors, each identified by prefixes "M", "T", and "D", respectively. Among many constructed features, some of them are not relevant depending on the considered estimation. The objective of selecting a minimum sensor configuration is not to exclude the other sensors from the house. The objective is to filter the sensors so that the algorithm goes faster with an acceptable compromise on recognition performance but with a performance gain in terms of real-time processing time. In this study, we focus on a minimal sensor configuration, selecting 11 sensors with the following identifiers: M003, M004, M009, M013, M014, M015, M017, M020, M026, M030, and D004. The data are sampled with a sampling interval of 1 s and a sampling frequency of 1 Hz. This choice is motivated by the consideration that increasing the sampling rate results in information loss. Therefore, a smaller sampling time enriches the segment with more events, enhancing the accuracy of the segmentation process. An extract from the dataset is shown in Figure 4.

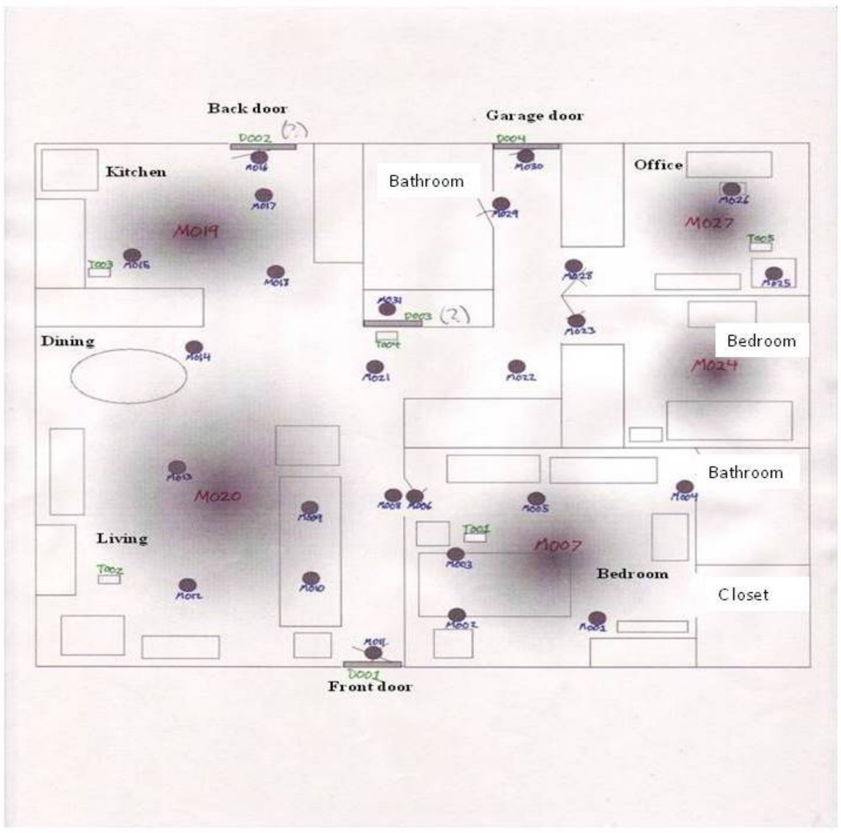
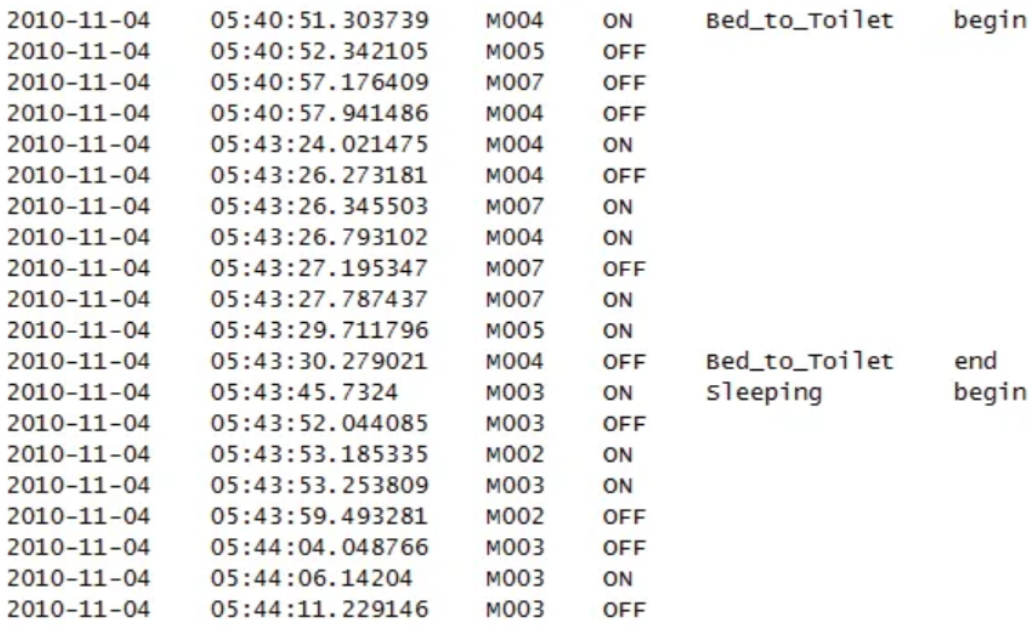


Figure 3. Aruba test bed.



```
2010-11-04    05:40:51.303739    M004    ON     Bed_to_Toilet    begin
2010-11-04    05:40:52.342105    M005    OFF
2010-11-04    05:40:57.176409    M007    OFF
2010-11-04    05:40:57.941486    M004    OFF
2010-11-04    05:43:24.021475    M004    ON
2010-11-04    05:43:26.273181    M004    OFF
2010-11-04    05:43:26.345503    M007    ON
2010-11-04    05:43:26.793102    M004    ON
2010-11-04    05:43:27.195347    M007    OFF
2010-11-04    05:43:27.787437    M007    ON
2010-11-04    05:43:29.711796    M005    ON
2010-11-04    05:43:30.279021    M004    OFF    Bed_to_Toilet    end
2010-11-04    05:43:45.7324      M003    ON     Sleeping         begin
2010-11-04    05:43:52.044085    M003    OFF
2010-11-04    05:43:53.185335    M002    ON
2010-11-04    05:43:53.253809    M003    ON
2010-11-04    05:43:59.493281    M002    OFF
2010-11-04    05:44:04.048766    M003    OFF
2010-11-04    05:44:06.14204     M003    ON
2010-11-04    05:44:11.229146    M003    OFF
```

Figure 4. An extract from the Aruba dataset.

The final two columns in the dataset provide details regarding the activity name and its corresponding segment, defined by both its start and end times.

The dataset contains 11 distinct daily activities, namely: Sleeping (401 occurrences), Bed to Toilet (157 occurrences), Wash Dishes (65 occurrences), Relax (2910 occurrences), Work (171 occurrences), Housekeeping (33 occurrences), Meal Preparation (1606 occurrences), Eating (257 occurrences), Leave Home (431 occurrences), and Enter Home (431 occurrences). The numerical values in parentheses signify the frequency of each activity in the dataset. The activities were manually annotated by the occupant.

4.2. Data Volume Ratio

Implementing algorithms on embedded systems demands detailed measurement and management of datetime format volume ratios. Crucial points include accurately handling datetime formats, ensuring consistency across various systems and programming languages, and using comparable tools. The volume of data involving datetime formats is equally vital, especially when working with large datasets, impacting algorithm efficiency and performance.

Table 2 summarizes the in-memory size measurements (in octets) of string datetimes and datetime objects across different platforms, including MicroPython on ESP32, MicroPython on STM32, Python on a Windows machine, and C language on STM32, as obtained from the Aruba dataset.

Table 2. Measurement of in-memory size of string datetimes in Aruba dataset (obtained results).

	String Datetime	**Datetime Object**
MicroPython on ESP32	16 octets	16 octets
MicroPython on STM32	16 octets	16 octets
Python on Windows machine	68 octets	48 octets
C language on STM32	20 octets	36 octets

The study focuses on the Aruba dataset, where raw datetimes use string datetime format. The implementation's initial step involves examining volume ratios between datetime object and string datetime formats in Python and C. In Python, datetime handling relies on the datetime module, providing an intuitive syntax. Conversely, C requires structures like `struct tm` and $time_t$. Comparative results reveal size equality for string and object datetimes in MicroPython on ESP32 and STM32. However, C implementation shows a smaller memory footprint for string datetime formats, emphasizing the influence of the execution platform on memory size. This compromise between memory efficiency and language-specific conventions underscores the complexity of managing datetime representations across programming languages. The subsequent adoption of the string datetime format for simulation reflects the nuanced considerations in balancing memory efficiency and language-specific nuances during algorithm implementation.

4.3. Difference between Architectures

The selection of devices aims to represent a diverse set of edge computing platforms, each with its unique strengths and use cases. The Raspberry Pi was selected for its versatility and computational power. As a single-board computer, it provides a Linux-based environment and is well-suited for running complex algorithms and applications. Its support for a wide range of peripherals and connectivity options makes it a popular choice for various IoT and edge computing projects. The STM32 microcontroller represents the category of microcontrollers commonly used in embedded systems. The STM32H753ZI microcontroller features 3 user LEDs, 2 user and reset push-buttons, and a 32.768 kHz crystal oscillator. The board connectors include SWD, ST Zio expansion connector with ARDUINO® Uno V3 compatibility, and ST morpho expansion connector. Flexible power-supply options encompass ST-LINK USB VBUS, USB connector, or external sources. It offers comprehensive free software libraries and examples through the STM32Cube MCU

Package and supports a wide range of Integrated Development Environments (IDEs) such as STM32CubeIDE. Various USB modes are supported such as USB Devices. Board connectors may include MIPI20 compatibility, USB with Micro-AB or USB Type-C®, and Ethernet RJ45. Furthermore, on-board ST-LINK (STLINK/V2-1, STLINK-V3E, or STLINK-V3EC) debugger/programmer with USB re-enumeration capability facilitates mass storage, Virtual COM port, and debug port functionalities, enhancing the development process. Its real-time capabilities and low power consumption make it suitable for applications where resource efficiency is critical.

The architecture of the Espressif ESP32 DevKit V4 centers around its core component, the ESP32-WROOM-32 module. This module integrates essential features such as the EN reset button and boot download button, which initiates firmware download mode for serial port firmware downloads. The board is equipped with a USB-to-UART bridge chip, enabling high-speed data transfer of up to 3 Mbps through the micro USB port. A 5V Power On LED indicates when the board is powered either through USB or an external 5V power supply. The board's I/O functionality is extensive, with most pins broken out to pin headers, allowing for versatile programming options including PWM, ADC, DAC, I2C, I2S, and SPI. Users have three power supply options: the default micro USB port, 5V and GND header pins, and 3V3 and GND header pins, providing flexibility in powering the board according to specific project requirements. The ESP32 was chosen for its low cost, low power consumption, and integration of wireless connectivity features, which contribute to its popularity in a variety of edge computing applications.

Table 3 details the characteristics of each architecture.

Table 3. Characteristics of hardware architectures.

Hardware Architecture	Characteristics
NUCLEO-H753ZI	• Arm Cortex-M7 Core at 480 MHz • 2 Mbytes of Flash memory • 1 Mbyte of SRAM • Voltage at device input = 3.3 V • Cost: 34 euros • Programmed in C or C++ using tools such as STM32CubeIDE.
ESP32 Devkit V4	• 32-bit Xtensa dual-core 240 MHZ • 520 KB SRAM of SRAM (16 KB for Cache) • 448 KB of ROM • Cost: 12 euros • Supports programming in C or C++ using the ESP-IDF, as well as MicroPython.
Raspberry-Pi3 B+	• 64-bit quad-core ARM Cortex-A53 processor at 1.4 GHz • 1 GB of RAM • Cost: 59 euros • Supports a variety of programming languages, including Python, C, C++, and others. Typically programmed using a Linux-based environment.
desktop with Linux OS	• Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60 GHz • 131,788,880 kB of total memory • CPU family: 6 • Model: 63

Regarding the processing time, there is a constraint to respect in this work, which is the following. Since an industrial system is multitasking, the processing time necessary to classify an activity must be less than the time between two events in the dataset. A statistical study has shown that the time delta between events is equal to 10 s for a complete configuration of sensors, while it is equal to 17 s with a minimum configuration of sensors.

4.4. Performance Metrics and Measuring Tools

Once the algorithm is implemented on the device, it is essential to evaluate its performance in terms of processing time, memory usage, and energy consumption. This section details some common techniques and tools we can use for each of these metrics. Before detailing the tools of measurement, let us define each metric as follows.

4.4.1. Definitions

- **Processing time**: Also known as execution time, refers to the amount of time it takes for a program, or a specific task to complete its execution. It is a measure of how long a particular operation or set of operations takes to run from start to finish. In this work, the processing time is related to how long time the tasks of sampling, segmentation, and classification take to run.
  In the context of programming and hardware platforms, the calculations must be completed before a new event occurs. It is therefore a question of comparing times.
- **Memory usage**: Refers to the amount of computer memory (RAM or Random Access Memory) that a program is using during its execution. Memory is a crucial resource in computing, and efficient memory usage is essential for optimal performance. Here are key points related to memory usage:
  - *RAM (Random Access Memory)*: This is the volatile memory that a computer uses to store data that is actively being used or processed. It is faster than long-term storage (like hard drives or SSDs) but is volatile, meaning it loses its contents when the power is turned off.
  - *Memory Allocation*: Programs dynamically allocate and deallocate memory as needed. Inefficient memory allocation or memory leaks can lead to increased memory usage over time. There is an important difference from one language to another: in C the programmer must manage this allocation/deallocation himself (more complicated, risk of error, but complete mastery); unlike Python where there is a garbage collector (fewer errors, but potentially not optimal).
  - *Memory Profiling*: Techniques and tools such as memory profiling are used to measure and analyze how a program uses memory. This is crucial for identifying potential memory leaks and optimizing memory usage.
- **Energy consumption**: Refers to the amount of electrical energy that a device or system consumes during its operation. It is a critical consideration in modern computing, particularly in portable devices and data centers. These machines sometimes have mechanisms to vary the processor frequency (and therefore consumption) depending on the calculation load.

4.4.2. Tools of Measurement of Metrics

Table 4 summarizes the common techniques to measure the cited metrics.

Table 4. Common techniques and tools.

Metric	Tools of Measurement
Processing time	Tools like "timeit" in Python measure execution time for specific code snippets
Memory Usage	Memory Profiling Librarie such as memory profiler in Python
Energy Consumption	hardware-specific tools or external devices For example, the Intel Power Gadget for Intel CPUs

These performances are studied and interpreted in detail in Section 5.

5. Results of Implementation

This section deals with the results of the classification of activities using the proposed algorithm (Section 5.1) and the results of implementation on a Raspberry Pi and comparison with desktop PC (Section 5.2), on STM32 and ESP32 with, respectively, MicroPython and C programming languages (in Sections 5.3 and 5.4, respectively). Once the performance is highlighted, the limits of microcontrollers regarding sampling data and activity recognition are studied in Section 5.5.

The average time between two events is equal to 10 s with a complete sensor configuration, compared to 17 s with a minimum sensor configuration. In this work, two periods of test are concerned:

- Period 1: which corresponds to the period of the long activity in the dataset.
- Period 2: which corresponds to the period of maximum number of records that the microcontrollers support to run the algorithm. Before determining this period, we first study the memory capacity of the device in relation to the quantity of data provided (i.e., the number of "timestamps-sensor names and associated values" recordings with which the algorithm can run on the device). Depending on the number of allocated records that constitute the second test period, we run the algorithm with this quantity of data and calculate the processing time used.

5.1. Results of Human Activity Recognition

The initial phase in the proposed methodology involves the real-time dynamic segmentation method, as detailed in [11], employing a dynamic window size. Figure 5 illustrated an instance of real-time dynamic segmentation recorded on 4 November 2010. The segment is characterized by its start time at 05:40:51 and its end time at 05:43:30. When the spatial event correlation reaches unity and the duration between the beginning and end aligns with the typical time frame for the Bed to Toilet activity, it signifies the identification of the corresponding segment for the Bed to Toilet activity.

An example of obtained segments using this approach on 4 November 2010 is given in Table 5.

Table 5. Comparison of Ground Truth and Simulated Segments for Sleeping Activity.

Activity	Ground Truth Segment	Simulated Segment

Sleeping (1)	begin: 00 H:03 Min: 50 S
	end: 05 H:40 Min:43 S
		begin: 00 H:03 Min:50 S
		end: 05 H:40 Min:44 S

Bed to toilet	begin: 05 H:40 Min:51 S
	end: 05 H:43 Min:30 S
		begin: 05 H:40 Min:51 S
		end: 05 H:43 Min:24 S

Sleeping (2)	begin: 05 H:43 Min:45 S
	end: 08 H:01 Min: 12 S
		begin: 05 H:43 Min:45 S
		end: 08 H:01 Min:09 S

Meal Preparation	begin: 08 H:11 Min:09 S
	end: 08 H:27 Min:02 S
		begin 08 H:11 Min:15 S
		end: 08 H:24 Min:48 S

Table 6 shows an extract of the results of classification with a minimum setup of sensors.

Validation is an essential part of evaluating the algorithm's performance. Here, to validate the proposed algorithm, we verify whether it can accurately estimate occupant activities using the F-score indicator [11] In statistical analysis, the F-score defined by Equation (4) is an indicator of the accuracy of a defined test. It is calculated from the precision (P) and recall (R) of the test defined by Equations (2) and (3), where:

$$P = \frac{TP}{TP + FP} \tag{2}$$

$$R = \frac{TP}{TP + FN} \tag{3}$$

$$\text{F-score} = 2 \times \frac{P \times R}{P + R} \tag{4}$$

- TP: represents the True Positive
- FP: represents the False Positive
- FN: represents the False Negative

Table 6. Comparison of Real and Simulated Activities with Time Intervals.

Activity Name	Real Activities	Simulated Activities

Sleeping	from 00:03:50 to 05:40:43
	from 05:43:45 to 08:01:12
		from 00:03:50 to 05:40:27
		from 05:43:45 to 08:00:21

Bed to toilet	from 5:40:51 to 05:43:30	from 05:40:51 to 05:43:24
		from 08:57:48 to 09:02:48
		from 13:32:00 to 13:33:24
		from 15:21:21 to 15:22:49
		from 23:38:36 to 23:42:36

Work	from 15:47:48 to 16:10:33
	from 17:55:24 to 17:57:47
		from 15:47:48 to 16:10:23
		from 17:55:24 to 17:57:46
		from 18:06:57 to 18:09:03

Relax	from 09:29:23 to 09:34:05
	from 14:46:25 to 15:13:24
	from 16:15:57 to 16:21:58
	from 17:06:21 to 17:27:12
	from 17:28:26 to 17:33:54
		from 08:08:38 to 08:10:40
		from 09:30:10 to 09:34:01
		from 14:46:25 to 15:13:25
		from 16:15:57 to 16:18:30
		from 16:18:33 to 16:21:53
		from 17:06:17 to 17:19:50

Eating	from 09:56:41 to 09:59:04
	from 09:59:47 to 10:02:48
	from 15:25:35 to 15:28:42
	from 17:35:16 to 17:37:11
		from 09:36:08 to 09:44:04
		from 09:56:41 to 10:03:17
		from 10:46:25 to 10:48:37
		from 11:35:35 to 11:38:11
		from 15:25:35 to 15:28:39
		from 17:35:16 to 17:37:08
		from 17:44:13 to 17:45:19

Meal Preparation	from 08:11:09 to 08:27:02
	from 08:33:52 to 08:35:45
	from 09:48:52 to 09:53:02
	from 09:54:58 to 09:56:27
	from 15:23:00 to 15:25:33
	from 16:21:59 to 16:31:26
	from 16:32:53 to 16:34:13
	from 16:36:10 to 17:06:00
	from 17:27:13 to 17:27:55
		from 08:16:28 to 08:24:27
		from 09:03:00 to 09:26:41
		from 09:48:54 to 09:50:46
		from 13:54:44 to 14:13:19
		from 16:25:02 to 16:27:24
		from 16:47:37 to 16:48:54
		from 17:50:38 to 17:51:42
		from 18:20:05 to 19:45:14
		from 21:36:36 to 23:25:59

Table 7 shows the evaluation of the quality of segmentation for the sleeping activity using the F-score. In this work, only the example of Sleeping activity is presented. The evaluation of the performance of activities classification is presented in [11].

Table 7. Performance Metrics Detection Algorithm on 4 November 2010.

	Accuracy	Recall	F-Score	Absolute Error
Sleeping (1)	1	0.99	0.99	9 sec
Sleeping (2)	0.99	0.99	0.99	78 sec
Average error	0.995	0.99	0.99	43.5 sec

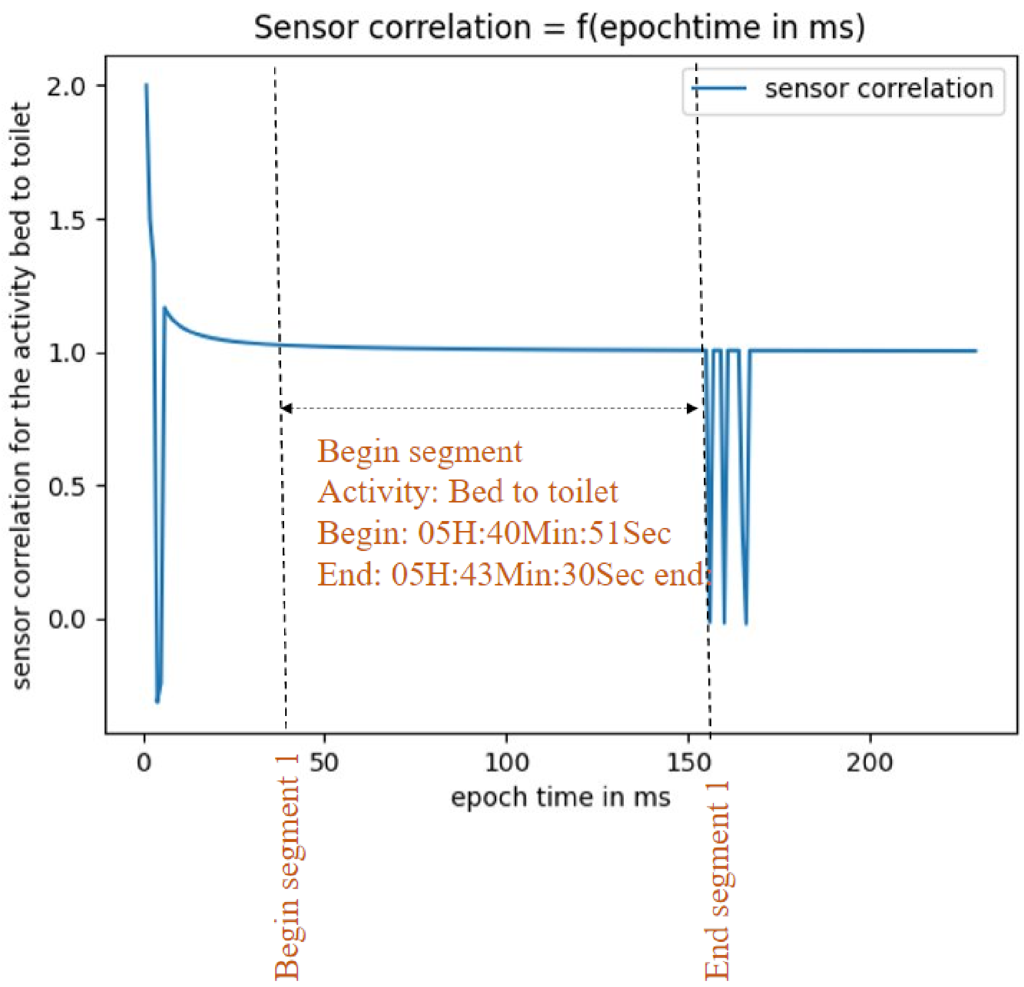


Figure 5. Segment corresponding to the activity Bed to toilet.

As a comparison, the recognition performance (i.e., the average F-score) for 8 activities (sleeping, bed to toilet, eating, housekeeping, relax, work, enter home, and leave home) of the proposed algorithm is 0.766 [11], while this value is equal to 0.73 using the decision tree technique presented in [26], 0.765 using sensor event-based windowing technique [10] and 0.760 using a combination of SWMI (fixed length sensor windows with mutual information based weighting of sensor events) and SWTW (fixed length sensor windows with time-based weighting of the sensor events) techniques [10], and a value of 0.773 using SWTW technique [10] and a value of 0.83 using the Naive Bayes tree technique presented in [26]. Despite certain biases of the dataset and the topology of the house, we are therefore naturally not as good as the algorithms in the field due to our compromises for the proposed simplifications, but we are not bad either.

In the following, the proposed framework is implemented on three hardware architectures. Results are discussed in Section 4 and compared with a desktop PC with OS.

5.2. Implementation on Raspberry Pi and Comparison with Desktop PC

In this section, we test on a PC, because this serves us for comparisons as if we were performing HAR in the cloud. At the same time, the Raspberry is the first "edge" computing, which happens to be very close to a small PC. On PC, we typically find two different OS and the objective is to test if the OS can also have an impact. In this work, we measure the OS performance in terms of activity recognition processing. Figures 6–8 show, respectively, the resulting processing time for implementation on machines with Windows, Linux operating systems and a Raspberry Pi. In these figures, the y-axis represents processing time in seconds, with time units, while the x-axis represents the number of iterations.

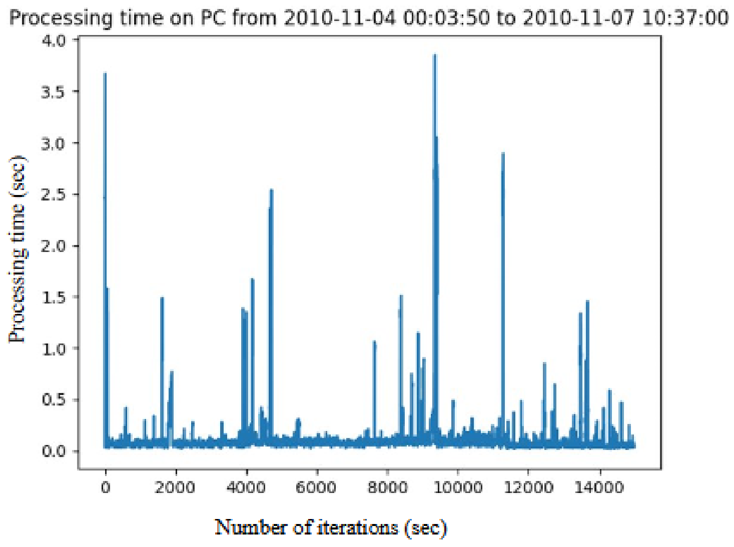


Figure 6. Resulting processing Time on Windows machine.

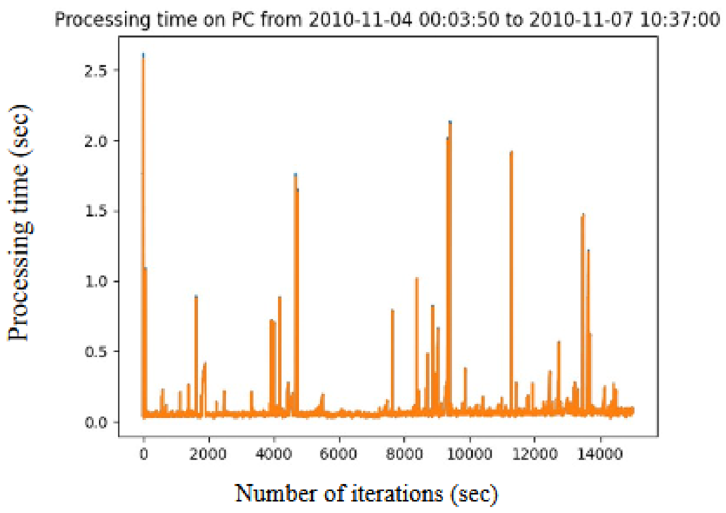


Figure 7. Resulting processing Time on Linux machine.

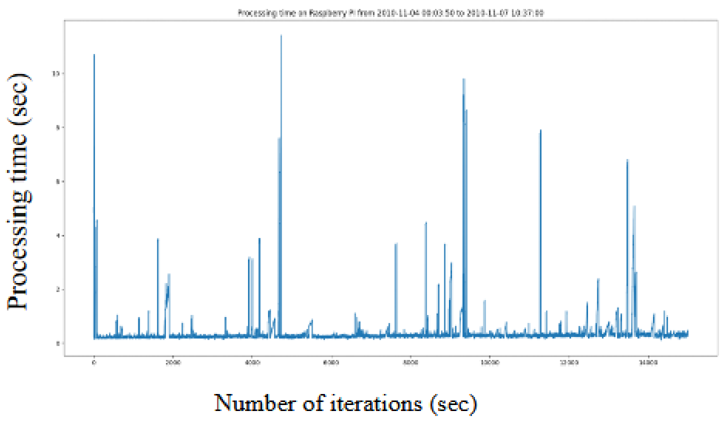


Figure 8. Resulting processing Time on Raspberry Pi.

The common feature among all curves is characterized by a notably low median alongside peaks, indicative of minimal baseline processing interspersed with occasional intensive computational activity. The observed trend in the height of the Y line representing processing time, where it is highest on a Raspberry Pi, lower on a PC with Windows OS, and even lower on a PC with Linux OS, aligns with expectations based on the varying computational power of these devices. The initial peaks could correspond to the period when the algorithm starts and performs pre-processing tasks, such as initial setup and data loading. During the dynamic segmentation phase, the algorithm can be more processing intensive because it analyzes data streams in real-time to identify activities. The peaks could represent times when segmentation is particularly complex. Once the segmentation is completed, the algorithm moves on to classifying the detected activities. This step can also lead to variations in processing time depending on the complexity of the activities to be classified.

Table 8 shows the performance of implementation, in terms of processing time and memory management, on a Raspberry Pi during Period 1. These results are compared with those on Windows and Linux machines, respectively.

Table 8. Performance Comparison of Hardware Architectures.

Hardware Architecture	Processing Time	Memory Space
PC (Windows OS)	0.008 s	mean memory = 107.16 min memory = 102.51 max memory = 111.73
PC (Linux OS)	0.012 s	mean memory = 94.21 min memory = 88.33 max memory = 98.10
Raspberry Pi	0.056 s	mean memory = 65.04 min memory = 63.76 max memory = 66.99

Table 9 shows the results obtained for a test period from 4 November 2010 00:03:50.209589 to 7 November 2010 10:37:00.414579.

Table 9. Performance Analysis of Hardware Architectures for Test Period 4 November 2010 to 7 November 2010 from 4 November 2010 00:03:50.209589 to 7 November 2010 10:37:00.414579.

Hardware Architecture	Processing Time	Memory Space
PC (Windows OS)	mean PT = 0.113 min PT = 0.015 max PT = 3.852	mean memory = 138.71 min memory = 75.65 max memory = 149.83
PC (Linux OS)	mean PT = 0.091 min PT = 0.020 max PT = 2.580	mean memory = 99.9 min memory = 97.14 max memory = 107.69
Raspberry Pi 3 B+	mean PT = 0.423 min PT = 0.102 max PT = 11.43	mean memory = 119.78

The observed differences in processing time and memory consumption can be attributed to various factors related to hardware and software optimization. The operating system (OS) itself can introduce overhead. Linux is generally known for its efficiency and low overhead, while Windows may have more background processes and services that could impact processing times. Let us explore possible interpretations for these differences:

- The Raspberry Pi, being a single-board computer with relatively modest hardware compared to a typical desktop or server, exhibits longer processing times due to its lower computational power. A PC often runs on more powerful hardware, which can result in faster processing times, especially for computationally intensive tasks.
- Experiment results show that a PC running a Windows operating system typically consumes more memory (RAM) than a PC running a Linux operating system, and the

memory usage of the Windows PC is close to that of a Raspberry Pi. Windows tends to consume more resources than Linux. Windows operating systems often require more memory and system resources to run efficiently. Linux is characterized by its ability to run on relatively modest hardware, which often results in lower memory usage. This memory does not have the same order of magnitude as the MCUs.

The simulation results on a Raspberry Pi and PC with Windows and Linux OS ensure that:

- *Performance Efficiency*—The fact that the processing time is consistently within the specified constraint (i.e the processing time should be less than the delta time between events), indicates that the system is performing well in terms of speed. The average processing time is significantly below the maximum allowable time. It suggests a good margin of efficiency.
- *Effectiveness under normal operating conditions*—In this work, the system's effectiveness is evaluated only under normal conditions. Such scenarios where the system operates continuously for an extended duration such as cases where activities last longer than expected or how the system responds to unexpected events are not studied because they are not the objective of this paper.
- *Stability and Reliability*—The absence of memory failures (i.e., the risk that the program, at certain times, cannot obtain from the hardware the memory resources it needs) is a positive sign for system stability. It suggests that the system is handling the data processing without encountering memory-related issues. In addition, there are no warning messages or errors in the logs during the tests. The absence of errors is a good sign of system reliability. In fact, when we implement the algorithm on an MCU with C language or with MP, an error message is displayed when a memory overflow is made at time t. This is not the case for an implementation on a Raspberry Pi or a PC.
- *Delta Time Consideration*—Since the processing time is less than the delta time between events, it means that the system can keep up with the pace of incoming events. This is crucial for real-time or near-real-time applications.
- *Scalability*—The experience shows that when varying the quantity of data processed from period 1 to period 2, the system performance remained consistent, which suggests that the system might be scalable. This means it can handle increased loads without a proportional decrease in performance.

We conclude that the system meets the specified constraint and does so consistently across different conditions. It is generally a positive sign. This means that the system is robust and capable of handling the specified workload.

5.3. Implementation on STM32 and ESP32 with MicroPython

The objective of this section is to study the performances of the algorithm on STM32-NUCLEO H753ZIT6 and Espressif ESP32 microcontrollers for the longer activity period (Sleeping in this work). MicroPython is an optimized implementation of the Python 3 programming language that includes a small subset of standard libraries of Python and is optimized to run on microcontrollers and in constrained environments. It is packed full of advanced features such as arbitrary precision integers, interactive prompts, generators, list comprehension, closures, exception handling, and more. It runs on microcontrollers in constrained environments within just 256 k of code space and 16 k of RAM. Table 10 presents the processing time and the memory used by the algorithm coded with MicroPython.

Table 10. Performance Comparison of MicroPython on STM32 and ESP32 for Long Sleeping Period.

	STM32	**ESP32**
Processing time	0.01 s	0.15 s
Memory	9232 octets	9232 octets

From experimental results, the following interpretations can be drawn:

1. For STM32:
   - Period of long activity in the dataset: The device is in a low-power state (sleeping) from t = 00:03 to t = 05:40.
   - Processing time: During each time step within the sleeping period, the algorithm takes 0.01 s to process.
   - Used memory: The amount of memory used by the algorithm is 9232 octets (bytes).
2. For ESP32:
   - Period of Sleeping activity: Similar to STM32, the ESP32 is in a low-power state (sleeping) from t = 00:03 to t = 05:40.
   - Processing time: During each time step within the sleeping period, the algorithm takes 0.15 s to process. This is significantly longer than the processing time on STM32.
   - Used memory: The amount of memory used by the algorithm is the same as on STM32, which is 9232 octets.

From these results, we conclude that the processing time on ESP32 is 15 times longer than on STM32. The processing time difference between ESP32 and STM32 could be influenced by various factors, and the presence or absence of cache memory is one of them. The ESP32 features a two-tiered cache system: an instruction cache and a data cache. Caches are designed to improve memory access times by storing frequently accessed instructions and data. However, the algorithm used here does not benefit much from the cache function. Cache memory in STM32 is smaller compared to ESP32. Also, ESP32s and STM32s have different hardware architectures. ESP32s are based on dual-core microcontrollers and have more hardware resources, including a dual-core Tensilica Xtensa processor, which can increase processing time. The ESP32 microcontroller uses more hardware resources compared to some other microcontrollers such as multiple cores and various peripheral interfaces, there is additional overhead involved in managing and coordinating the utilization of those resources. For example, in the case of the ESP32's dual-core architecture, coordinating tasks between the two cores and ensuring optimal resource allocation can introduce overhead that affects processing time. The coordination between the cores of the ESP32 microcontroller is indeed facilitated by an onboard mini RTOS. This real-time operating system manages task allocation and scheduling across the dual cores, enabling parallel processing capabilities. However, it is important to note that the algorithm employed in the project is not specifically optimized to fully exploit this parallelism. As a result, the processing time is adversely affected, reflecting the need for more efficient utilization of the ESP32's resources.

In addition, development tools, available libraries, and programming environments vary between the two platforms. Some tools or libraries may be optimized for a platform, which may affect performance. In summary, while both devices use the same amount of memory, the significant difference in processing time suggests that the ESP32 might have performance implications for the given human activity recognition algorithm. Figures 9 and 10 show, respectively, the evolution of processing time with a maximum number of records. In these figures, the y-axis represents processing time in seconds, with time units, while the x-axis represents the number of iterations.

We notice that, during the period with a maximum number of records, the processing time always respects the constraint of processing time less than the time difference between two events.

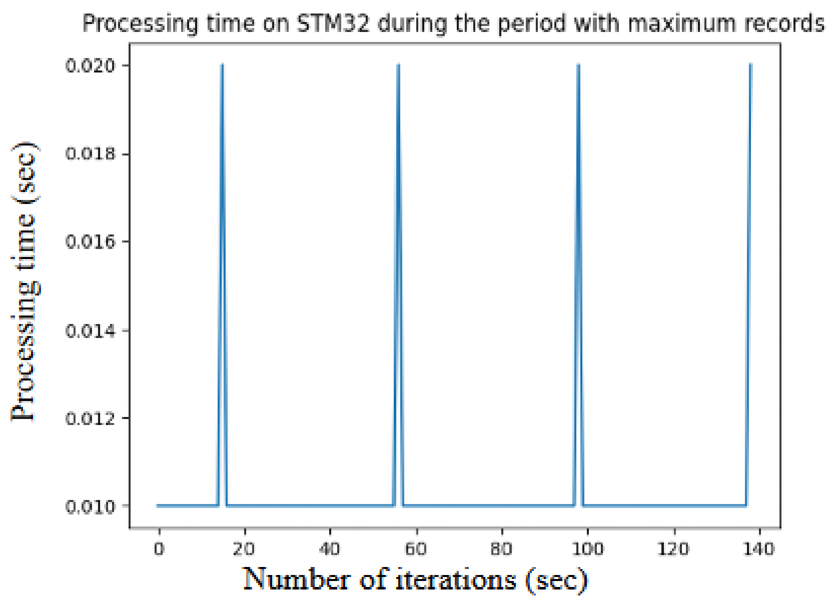


Figure 9. Processing Time evolution in STM32 MCU over a maximum number of records: use of MP programming language.

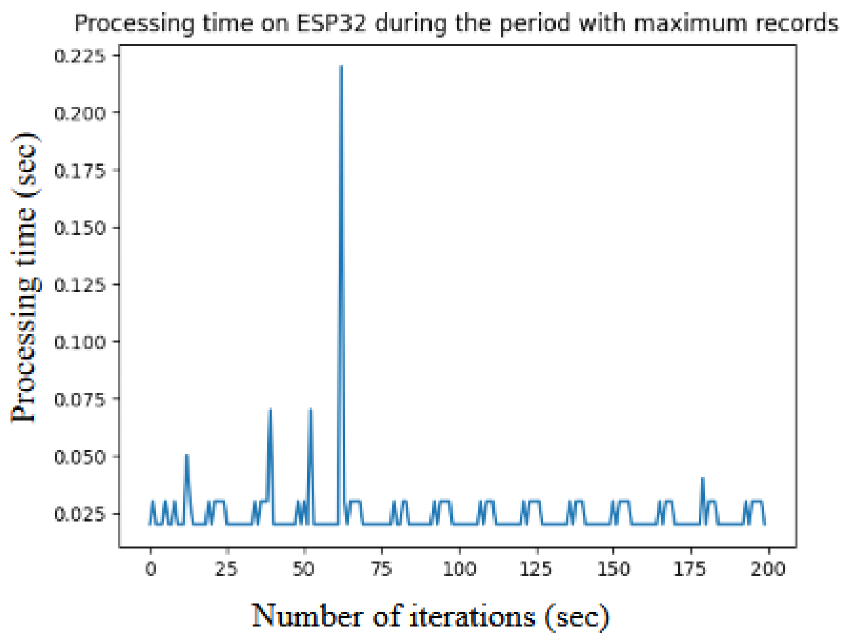


Figure 10. Processing Time evolution in STM32 MCU over a maximum number of records.

5.4. Implementation on STM32 and ESP32 with C Language

The objective of this section is to study the performances of the algorithm implemented in C programming language on STM32-NUCLEO H753ZIT6 and on Espressif ESP32 microcontrollers. Table 11 presents the processing time and the used memory. The objective is to study the performance for the long activity period (Sleeping in this work). Table 11 provides the results of experiments.

Table 11. Performance Comparison of C Programming Language on STM32 and ESP32 for Long Sleeping Period.

	STM32	**ESP32**
Processing time	0.004018 s	0.0022 s
Memory	68.98 kB Flash (3.37% usage) 6.5 KB RAM D1 (1.27% usage)	Used static DRAM = 12,896 bytes Used static IRAM = 48,776 bytes

From experimental results, the following conclusions can be drawn:

- *Processing time*—ESP32 performs better in terms of processing time compared to STM32. The algorithm executes in approximately half the time on the ESP32 (0.0022 s) compared to STM32 (0.004018 s). The C implementation generally shows shorter processing times compared to MicroPython on both STM32 and ESP32. The C implementation on ESP32 still has a shorter processing time (0.0022 s) compared to MicroPython on ESP32 (0.15 s).
- *Memory usage*—The C implementation uses specific memory categories like Flash Memory, RAM D1, static DRAM, and static IRAM. MicroPython uses a unified measure of memory in octets (bytes). However, based on the provided information, the memory usage is similar for both STM32 and ESP32 in C and MicroPython implementations (around 9232 bytes).
- *Performance considerations*—When comparing the processing time related to sampling, segmentation, and classification on STM32 and ESP32 microcontrollers, we have observed that the processing time meets the required constraint. However, we experienced memory failures when exceeding a certain number of data records. In terms of processing time, the C implementation generally outperforms MicroPython in this specific use case.

Memory usage is similar between the two implementations, indicating that the specific language used (C or MicroPython) might not have a significant impact on memory requirements. Additionally, we noted a difference in memory capacity when coding in C compared to MicroPython on STM32. These observations could be explained by:

- The fact that the processing time meets the required constraint is a positive outcome. It indicates that the microcontrollers are capable of handling the processing tasks within the specified time frame. In addition, it is important to assess the consistency of the processing time under different conditions. If the processing time remains within the desired range across various scenarios, it suggests stability and reliability in the system.
- Another important point is the memory capacity difference between C and MicroPython. In fact, this difference is not uncommon. It can be explained by the fact that MicroPython, being a higher-level language, might introduce additional overhead for features such as garbage collector and dynamic typing, potentially leading to increased memory usage. Also, regarding optimization, when working in C, we have more direct control over memory management and optimizations. Moreover, we can fine-tune the code for efficiency, which may result in lower memory usage compared to a higher-level language like MicroPython. In addition, coding in C provides hardware-specific libraries to simplify tasks related to controlling peripherals such as turning on an LED when starting the computation of processing time related to a specific task.
  Since there are no particular project constraints, we can perhaps make a compromise between ease of development (using a higher-level language) and resource efficiency.

5.5. Limits of Microcontrollers Regarding Handling Sampling Data and Activity Recognition

When assessing the performances of the two microcontrollers, the question that arises is the following: what are the on-board limitations concerning data recording and activity classification? To address this, we test the capacity of each microcontroller to handle continuous data recording, determining the maximum duration achievable.

The objective is to explore the onboard capacity of each microcontroller in relation to the sampled data processing rate, thus delimiting the number of activities to be recognized. Since the activity recognition algorithm is based on data sampled every second, by running the algorithm onboard, we noticed that microcontrollers have a limit when processing a large amount of sampled data and this limit differs from one MCU to another. The alternative of increasing the sampling period, thus aiming to reduce the quantity of data to manage influences the quality of classification.

Experience shows that there is a performance difference between the MicroPython-based implementation and the C language implementation when considering the number of records processed per second. For a MicroPython implementation, the algorithm has the capacity to handle up to 750 records, each consisting of dates and times, sensor readings, and corresponding values. MicroPython uses dynamic typing and has automatic memory management (garbage collection). This can simplify code development and reduce the likelihood of memory-related errors. For high-frequency data sampling, the convenience of dynamic typing can be beneficial, allowing for more agile development and easier adaptation to changing data quantity. Also, it is known for its suitability in rapid prototyping and development. The quick iteration provided by MicroPython can be advantageous in scenarios where constant adjustments and improvements are needed, common in applications with high data sampling frequency. This robust data processing per second performance highlights the efficiency and versatility of MicroPython for applications requiring high data sampling frequency.

Conversely, the C language implementation, while powerful, operates on a different scale. Faced with data sampled every second, the microcontroller running the C algorithm demonstrates an ability to manage a more modest number of 17 records. This significant contrast in throughput suggests that, despite the implicit efficiency of the C language, there may be inherent challenges or optimizations to consider when running the algorithm on this particular microcontroller. These results highlight the importance of programming language choice to study the practicality of data-intensive applications on limited hardware platforms.

6. Conclusions

In this study, we explore the implementation of a HAR algorithm on embedded systems, focusing particularly on the STM32 and ESP32 microcontrollers, Raspberry Pi, and a desktop PC with Windows and Linux OS. The key objectives of our investigation included assessing processing time, memory constraints, energy consumption, and language-specific implications on memory usage.

In this work, a real-time HAR framework is used to be implemented firstly on a desktop PC with Windows and Linux OS and then on three hardware architectures (Raspberry Pi, NUCLEO-H753ZI, and ESP32 microcontrollers). Firstly, we evaluate a desktop PC to have a reference point (typical performance that could be obtained with a "cloud" solution), and with some Linux/Windows variants. Then we evaluate what this gives on "edge computing" machines, and we compare. The HAR algorithm is based on three steps: data-processing, real-time dynamic segmentation, which determines the beginning and the end of each segment of activity, and a classification using the concept of a trigger sensor and the habitual duration of activities. The algorithm differs from existing techniques by the fact that it uses data from home automation sensors.

This study reveals that both STM32 and ESP32 microcontrollers demonstrate good performance regarding the processing time constraint, ensuring timely and responsive human recognition. However, a notable concern emerged with respect to memory failures on both microcontrollers when handling an elevated number of data records. This emphasizes the critical importance of efficient memory management in the design of embedded systems. Our analysis indicates that careful consideration of memory allocation and deallocation strategies is essential to mitigate memory-related issues.

Furthermore, the comparison between C and MicroPython on the STM32 microcontroller unveiled a discrepancy in memory capacity. While C exhibited a larger memory capacity, MicroPython introduced additional overhead, resulting in higher memory usage. This underscores the trade-offs between the efficiency of C and the ease of development in a higher-level language such as MicroPython.

In extending our study to include the Raspberry Pi, we conclude that The Raspberry Pi exhibited robust processing capabilities, demonstrating an ability to meet the required constraints on processing time, compared to the STM32 and ESP32 microcontrollers. However, it is noteworthy that the Raspberry Pi, being a more versatile and resource-rich platform, showcased a higher tolerance for memory-intensive tasks compared to the microcontrollers.

Interestingly, while the STM32 and ESP32 excelled in specific aspects, such as real-time responsiveness, the Raspberry Pi excelled in handling larger datasets without encountering memory-related issues. This disparity highlights the trade-offs inherent in choosing the appropriate hardware platform for specific applications within intelligent habitats. The Raspberry Pi's versatility and ample resources position it as an attractive choice for scenarios where computational power and memory availability outweigh strict real-time constraints.

In conclusion, our comparative analysis of human recognition algorithms across the STM32 microcontroller, ESP32 microcontroller, and Raspberry Pi provides a nuanced perspective on the interplay between hardware selection, algorithm performance, and memory considerations. But above all, it shows that doing HAR on the edge is a reasonable strategy. Certainly, certain aspects must be optimized, but using the cloud is not a necessity.

This comparative study provides information regarding the performance and memory characteristics of human recognition algorithms in embedded systems. The identified strengths and limitations serve as a foundation for further research and optimization efforts.

Addressing memory management challenges is essential to the deployment of activity recognition algorithms in smart homes, especially in applications where real-time processing and memory efficiency are paramount. As we advance in the era of smart environments, these findings contribute to the ongoing discourse on optimizing embedded systems for seamless integration into intelligent living spaces. These results contribute to the ongoing discourse on optimizing algorithms for integration into smart living spaces.

Future work will focus on the exploitation of methodological tools such as Keysight technology to measure the current consumption of different devices.

Author Contributions: Writing—original draft, H.N.; Writing—review and editing, C.L. and B.L. All authors have read and agreed to the published version of the manuscript.

Funding: The work is supported by the HARSHEE project, funded by the "France Relance" program and the Delta Dore company.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: Author Houda has spent most of her working time at Delta Dore company. Author Benoit Leduc is an employee of Delta Dore. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. The authors declare that this study received partial funding from Delta Dore. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

Abbreviations

The following abbreviations are used in this manuscript:

HAR	Human Activity Recognition
RAM	Random Access Memory
MP	Micro Python
OS	Operating System
PC	Personal Computer
sec	seconds

References

1. Cicirelli, F.; Fortino, G.; Giordano, A.; Guerrieri, A.; Spezzano, G.; Vinci, A. On the design of smart homes: A framework for activity recognition in home environment. *J. Med. Syst.* **2016**, *40*, 200. [CrossRef] [PubMed]
2. Rashidi, P.; Cook, D.J. Keeping the resident in the loop: Adapting the smart home to the user. *IEEE Trans. Syst. Man-Cybern.-Part Syst. Hum.* **2009**, *39*, 949–959. [CrossRef]
3. Boukhechba, M.; Chow, P.; Fua, K.; Teachman, B.A.; Barnes, L.E. Predicting social anxiety from global positioning system traces of college students: Feasibility study. *JMIR Ment. Health* **2018**, *5*, e10101. [CrossRef]
4. Mazilu, S.; Blanke, U.; Hardegger, M.; Tröster, G.; Gazit, E.; Hausdorff, J.M. GaitAssist: A daily-life support and training system for parkinson's disease patients with freezing of gait. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Toronto, ON, CA, 26 April–1 May 2014; pp. 2531–2540.
5. Boukhechba, M.; Daros, A.R.; Fua, K.; Chow, P.I.; Teachman, B.A.; Barnes, L.E. DemonicSalmon: Monitoring mental health and social interactions of college students using smartphones. *Smart Health* **2018**, *9*, 192–203. [CrossRef]
6. Patel, S.; Park, H.; Bonato, P.; Chan, L.; Rodgers, M. A review of wearable sensors and systems with application in rehabilitation. *J. Neuroeng. Rehabil.* **2012**, *9*, 21. [CrossRef] [PubMed]
7. Avci, A.; Bosch, S.; Marin-Perianu, M.; Marin-Perianu, R.; Havinga, P. Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In Proceedings of the 23th International Conference on Architecture of Computing Systems 2010, Hannover, Germany, 22–23 February 2010; pp. 1–10.
8. Stiefmeier, T.; Roggen, D.; Ogris, G.; Lukowicz, P.; Tröster, G. Wearable activity tracking in car manufacturing. *IEEE Pervasive Comput.* **2008**, *7*, 42–50. [CrossRef]
9. Dehghani, A.; Sarbishei, O.; Glatard, T.; Shihab, E. A quantitative comparison of overlapping and non-overlapping sliding windows for human activity recognition using inertial sensors. *Sensors* **2019**, *19*, 5026. [CrossRef]
10. Krishnan, N.C.; Cook, D.J. Activity recognition on streaming sensor data. *Pervasive Mob. Comput.* **2014**, *10*, 138–154. [CrossRef]
11. Najeh, H.; Lohr, C.; Leduc, B. Dynamic Segmentation of Sensor Events for Real-Time Human Activity Recognition in a Smart Home Context. *Sensors* **2022**, *22*, 5458. [CrossRef]
12. Biagetti, G.; Crippa, P.; Falaschetti, L.; Focante, E.; Madrid, N.M.; Seepold, R.; Turchetti, C. Machine learning and data fusion techniques applied to physical activity classification using photoplethysmographic and accelerometric signals. *Procedia Comput. Sci.* **2020**, *176*, 3103–3111. [CrossRef]
13. Biagetti, G.; Crippa, P.; Falaschetti, L.; Orcioni, S.; Turchetti, C. Human activity recognition using accelerometer and photoplethysmographic signals. In *Intelligent Decision Technologies 2017: Proceedings of the 9th KES International Conference on Intelligent Decision Technologies (KES-IDT 2017)—Part II, Algarve, Portugal, 21–23 June 2017*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 53–62.
14. Mekruksavanich, S.; Jitpattanakul, A. Biometric user identification based on human activity recognition using wearable sensors: An experiment using deep learning models. *Electronics* **2021**, *10*, 308. [CrossRef]
15. Novac, P.E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and deployment of deep neural networks on microcontrollers. *Sensors* **2021**, *21*, 2984. [CrossRef] [PubMed]
16. Zhao, Y.; Yang, R.; Chevalier, G.; Xu, X.; Zhang, Z. Deep residual bidir-LSTM for human activity recognition using wearable sensors. *Math. Probl. Eng.* **2018**, *2018*, 7316954. [CrossRef]
17. Agarwal, P.; Alam, M. A lightweight deep learning model for human activity recognition on edge devices. *Procedia Comput. Sci.* **2020**, *167*, 2364–2373. [CrossRef]
18. Ordóñez, F.J.; Roggen, D. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors* **2016**, *16*, 115. [CrossRef] [PubMed]
19. Alessandrini, M.; Biagetti, G.; Crippa, P.; Falaschetti, L.; Turchetti, C. Recurrent neural network for human activity recognition in embedded systems using ppg and accelerometer data. *Electronics* **2021**, *10*, 1715. [CrossRef]
20. Xu, Y.; Qiu, T.T. Human activity recognition and embedded application based on convolutional neural network. *J. Artif. Intell. Technol.* **2021**, *1*, 51–60. [CrossRef]
21. Stolovas, I.; Suárez, S.; Pereyra, D.; De Izaguirre, F.; Cabrera, V. Human activity recognition using machine learning techniques in a low-resource embedded system. In Proceedings of the 2021 IEEE URUCON, Montevideo, Uruguay, 24–26 November 2021; pp. 263–267.
22. Shakerian, A.; Douet, V.; Shoaraye Nejati, A.; Landry, R., Jr. Real-time sensor-embedded neural network for human activity recognition. *Sensors* **2023**, *23*, 8127. [CrossRef]

23. Najeh, H.; Lohr, C.; Leduc, B. Considering the mutual information criterion for sensor configuration selection in human activity recognition in smart homes. In Proceedings of the BS 2023: 18th Conference of IBPSA, Building Simulation, Shanghai, China, 4–6 September 2023.
24. Xu, Z.; Wang, G.; Guo, X. Online Activity Recognition Combining Dynamic Segmentation and Emergent Modeling. *Sensors* **2022**, *22*, 2250. [CrossRef]
25. Cook, D.J.; Crandall, A.S.; Thomas, B.L.; Krishnan, N.C. CASAS: A smart home in a box. *Computer* **2012**, *46*, 62–69. [CrossRef]
26. Wan, J.; O'grady, M.J.; O'Hare, G.M. Dynamic sensor event segmentation for real-time activity recognition in a smart home context. *Pers. Ubiquitous Comput.* **2015**, *19*, 287–301. [CrossRef]